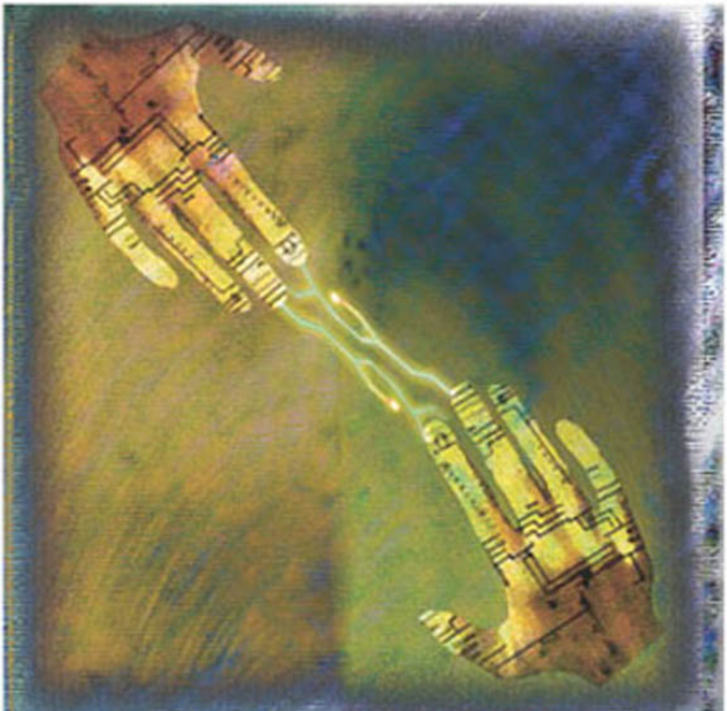
PEER TO PEER COMPUTING

The Evolution of a Disruptive Technology



Ramesh Subramanian
& Brian D. Goodman

Peer-to-Peer Computing: The Evolution of a Disruptive Technology

Ramesh Subramanian
Quinnipiac University, USA

Brian D. Goodman
IBM Corporation, USA

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore

Acquisitions Editor:	Mehdi Khosrow-Pour
Senior Managing Editor:	Jan Travers
Managing Editor:	Amanda Appicello
Development Editor:	Michele Rossi
Copy Editor:	Joyce Li
Typesetter:	Sara Reed
Cover Design:	Lisa Tosheff
Printed at:	Integrated Book Technology

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Library of Congress Cataloging-in-Publication Data

Peer-to-peer computing : the evolution of a disruptive technology / Ramesh Subramanian and Brian D. Goodman, editors.
p. cm.
Includes bibliographical references and index.
ISBN 1-59140-429-0 (hard cover) -- ISBN 1-59140-430-4 (soft cover) -- ISBN 1-59140-431-2 (Ebook)
1. Peer-to-peer architecture (Computer networks) I. Subramanian, Ramesh. II. Goodman, Brian D.
TK5105.525.P443 2004
004.6'5--dc22
2004022155

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Peer-to-Peer Computing: The Evolution of a Distruptive Technology

Table of Contents

Foreword

After decades of growth, we are now about 5% of the way into what the Internet has in store for our business and personal lives. Soon, a billion people will be using the Net, empowering themselves to get what they want, when they want it, from wherever they are. Each day we get closer to a new phase of the Internet that will make today's version seem primitive. Not only will this next-generation Internet be orders of magnitude faster, but it also will be always on, everywhere, natural, intelligent, easy, and trusted.

Fast and reliable connectivity is finally appearing and the competition to provide it is beginning to heat up. Cable, telecom, satellite, and the power grid are each threatening the other and the result will be more speed, improved service, and lower prices. More important than the speed is the always-on connection, which will increase propensities to use online services—and also increase expectations. The impact of WiFi is bigger than coffee shops and train stations. With WiFi chips in handheld devices and the rapid adoption of voice over IP, the Internet becomes available everywhere and a voice conversation becomes just one of the many things you can do while connected. Long distance will no longer mean anything. WiFi will soon be as secure and as fast as today's wired Ethernet. Advanced antenna and radio technologies will ensure ubiquity. With more people always on and having adequate bandwidth, information-oriented e-businesses will lead the charge for the reemergence of the application service provider.

Web services are enabling a global application Web where any and all applications can be linked together seamlessly. Not only will you be able to use frequent flyer points to pay for hotel reservations online, but also to designate from a checkbox on that same hotel Web page the airline from whose frequent-flier program the points should be deducted.

It will soon be clear that Linux is not about "free." It is about achieving scalability, reliability, and security. The world will remain heterogeneous but the underlying operating systems need to be open so that all can see how it works and contribute to it. The "open source" model also will mean more rapid innovation.

Security will no longer be the biggest issue—authentication will. Digital certificates will enable people, computers, handhelds, and applications to interact securely in a distributed Web of trust. With a redesign of e-mail protocols, we also will gain confidence and control over whom we communicate with.

The potential of the Internet is much greater than meets the eye. As the Internet evolves, it will become so pervasive, reliable, and transparent that we will take it for granted. It will be part of our life and, more important, begin to simplify our lives.

One of the many magical elements of the Internet is that every computer connected to it is also connected to every other computer connected to it. There is no central switching office as with the telephone system. Some of the computers on the Net are servers providing huge amounts of information and transactions, but most of the computers are home and office PCs operated by individuals. When one of these individuals connects with another one, it is called a peer-to-peer connection.

Like most technologies that have gained attention on the Internet, peer-to-peer is not a new idea. Peer-to-peer went mainstream during the dot com era of the late 1990s when a teenager named Shawn Fenning appeared on the cover of *Time* magazine after having founded a company called Napster. Napster devised a technology for using peer-to-peer connections to exchange compressed music files (MP3s). Because MP3 music downloaded from the Net sounds the same as music from a CD, and because there are millions of college students with fast Internet connections, the peer-to-peer phenomenon experienced a meteoric growth in popularity.

The recording industry should have anticipated music sharing but instead found itself on the defense and then resorted to legal action to stem the tide. Over the next few years, we will find out if it was too late and the upstarts such as tunes will reshape the music industry.

But peer-to-peer is much bigger than music sharing. It is also information sharing. Not just college students but also business colleagues. Not just music but video conferences. Not just for fun but for serious collaboration in business, government, medicine, and academia. Not just person to person but peer-to-peer networks of many persons—millions, perhaps hundreds of millions. Not just communicating and sharing but combining the computing power of large numbers of computers to find life in outer space, a cure for cancer, or how to untangle the human genome.

It is understandable that the music industry committed itself to an all-out fight against the explosion of peer-to-peer file sharing networks. It is also understandable that many major enterprises have banned peer-to-peer file sharing tools because of a concern that their employees may be importing illegally obtained intellectual property and also out of a justified fear that peer-to-peer networks have spread deadly viruses.

Peer-to-peer is too important to be categorically banned. It needs to be understood and exploited for its merits while policy makers work through the legal and societal issues. Once we truly understand peer-to-peer, we will find that the reality exceeds the hype.

Peer-to-Peer computing: The Evolution of a Disruptive Technology is an important book because it unravels the details of peer-to-peer. This cohesive body of work focuses on the genesis of peer-to-peer—the technologies it is based on, its growth, its adoption in various application areas, and its economic and legal aspects. It also goes deep into peer-to-peer across a broad range of technologies including file sharing, e-mail, grid-based computing, collaborative computing, digital asset management, virtual organizations, new ways of doing business, and the legal implications.

Subramanian and Goodman combine their academic and technology talents to create a compendium filled with practical ideas from existing projects. The book offers a view of peer-to-peer through a series of current articles from academics, IT practitioners, and consultants from around the world.

If you are interested in a complete picture of peer-to-peer technologies, their foundations and development over the years, their applications and business and commercial aspects, then this is a great reference text. Whether you want to gain a basic understanding of peer-to-peer or dive deep into the complex technical aspects, you will find this book a great way to gain ideas into the future of peer-to-peer computing.

John R. Patrick
President, Attitude LLC
Connecticut
May 2004

Preface

In May 1999, Shawn Fanning and Sean Parker created Napster Inc., thus beginning an unforeseen revolution. At the time, Napster was arguably the most controversial free peer-to-peer (P2P) file sharing system the Internet had ever seen. Napster was in many ways an expression of the underground movement that came before it—the world of bulletin board systems, anonymous FTP servers, and the idea of *warez*. Warez refers to pirated software that has been modified or packaged with registration information. Anyone in possession of warez is able to install and run the software as if they had purchased the real license. The successful propagation of pirated software on the Internet is directly attributable to the ease with which loosely associated but highly organized communities can be formed and maintained on the Net. Napster not only answered the need for an easy way to find and share music files, but it also built a community around that concept. People make copies of video, audiotapes, and CDs for personal use all the time. They sometimes share these copies with other people as simply *part of their social mores*. The advent of the MP3 audio format has made the exchange of music all the more easy. People can quickly digitize their music collections and share them with others, using the Internet. Indeed, the Internet provides an extraordinary ability to abuse copyright; it is fast, relatively easy, and with the entry of file sharing software, music can be shared with not just one friend, but with anybody in the world who desires it.

Let's fast-forward to the present time. Now, after endless litigation spearheaded by the Recording Industry Association of America (RIAA), Napster is a for-profit business with strong ties to the music trade—a different avatar from its original revolutionary self.

Chronologies of P2P computing often begin with a reference to Napster. It is the most popular example of just how powerfully one-to-one and one-to-many communications can be realized through computing technology. However, if we look further back, instant messaging was probably an earlier incarnation of P2P. Instant messaging represents a different form of communication. People no longer write as many e-mails—they are engaging in real-time messaging.

Instant messaging provides a compelling hybrid of the telephone and letter writing; all the immediacy of a phone call with all the control of an e-mail. Instant messaging has transformed the Internet landscape and continues to revolutionize the business world.

In fact, from a technology viewpoint, peer-to-peer computing is one of those revisits to past technologies and mind-sets. Often, really great ideas are initially met with little embrace as the environment in which they might flourish lacks nourishment. The concepts that made Napster a reality are not new. Napster simply became an icon of the great P2P underground movement by bringing to reality some of the most basic networking concepts that have existed for a long time. Napster's success was shared by other similar, contemporaneous tools, and the buzz this generated underscored the fact that the time was indeed right for a technology revisit.

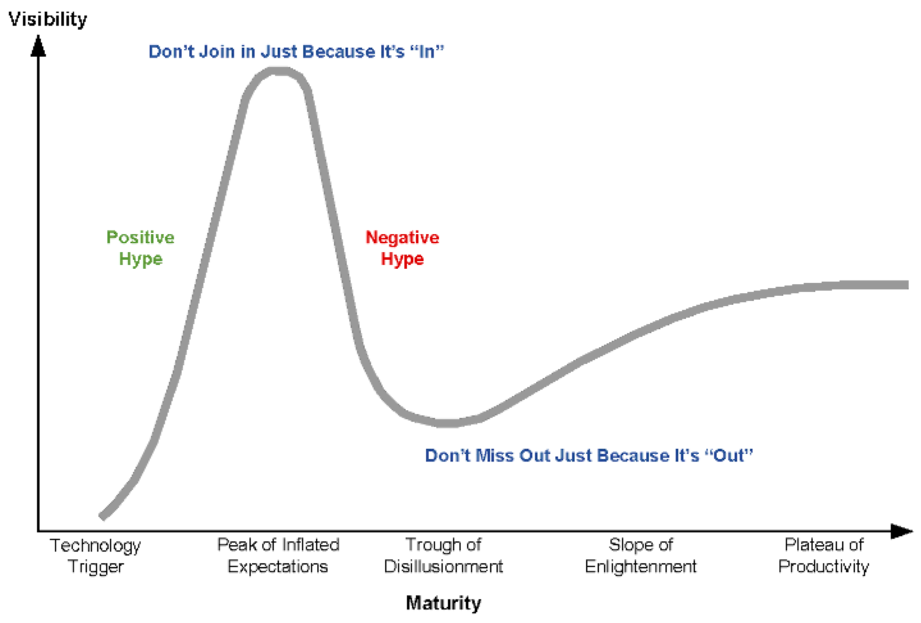
P2P computing has become so commonplace now that some regard it as old news. However, the reality is that we have yet to discover all the ramifications of P2P computing—the maturity of peer systems, the proliferation of P2P applications, and the continually evolving P2P concepts are new.

The goal of this book is to provide insight into this continuing evolution of P2P computing more than four years after its popular and notorious debut. It draws upon recent relevant research from both academia and industry to help the reader understand the concepts, evolution, breadth, and influence of P2P technologies and the impact that these technologies have had on the IT world. In order to explore the evolution of P2P as a disruptive technology, this book has been broken up into three major sections. Section I begins by exploring some of P2P's past—the basic underpinnings, the networks, and the direction they began to take as distribution and data systems. Section II addresses trust, security and law in P2P systems and communities. Section III explores P2P's domain proliferation. It attempts to capture some of the areas that have been irreversibly influenced by P2P approaches, specifically in the area of collaboration, Web services, and grid computing.

Looking at Disruptive Technologies

Disruptive technologies are at the heart of change in research and industry. The obvious challenge is to distinguish the hype from reality. Gartner Research's "Hype Cycles" work (see Figure 1) charts technologies along a life-cycle path, identifying when the technology is just a buzzword through to its late maturation or productivity (Linden and Fenn, 2001, 2003). In 2002, peer-to-peer computing was entering the *Trough of Disillusionment*. This part of the curve represents the technologies' failure to meet the hyped expectations. Every technology en-

Figure 1. Hype cycles



Source: Gartner Research (May 2003)

ters this stage where activities in the space are less visible. Business and venture capitalists continue to spend time and money as the movement climbs the *Slope of Enlightenment* beginning the path of adoption. It is thought that peer-to-peer will plateau anywhere from the year 2007 to 2012. As the peer-to-peer mind-set continues to permeate and flourish across industries, there is a greater need to take a careful reading of the technology pulse. Peer-to-peer represents more than file sharing and decentralized networks. This book is a collection of chapters exemplifying cross-domain P2P proliferation—a check of the P2P pulse.

The Book

Section I of the book deals with the issues of "then and now"—understanding P2P spirit, networks, content distribution, and data storage.

In Chapter I, Detlef Schoder, Kai Fischbach, and Christian Schmitt review the core concepts in peer-to-peer networking. Some of the issues that the authors address are the management of resources such as bandwidth, storage, information, files, and processor cycles using P2P networks. They introduce a model that differentiates P2P infrastructures, P2P applications, and P2P communities. Schoder et al. also address some of the main technical as well as social challenges that need to be overcome in order to make the use of P2P more widespread.

Choon Hoong Ding, Sarana Nutanong, and Rajkumar Buyya continue the overview of P2P computing in Chapter II with a special focus on network topologies used in popular P2P systems. The authors identify and describe P2P architectural models and provide a comparison of four popular file sharing software—namely, Napster, Gnutella, Fasttrack, and OpenFT.

Historically, most peer-to-peer work is done in the area of data sharing and storage. Chapter III focuses on modern methods and systems addressing data management issues in organizations. Dinesh Verma focuses on the data storage problem and describes a peer-to-peer approach for managing data backup and recovery in an enterprise environment. Verma argues that data management systems in enterprises constitute a significant portion of the total cost of management. The maintenance of a large dedicated backup server for data management requires a highly scalable network and storage infrastructure, leading to a major expense. Verma suggests that an alternative peer-to-peer paradigm for data management can provide an approach that provides equivalent performance at a fraction of the cost of the centralized backup system.

Continuing the theme of data storage, Cristina Schmidt and Manish Parashar investigate peer-to-peer (P2P) storage and discovery systems in Chapter IV. They present classification of existing P2P discovery systems, the advantages and disadvantages of each category, and survey existing systems in each class. They then describe the design, operation, and applications of Squid, a P2P information discovery system that supports flexible queries with search guarantees.

Section II of the book shifts the focus to systems and assets, and the issues arising from decentralized networks in diverse areas such as security and law.

In Chapter V, Ross Lee Graham traces the history of peered, distributed networks, and focuses on their taxonomy. He then introduces nomadic networks as implementations of peer-to-peer networks, and discusses the security issues in such networks, and then provides a discussion on security policies that could be adopted with a view to building trust management.

Sridhar Asvathanarayanan takes a data-centered approach in Chapter VI, and details some of the security issues associated with databases in peer networks. Microsoft Windows® is currently one of the most popular operating systems in the world and in turn is a common target environment for peer-to-peer applications, services, and security threats. Asvathanarayanan uses Microsoft® SQL server as an example to discuss the security issues involved in extracting sensitive data through ODBC (open database connectivity) messages and suggests ways in which the process could be rendered more secure. The author underscores that security starts by analyzing and being treated at the technology level.

Michael Bursell offers a more holistic focus on security in Chapter VII by examining the issue of security in peer-to-peer (P2P) systems from the standpoint of trust. The author defines trust, explains why it matters and argues that trust as a social phenomenon. Taking this socio-technical systems view, the author identifies and discusses three key areas of importance related to trust: identity, social contexts, and punishment and deterrence. A better understanding of these areas and the trade-offs associated with them can help in the design, implementation, and running of P2P systems.

In Chapter VIII, law professor Stacey Dogan discusses the challenges that peer-to-peer networks pose to the legal and economic framework of United States Copyright Law. According to Dogan, peer-to-peer networks "debunk the historical assumption that copyright holders could capture their core markets by insisting on licenses from commercial copiers and distributors who actively handled their content." The main way by which peer-to-peer networks accomplish that is through the adherence to communitarian values such as sharing and trust. In this chapter, the author explains why peer-to-peer technology presents such a challenge for copyright, and explores some of the pending proposals to solve the current dilemma.

After addressing the complex and seemingly intractable issues such as security and law as they relate to peer-to-peer networks, we move to Section III of the book, which deals with P2P domain proliferation—the applications of peer-to-peer computing, and the perspectives and influences of peer concepts in the areas of collaboration, Web services, and grid computing.

Peer-to-peer computing has been promoted especially by academics and practitioners alike as the next paradigm in person-to-person collaboration. In Chapter IX, Werner Geyer, Juergen Vogel, Li-Te Cheng, and Michael Muller describe the design and system architecture of such a system that could be used for personal collaboration. Their system uses the notion of shared objects such as a chat mechanism and a shared whiteboard that allow users to collaborate in a rich but lightweight manner. This is achieved by organizing different types of shared artifacts into semistructured activities with dynamic membership, hierarchical object relationships, and synchronous and asynchronous collaboration. The authors present the design of a prototype system and then develop an enhanced consistency control algorithm that is tailored to the needs of this new environment. Finally, they demonstrate the performance of this approach through simulation results.

In Chapter X, Vladimir Soroka, Michal Jacovi, and Yoelle Maarek continue the thread on P2P collaboration and analyze the characteristics that make a system peer-to-peer and offer a P2P litmus test. The authors classify P2P knowledge sharing and collaboration models and propose a framework for a peer-to-peer systems implementation that is an advancement over existing models. They refer to this model as the *second degree peer-to-peer model*, and illustrate it with ReachOut, a tool for peer support and community building.

In Chapter XI, Giorgos Cheliotis, Chris Kenyon, and Rajkumar Buyya introduce a new angle to the discussion of P2P applications and implementations. They argue that even though several technical approaches to resource sharing through peer-to-peer computing have been established, in practice, sharing is still at a rudimentary stage, and the commercial adoption of P2P technologies is slow because the existing technologies do not help an organization decide how best to allocate its resources. They compare this situation with financial and commodity markets, which "have proved very successful at dynamic allocation of different resource types to many different organizations." Therefore they propose that the lessons learned from finance could be applied to P2P implementations. They present 10 basic lessons for resource sharing derived from a financial perspective and modify them by considering the nature and context of IT resources.

In Chapter XII, Xin Li and Aryya Gangopadhyay introduce applications of Web services in bioinformatics as a specialized application of peer-to-peer (P2P) computing. They explain the relationship between P2P and applications of Web services in bioinformatics, state some problems faced in current bioinformatics tools, and describe the mechanism of Web services framework. The authors then argue that a Web services framework can help to address those problems and give a methodology to solve the problems in terms of composition, integration, automation, and discovery.

In Chapter 13, Irwin Boutboul and Dikran Meliksetian describe a method for content delivery within a computational grid environment. They state that the increasing use of online rich-media content, such as audio and video, has created new stress points in the areas of content delivery. Similarly, the increasing size of software packages puts more stress on content delivery networks. New applications are emerging in such fields as bio-informatics and the life sciences that have increasingly larger requirements for data. In parallel, to the increasing size of the data sets, the expectations of end users for shorter response times and better on-demand services are becoming more stringent. Moreover, content delivery requires strict security, integrity, and access control measures.

All those requirements create bottlenecks in content delivery networks and lead to the requirements for expensive delivery centers. The authors argue that the technologies that have been developed to support data retrieval from networks are becoming obsolete, and propose a grid-based approach that builds upon both grid technologies and P2P to solve the content delivery issue. This brings us full circle and exemplifies how at the core of content distribution lies a discernible P2P flavor.

References

Linden, A., & Fenn, J. (2001). 2002 emerging technologies hype cycle: Trigger to peak. Gartner, COM-16-3485, 2.

Linden, A., & Fenn, J. (2003). Understanding Gartner's hype cycles. Gartner, R-20-1971.

Acknowledgments

We would like to thank all the authors for their diverse and excellent contributions to this book. We would also like to thank all of our reviewers and the editors and support staff at the Idea Group Inc.. We would especially like to thank our institutions—IBM Corporation and Quinnipiac University—for their generous understanding and support in providing the time and resources to bring such diverse worlds together. Finally, we would like to express our deepest gratitude to our families, friends, and colleagues for all their support during this project.

Ramesh Subramanian
North Haven, Connecticut

Brian D. Goodman
New York, New York

May 2004

Section I

Then and Now: Understanding P2P Spirit, Networks, Content Distribution and Data Storage

Chapter I

Core Concepts in Peer-to-Peer Networking

Detlef Schoder
University of Cologne, Germany

Kai Fischbach
University of Cologne, Germany

Christian Schmitt
University of Cologne, Germany

Abstract

This chapter reviews core concepts of peer-to-peer (P2P) networking. It highlights the management of resources, such as bandwidth, storage, information, files, and processor cycles based on P2P networks. A model differentiating P2P infrastructures, P2P applications, and P2P communities is introduced. This model provides a better understanding of the different perspectives of P2P. Key technical and social challenges that still limit the potential of information systems based on P2P architectures are discussed.

Introduction

Peer-to-peer (P2P) has become one of the most widely discussed terms in information technology (Schoder, Fischbach, &Teichmann, 2002; Shirky, Truelove, Dornfest, Gonze, & Dougherty, 2001). The term *peer-to-peer* refers to the concept that in a network of equals (peers) using appropriate information and communication systems, two or more individuals are able to spontaneously collaborate without necessarily needing central coordination (Schoder & Fischbach, 2003). In contrast to client/server networks, P2P networks promise improved scalability, lower cost of ownership, self-organized and decentralized coordination of previously underused or limited resources, greater fault tolerance, and better support for building ad hoc networks. In addition, P2P networks provide opportunities for new user scenarios that could scarcely be implemented using customary approaches.

This chapter is structured as follows: The first paragraph presents an overview of the basic principles of P2P networks. Further on, a framework is introduced which serves to clarify the various perspectives from which P2P networks can be observed: P2P infrastructures, P2P applications, P2P communities. The following paragraphs provide a detailed description of each of the three corresponding levels. First, the main challenges—namely, interoperability and security—of P2P infrastructures, which act as a foundation for the above levels, are discussed. In addition, the most promising projects in that area are highlighted. Second, the fundamental design approaches for implementing P2P applications for the management of resources, such as bandwidth, storage, information, files, and processor cycles, are explained. Finally, socioeconomic phenomena, such as free-riding and trust, which are of importance to P2P communities, are discussed. The chapter concludes with a summary and outlook.

P2P Networks: Characteristics and a Three-Level Model

The shared provision of distributed resources and services, decentralization and autonomy are characteristic of P2P networks (M. Miller, 2001; Barkai, 2001; Aberer & Hauswirth, 2002, Schoder & Fischbach, 2002; Schoder et al., 2002; Schollmeier, 2002):

1. Sharing of distributed resources and services: In a P2P network each node can provide both client and server functionality, that is, it can act as both a

provider and consumer of services or resources, such as information, files, bandwidth, storage and processor cycles. Occasionally, these network nodes are referred to as servents—derived from the terms client and server.

2. Decentralization: There is no central coordinating authority for the organization of the network (setup aspect) or the use of resources and communication between the peers in the network (sequence aspect). This applies in particular to the fact that no node has central control over the other. In this respect, communication between peers takes place directly.

   Frequently, a distinction is made between pure and hybrid P2P networks. Due to the fact that all components share equal rights and equivalent functions, pure P2P networks represent the reference type of P2P design. Within these structures there is no entity that has a global view of the network (Barkai, 2001, p. 15; Yang & Garcia-Molina, 2001). In hybrid P2P networks, selected functions, such as indexing or authentication, are allocated to a subset of nodes that as a result, assume the role of a coordinating entity. This type of network architecture combines P2P and client/server principles (Minar, 2001, 2002).

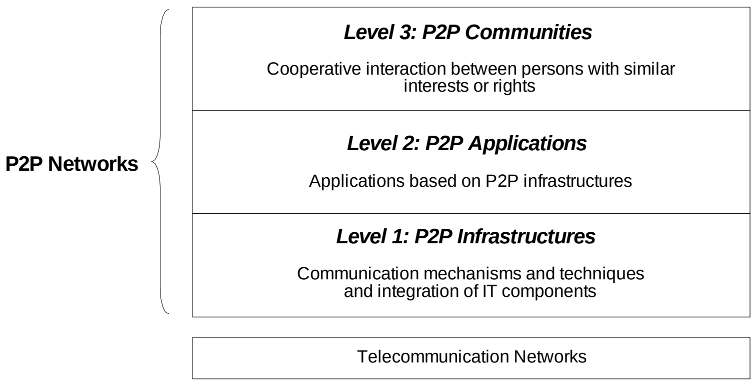
3. Autonomy: Each node in a P2P network can autonomously determine when and to what extent it makes its resources available to other entities.

On the basis of these characteristics, P2P can be understood as one of the oldest architectures in the world of telecommunication (Oram, 2001). In this sense, the Usenet, with its discussion groups, and the early Internet, or ARPANET, can be classified as P2P networks. As a result, there are authors who maintain that P2P will lead the Internet back to its origins—to the days when every computer had equal rights in the network (Minar & Hedlund, 2001).

Decreasing costs for the increasing availability of processor cycles, bandwidth, and storage, accompanied by the growth of the Internet have created new fields of application for P2P networks. In the recent past, this has resulted in a dramatic increase in the number of P2P applications and controversial discussions regarding limits and performance, as well as the economic, social, and legal implications of such applications (Schoder et al., 2002; Smith, Clippinger, & Konsynski, 2003). The three level model presented below, which consists of P2P infrastructures, P2P applications, and P2P communities, resolves the lack of clarity in respect to terminology, which currently exists in both theory and practice.

Level 1 represents P2P infrastructures. P2P infrastructures are positioned above existing telecommunication networks, which act as a foundation for all levels. P2P infrastructures provide communication, integration, and translation functions between IT components. They provide services that assist in locating

Figure 1. Levels of P2P networks


Level 3: P2P Communities
Cooperative interaction between persons with similar interests or rights
Level 2: P2P Applications
Applications based on P2P infrastructures
Level 1: P2P Infrastructures
Communication mechanisms and techniques and integration of IT components
P2P Networks
Telecommunication Networks

and communicating with peers in the network and identifying, using, and exchanging resources, as well as initiating security processes such as authentication and authorization.

Level 2 consists of P2P applications that use services of P2P infrastructures. They are geared to enable communication and collaboration of entities in the absence of central control.

Level 3 focuses on social interaction phenomena, in particular, the formation of communities and the dynamics within them.

In contrast to Levels 1 and 2, where the term *peer* essentially refers to technical entities, in Level 3 the significance of the term *peer* is interpreted in a nontechnical sense (peer as person).

P2P Infrastructures

The term *P2P infrastructures* refers to mechanisms and techniques that provide communication, integration, and translation functions between IT components in general, and applications, in particular. The core function is the provision of interoperability with the aim of establishing a powerful, integrated P2P infrastructure. This infrastructure acts as a "P2P Service Platform" with standardized APIs and middleware which in principle, can be used by any application (Schoder & Fischbach, 2002; Shirky et al., 2001; Smith et al., 2003).

Among the services that the P2P infrastructure makes available for the respective applications, security has become particularly significant (Barkai, 2001). Security is currently viewed as the central challenge that has to be resolved if P2P networks are to become interesting for business use (Damker, 2002).

Interoperability

Interoperability refers to the ability of any entity (device or application) to speak to, exchange data with, and be understood by any other entity (Loesgen, n.d.). At present, interoperability between various P2P networks scarcely exists. The developers of P2P systems are confronted with heterogeneous software and hardware environments as well as telecommunication infrastructures with varying latency and bandwidth. Efforts are being made, however, to establish a common infrastructure for P2P applications with standardized interfaces. This is also aimed at shortening development times and simplifying the integration of applications in existing systems (Barkai, 2001; Wiley, 2001). In particular, within the World Wide Web Consortium (W3C) (W3C, 2004) and the Global Grid Forum (GGF, n.d.) discussions are taking place about suitable architectures and protocols to achieve this aim. Candidates for a standardized P2P infrastructure designed to ensure interoperability include JXTA, Magi, Web services, Jabber, and Groove (Baker, Buyya, & Laforenza, 2002; Schoder et al., 2002).

- JXTA is an open platform that aims at creating a virtual network of various digital devices that can communicate via heterogeneous P2P networks and communities. The specification includes protocols for locating, coordinating, monitoring and the communication between peers (Gong, 2001; Project JXTA, n.d.).
- Magi is designed to set up secure, platform-independent, collaborative applications based on Web standards. A characteristic of Magi is the shared use of information and the exchange of messages between devices of any type, in particular, handheld devices (Bolcer et al., 2000).
- Web services are frequently classified as an application area in the context of P2P. However, they represent a complementary conceptual design, which can be used as one of the technological foundations for P2P applications. Evidence of this, for example, is the growing change in direction of the middleware initiatives of the National Science Foundation and the Global Grid Forum toward Web services. Both initiatives enjoy wide acceptance in both research and practice and are leading the way in the continuing development of grid computing. With their own initiatives, important players in the market, such as Microsoft with .NET (Dasgupta, 2001), IBM with WebSphere, and Sun with SunONE, are pushing forward the development of Web services. Key technologies of Web services are

the Extensible Markup Language as a file format, the Simple Object Access Protocol for communication, the Web Services Description Language for the description of services, the Web Services Flow Language for the description of work flows, and Universal Description, Discovery, and Integration for the publication and location of services (Baker et al., 2002; Bellwood et al., 2003; Web services activity, 2004; Wojciechowski & Weinhardt, 2002).

- The Jabber Open Source Project (Jabber, 2004) is aimed at providing added value for users of instant messaging systems. Jabber functions as a converter, providing compatibility between the most frequently used and incompatible instant messaging systems of providers, such as Yahoo, AOL, and MSN. This enables users of the Jabber network to exchange messages and present information with other peers, regardless of which proprietary instant messaging network they actually use. Within the framework of the Jabber-as-Middleware-Initiative, Jabber developers are currently working on a protocol that is aimed at extending the existing person-to-person functionality to person-to-machine and machine-to-machine communication (J. Miller, 2001).
- The Groove platform provides system services that are required as a foundation for implementing P2P applications. A well-known sample application that utilizes this platform is the P2P Groupware Groove Virtual Office. The platform provides storage, synchronization, connection, security and awareness services. In addition, it includes a development environment that can be used to create applications or to expand or adapt them. This facilitates the integration of existing infrastructures and applications (such as Web Services or .NET Framework) (Edwards, 2002).

Security

The shared use of resources frequently takes place between peers that do not know each other and, as a result, do not necessarily trust each other. In many cases, the use of P2P applications requires granting third parties access to the resources of an internal system, for example, in order to share files or processor cycles. Opening an information system to communicate with, or grant access to, third parties can have critical side effects. This frequently results in conventional security mechanisms, such as firewall software, being circumvented. A further example is communication via instant messaging software. In this case, communication often takes place without the use of encryption. As a result, the security goal of confidentiality is endangered. Techniques and methods for providing authentication, authorization, availability checks, data integrity, and confidentiality are among the key challenges related to P2P infrastructures (Damker, 2002).

A detailed discussion of the security problems which are specifically related to P2P, as well as prototypical implementations and conceptual designs can be found in Barkai (2001), Damker (2002), Udell, Asthagiri, and Tuvell (2001), Groove Networks (2004), Grid Security (2004), Foster, Kesselman, Tsudic, and Tuecke (1998), and Butler et al. (2000).

P2P Applications: Resource Management Aspects

In the respective literature, P2P applications are often classified according to the categories of instant messaging, file sharing, grid computing and collaboration (Schoder & Fischbach, 2002; Shirky et al., 2001). This form of classification has developed over time and fails to make clear distinctions. Today, in many cases the categories can be seen to be merging. For this reason, the structure of the following sections is organized according to resource aspects, which in our opinion, are better suited to providing an understanding of the basic principles of P2P networks and the way they function. Primary emphasis is placed on providing an overview of possible approaches for coordinating the various types of resources, that is, information, files, bandwidth, storage, and processor cycles in P2P networks.

Information

The following sections explain the deployment of P2P networks using examples of the exchange and shared use of presence information, of document management, and collaboration.

- Presence information: Presence information plays a very important role in respect to P2P applications. It is decisive in the self-organization of P2P networks because it provides information about which peers and which resources are available in the network. It enables peers to establish direct contact to other peers and inquire about resources. A widely distributed example of P2P applications that essentially use presence information are instant messaging systems. These systems offer peers the opportunity to pass on information via the network, such as whether they are available for communication processes. A more detailed description of the underlying architecture of instant messaging system can be found in Hummel (2002).

The use of presence information is interesting for the shared use of processor cycles and in scenarios related to omnipresent computers and information availability (ubiquitous computing). Applications can independently recognize which peers are available to them within a computer grid and determine how intensive computing tasks can be distributed among idle processor cycles of the respective peers. Consequently, in ubiquitous computing environments it is helpful if a mobile device can independently recognize those peers which are available in its environment, for example in order to request Web Services, information, storage or processor cycles. The technological principles of this type of communication are discussed in Wojciechowski and Weinhardt (2002).

- Document management: Customary, Document Management Systems (DMS), which are usually centrally organized, permit shared storage, management, and use of data. However, it is only possible to access data that have been placed in the central repository of the DMS. As a result, additional effort is required to create a centralized index of relevant documents. Experience shows that a large portion of the documents created in a company are distributed among desktop PCs, without a central repository having any knowledge of their existence. In this case, the use of P2P networks can be of assistance. For example, by using the NextPage-NXT 4 platform, it is possible to set up networks that create a connected repository from the local data on the individual peers (Next Page Inc., n.d.). Indexing and categorization of data is accomplished by each peer on the basis of individually selected criteria.

  In addition to linking distributed data sources, P2P applicationa can offer services for the aggregation of information and the formation of self-organized P2P knowledge networks. Opencola (Leuf, 2002) was one of the first P2P applications that offer their users the opportunity to gather distributed information in the network from the areas of knowledge that interest them. For this purpose, users create folders on their desktop that are assigned keywords that correspond to their area of interest. Opencola then searches the knowledge network independently and continuously for available peers that have corresponding or similar areas of knowledge without being dependent on centrally administered information. Documents from relevant peers are analyzed, suggested to the user as appropriate, and automatically duplicated in the user's folder. If the user rejects respective suggestions, the search criteria are corrected. The use of Opencola results in a spontaneous networking of users with similar interests without a need for a central control.

- Collaboration: P2P groupware permits document management at the level of closed working groups. As a result, team members can communicate synchronously, conduct joint online meetings, and edit shared documents, either synchronously or asynchronously. In client/server-based groupware a corresponding working area for the management of central data has to be set up and administered on the server for each working group. In order to avoid this additional administration task, P2P networks can be used for collaborative work. The currently best-known application for collaborative work based on the principles of P2P networks is Groove Virtual Office. This system offers similar functions (instant messaging, file sharing, notification, co-browsing, whiteboards, voice conferences, and data bases with real-time synchronization) to those of the widely used client/server-based Lotus products, Notes, Quickplace, and Sametime, but does not require central data management. All of the data created are stored on each peer and are synchronized automatically. If peers cannot reach each other directly, there is the option of asynchronous synchronization via a directory and relay server. Groove Virtual Offce offers users the opportunity to set up so-called shared spaces that provide a shared working environment for virtual teams formed on an ad hoc basis, as well as to invite other users to work in these teams. Groove Virtual Office can be expanded by system developers. A development environment, the Groove Development Kit, is available for this purpose (Edwards, 2002).

Files

File sharing is probably the most widespread P2P application. It is estimated that as much as 70% of the network traffic in the Internet can be attributed to the exchange of files, in particular music files (Stump, 2002) (more than one billion downloads of music files can be listed each week [Oberholzer & Strumpf, 2004]). Characteristic of file sharing is that peers that have downloaded the files in the role of a client subsequently make them available to other peers in the role of a server. A central problem for P2P networks in general, and for file sharing in particular, is locating resources (lookup problem) (Balakrishnan, Kaashoek, Karger, Morris, & Stoica, 2003). In the context of file sharing systems, three different algorithms have developed: the flooded request model, the centralized directory model, and the document routing model (Milojicic et al., 2002). These can be illustrated best by using their prominent implementations—Gnutella, Napster, and Freenet.

P2P networks that are based on the Gnutella protocol function without a central coordination authority. All peers have equal rights within the network. Search requests are routed through the network according to the flooded request model which means that a search request is passed on to a predetermined number of peers. If they cannot answer the request, they pass it on to other nodes until a predetermined search depth (ttl = time-to-live) has been reached or the requested file has been located. Positive search results are then sent to the requesting entity which can then download the desired file directly from the entity that is offering it. A detailed description of searches in Gnutella networks, as well as an analysis of the protocol, can be found in Ripeanu, Foster, and Iamnitchi (2002) and Ripeanu (2001). Due to the fact that the effort for the search, measured in messages, increases exponentially with the depth of the search, the inefficiency of simple implementations of this search principle is obvious. In addition, there is no guarantee that a resource will actually be located. Operating subject to certain prerequisites (such as nonrandomly structured networks), numerous prototypical implementations (for example, Aberer et al., 2003; Crowcroft & Pratt, 2002; Dabek et al., 2001; Druschel & Rowstron, 2001; Nejdl, et al. 2003; Pandurangan & Upfal, 2001; Ratnasamy, Francis, Handley, Karp, & Shenker, 2001; Lv, Cao, Cohen, Li, & Shenker, 2002; Zhao et al., 2004) demonstrate how searches can be effected more “intelligently” (see, in particular, Druschel, Kaashoek, & Rowstron [2002], and also Aberer & Hauswirth [2002] for a brief overview). The FastTrack protocol enjoys widespread use in this respect. It optimizes search requests by means of a combination of central supernodes which form a decentralized network similar to Gnutella.

In respect of its underlying centralized directory model, the early Napster (Napster, 2000) can be viewed as a nearly perfect example of a hybrid P2P system in which a part of the infrastructure functionality, in this case the index service, is provided centrally by a coordinating entity. The moment a peer logs into the Napster network, the files that the peer has available are registered by the Napster server. When a search request is issued, the Napster server delivers a list of peers that have the desired files available for download. The user can obtain the respective files directly from the peer offering them.

Searching for and storing files within the Freenet network (Clarke, Miller, Hong, Sandberg, & Wiley, 2002; Clarke, 2003) takes place via the so-called document routing model (Milojicic et al., 2002). A significant difference to the models that have been introduced so far is that files are not stored on the hard disk of the peers providing them, but are intentionally stored at other locations in the network. The reason behind this is that Freenet was developed with the aim of creating a network in which information can be stored and accessed anonymously. Among other things, this requires that the owner of a network node does not know what documents are stored on his/her local hard disk. For this reason, files and peers are allocated unambiguous identification numbers. When a file is created, it is transmitted, via neighboring peers, to the peer with the identification number that is numerically closest to the identification number of the file and is stored there. The peers that participate in forwarding the file save the identification number of the file and also note the neighboring peer to which they have transferred it in a routing table to be used for subsequent search requests.

The search for files takes place along the lines of the forwarding of search queries on the basis of the information in the routing tables of the individual peers. In contrast to searching networks that operate according to the flooded request model, when a requested file is located, it is transmitted back to the peer requesting it via the same path. In some applications, each node on this route stores a replicate of the file in order to be able to process future search queries more quickly. In this process, the peers only store files up to a maximum capacity. When their storage is exhausted, files are deleted according to the least recently used principle. This results in a correspondingly large number of replicates of popular files being created in the network, whereas, over time, files that are requested less often are removed (Milojicic et al., 2002).

In various studies (Milojicic et al., 2002), the document routing model has been proven suitable for use in large communities. The search process, however, is more complex than, for example, in the flooded request model. In addition, it can result in the formation of islands—that is, a partitioning of the network in which the individual communities no longer have a connection to the entire network (Clarke et al., 2002; Langley, 2001).

Bandwidth

Due to the fact that the demands on the transmission capacities of networks are continuously rising, in particular due to the increase in large-volume multimedia data, effective use of bandwidth is becoming more and more important. Currently, in most cases, centralized approaches in which files are held on the server of an information provider and transferred from there to the requesting client are primarily used. In this case, a problem arises when spontaneous increases in demand exert a negative influence on the availability of the files due to the fact that bottlenecks and queues develop.

Without incurring any significant additional administration, P2P-based approaches achieve increased load balancing by taking advantage of transmission routes that are not being fully exploited. They also facilitate the shared use of the bandwidth provided by the information providers.

- Increased load balancing: In contrast to client/server architectures, hybrid P2P networks can achieve a better load balancing. Only initial requests for files have to be served by a central server. Further requests can be automatically forwarded to peers within the network, which have already received and replicated these files. This concept is most frequently applied in the areas of streaming (for example, PeerCast ["PeerCast", n.d.], P2P-

Radio [“P2P-Radio”, 2004], SCVI.net [“SCVI.NET”, n.d.]) and video on demand. The P2P-based Kontiki network (Kontiki, n.d.) is pursuing an additional design that will enable improved load balancing. Users can subscribe to information channels or software providers from which they wish to obtain information or software updates. When new information is available the respective information providers forward it to the peers that have subscribed. After receiving the information, each peer instantaneously acts as a provider and forwards the information to other peers. Application areas in which such designs can be implemented are the distribution of eLearning courseware in an intranet (Damker, 2002, p. 218), the distribution of antivirus and firewall configuration updates (for example, Rumor [McAfee, n.d.]) and also updating computer games on peer computers (for example, Descent (Planet DESCENT, n.d.) and Cybiko [Milojicic et al., 2002]).

- Shared use of bandwidth: In contrast to client/server approaches, the use of P2P designs can accelerate the downloading and transport of big files that are simultaneously requested by different entities. Generally, these files are split into smaller blocks. Single blocks are then downloaded by the requesting peers. In the first instance each peer receives only a part of the entire file. Subsequently, the single file parts are exchanged by the peers without a need for further requests to the original source. Eventually, the peers reconstruct the single parts to form an exact copy of the original file. An implementation utilizing this principle can be found in BitTorrent (Cohen, 2003).

Storage

Nowadays, Direct Attached Storage (DAS), Network Attached Storage (NAS), or Storage Area Networks (SAN) are the main design concepts used to store data in a company. These solutions have disadvantages such as inefficient use of the available storage, additional load on the company network, or the necessity for specially trained personnel and additional backup solutions.

However, increased connectivity and increased availability of bandwidth enable alternative forms of managing storage which resolve these problems and require less administration effort. With P2P storage networks, it is generally assumed that only a portion of the disk space available on a desktop PC will be used. A P2P storage network is a cluster of computers, formed on the basis of existing networks, that share all storage available in the network. Well-known approaches to this type of system are PAST (Rowstron & Druschel, 2001), Pasta (Moreton, Pratt, & Harris, 2002), OceanStore (Kubiatowicz et al., 2000), CFS (Dabek, Kasshoek, Karger, Morris, & Stoica, 2001), Farsite (Adya et al., 2002), and Intermemory (Goldberg & Yianilos, 1998). Systems that are particularly suitable for explaining the way in which P2P storage networks work are PAST, Pasta, and OceanStore. They have basic similarities in the way they are constructed and organized. In order to participate in a P2P storage network, each peer receives a public/private key pair. With the aid of a hash function, the public key is used to create an unambiguous identification number for each peer. In order to gain access to storage on another computer, the peer has to either make available some of its own storage, or pay a fee. Corresponding to its contribution, each peer is assigned a maximum volume of data that it can add to the network. When a file is to be stored in the network, it is assigned an unambiguous identification number, created with a hash function from the name or the content of the respective file, as well as the public key of the owner. Storing of the file and searching for it in the network take place in the manner described as the document routing model before. In addition, a freely determined number of file replicates are also stored. Each peer retrieves its own current version of the routing table which is used for storage and searches. The peer checks the availability of its neighbors at set intervals in order to establish which peers have left the network. In this way, new peers that have joined the network are also included in the table.

To coordinate P2P storage networks, key pairs have to be generated and distributed to the respective peers and the use of storage has to be monitored. OceanStore expands the administrative tasks to include version and transaction management. As a rule, these tasks are handled by a certain number of particularly high-performance peers that are also distinguished by a high degree of availability in the network. In order to ensure that a lack of availability on the part of one of these selected peers does not affect the functional efficiency of the entire network, the peers are coordinated via a Byzantine agreement protocol (Castro, 2001). Requests are handled by all available selected peers. Each sends a result to the party that has issued the request. This party waits until a certain number of identical results are received from these peers before accepting the result as correct.

By means of file replication and random distribution of identification numbers to peers using a hash function, the P2P storage network automatically ensures that various copies of the same file are stored at different geographical locations. No additional administration or additional backup solution is required to achieve protection against a local incident or loss of data. This procedure also reduces the significance of a problem which is characteristic of P2P networks: in P2P networks there is no guarantee that a particular peer will be available in the network at a particular point in time (availability problem). In case of P2P storage networks, this could result in settings where no peer is available in the network that stores the file being requested. Increasing the number of replicates stored at various geographical locations can, however, enhance the probability that at least one such peer will be available in the network.

The low administration costs, which result from the self-organized character of P2P storage networks, and the fact that additional backup solutions are seldom required are among the advantages these new systems offer for providing and efficiently managing storage.

Processor Cycles

Recognition that the available computing power of the networked entities was often unused was an early incentive for using P2P applications to bundle computing power. At the same time, the requirement for high-performance computing, that is, computing operations in the field of bio-informatics, logistics, or the financial sector, has been increasing. By using P2P applications to bundle processor cycles, it is possible to achieve computing power that even the most expensive supercomputers can scarcely provide. This is effected by forming a cluster of independent, networked computers in which a single computer is transparent and all networked nodes are combined into a single logical computer. The respective approaches to the coordinated release and shared used of distributed computing resources in dynamic, virtual organizations that extend beyond any single institution, currently fall under the term *grid computing* (Baker et al., 2002; Foster, 2002; Foster & Kesselman, 2004; Foster, Kesselman, & Tuecke, 2002; GGF, n.d.). The term *grid computing* is an analogy to customary power grids. The greatest possible amount of resources, in particular computing power, should be available to the user, ideally unrestricted and not bound to any location—similar to the way in which power is drawn from an electricity socket. The collected works of Bal, Löhr, and Reinefeld (2002) provide an overview of diverse aspects of grid computing.

One of the most widely cited projects in the context of P2P, which, however, is only an initial approximation of the goal of grid computing, is SETI@home (Search for Extraterrestrial Intelligence) (Anderson, 2001). SETI@home is a scientific initiative launched by the University of California, Berkeley, with the goal of discovering radio signals from extraterrestrial intelligence. For this purpose, a radio telescope in Puerto Rico records a portion of the electromagnetic spectrum from outer space. This data is sent to the central SETI@home server in California. There, they take advantage of the fact that the greater part of processor cycles on private and business computers remains idle. Rather than analyzing the data in a costly supercomputer, the SETI server divides the data into smaller units and sends these units to the several million computers made available by the volunteers who have registered to participate in this project. The SETI client carries out the calculations during the idle processor cycles of the participants' computers and then sends back the results. In the related literature, SETI@home is consistently referred to as a perfect example of a P2P application in general, and more specifically, a perfect example of grid computing (Oram, 2001; M. Miller, 2001). This evaluation, however, is not completely accurate, as the core of SETI@home is a classical client/server application, due to the fact that a central server coordinates the tasks of the nodes and sends them task packets. The peers process the tasks they have been assigned and return the results. In this system there is no communication between the individual nodes. SETI@home does have, however, P2P characteristics (Milojicic et al., 2002). The nodes form a virtual community and make resources available in the form of idle processor cycles. The peers are to a large extent autonomous, as they determine if and when the SETI@home software is allowed to conduct computing tasks (Anderson, 2001; Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002). The shared accomplishment of these types of distributed computing tasks, however, is only possible if the analytic steps can be separated and divided into individual data packets.

The vision of grid computing described earlier, however, extends far beyond projects such as SETI@home. At an advanced stage of development, it should not only be possible for each network node to offer its own resources, but it should also be possible for it to take advantage of the resources available in the P2P network. The first implementations suitable for industrial use are already being announced by the big players in the market. It will most probably be quite some time before a generally available, open grid platform is created, due to the fact that suitable middleware architectures and APIs are still in the development phase (Baker et al., 2002). A currently influential initiative is the Globus Project (The Globus Alliance, 2004), which is working on a standardized middleware for grid application and has been greeted with wide acceptance throughout the grid community. The project is being supported by important market players such as IBM, Microsoft, Sun, HP, and NEC.

P2P Communities

The term *virtual community* was introduced by Licklider and Taylor (1968): "[I]n most fields they will consist of geographically separated members, sometimes grouped in small clusters and sometimes working individually. They will be communities not of common location but of common interest." Today, numerous variations and extensions of the original definition can be found in the related literature. Schoberth and Schrott (2001) have identified a minimum consensus among the various definitions according to which common interest, common norms and values, as well as a common interaction platform are the elements that constitute a virtual community. Emotional links, continuity, alternating relationships, and self-determination represent additional qualifying elements. Based on this, we define P2P communities as virtual communities that use P2P applications as communication and interaction platforms to support the communication, collaboration, and coordination of working groups or groups of persons. Virtually no research has been conducted on the extent to which, or if in fact, all of the elements cited are evident or fulfilled in concrete P2P communities. In contrast, it is relatively easy to identify common interests and common infrastructures in the context of P2P communities. Users of file sharing networks, for example, wish to exchange music and operate an interaction platform to do so. This platform is established by the networked entities and the protocols such as FastTrack or Gnutella. In the anything@home networks, users are linked by interests, for example, for the search for extraterrestrial life forms (SETI@home) or for a cure for AIDS (fightaids@home). The existence or the pronounced development and efficiency of common norms and values in P2P networks can scarcely be determined from the currently available research. In this respect, it can at least be assumed that the establishment of common norms and values will increase as the availability of sanctions and reputation mechanisms grows. The early Napster clients, for example, enable users to deny access to their own resources for selected persons who have no resources or only undesired resources to offer, so that free-riding persons can be barred from the community.

It remains open as to whether qualifying elements such as emotional attachment or a feeling of belonging or continual membership in a community exist—or for that matter, whether they are even relevant. On the other hand, the criterion for self-determination is directly fulfilled by the autonomy of peers described as a characteristic of P2P networks at the onset.

Initial steps toward studying P2P communities are currently being undertaken in the areas of the restructuring of value chains by P2P communities, P2P communities as a business model and trust, as well as free riding and accountability. These will be described briefly in the following paragraphs.

- Restructuring of value chains: Hummel and Lechner (2001) use the music industry as an example of the fact that P2P communities can achieve scales that result in changes to the configuration of the value chain, as well as having a tendency to transfer control over individual value creation steps from the central market players to consumers and, in some cases, to new intermediaries.
- Communities as business models: Viewing communities as business models originates from Hagel and Armstrong (Armstrong & Hagel, 1996; Hagel & Armstrong, 1997). These authors broke away from the idea of viewing communities as purely sociological phenomena and observed virtual communities as a business strategy for achieving economic goals. Their research interest in this respect focuses on the question regarding the

extent to which virtual communities can actually be used by individuals with commercial interests (Hummel & Lechner, 2002). It is particularly unclear which monetary and nonmonetary attractions motivate potential members to participate.

- Trust: There are limitations on virtual collaboration structures. This is of particular importance for P2P applications, as trust is an essential prerequisite for opening up an internal system to access by others. Limitations on trust also constitute limitations on collaboration in P2P networks.

  As a result, it is necessary to have designs that allow trust to be built up between communication partners. Reputation can prove to be a very important element for the creation of trust in P2P networks (Lethin, 2001). Reputation is aggregated information regarding the previous behavior of an actor. In P2P networks, it is possible to take advantage of a wide spectrum of central and decentralized reputation mechanisms that have already been discussed and, in part, implemented in the area of electronic commerce (Eggs, 2001; discussed with particular reference to P2P in Eggs et al. [2002]).

- Free riding and accountability: The success of the file sharing communities established by early Napster and other similar applications has a remarkable origin. Individual maximization of usage in P2P communities leads to—with regard to the community—collectively desirable results. This is due to the fact that when a file is downloaded, a replicate of the music file is added to the database of the file sharing community. These dynamics are threatened by free riders by denying access to the downloaded file or moving the file immediately after downloading so that the collective database doesn't increase. Free-riding peers use the resources available in the P2P network, but do not make any resources available (Adar & Hubermann, 2000). For most P2P applications in general, and for file sharing systems in particular, this can create a significant problem and prevent a network from developing its full potential. Free riding reduces the availability of information as well as the level of network performance (Golle, Leyton-Brown, & Mironov, 2001; Ramaswamy & Liu, 2003). A possible solution for overcoming this problem is accountability (Lui, Lang, & Kwok, 2002). It consists of the protocolling and assignment of used or provided resources and the implementation of negative (debits) or positive (credits in the form of money or user rights) incentives. In view of the absence of central control, however, difficult questions arise regarding the acceptance, enforceability, privacy of user data, and so forth, and as a result, the practicality of such measures.

Schoberth and Schrott (2001) have noted a scarcity of empirical research in the area of virtual communities (and consequently, in the area of P2P communities) as well as a lack of models for illustrating the development, interaction, and disintegration processes of virtual communities. There is a clear need for research regarding the development, motivation, stabilization, and control of P2P communities.

Conclusion

The introduction of a three-level model comprising P2P infrastructures, P2P applications, and P2P communities helps to differentiate the discussion regarding P2P networks. The management of significant resources such as information, files, bandwidth, storage, and processor cycles can benefit from the existence of P2P networks.

Initial experience with protocols, applications, and application areas is available and provides evidence of both the weaknesses and the potential of P2P-based resource management. The use of P2P networks for resource management promises advantages such as reduced cost of ownership, good scalability, and support for ad hoc networks.

- Reduced cost of ownership: The expenditure for acquiring and operating infrastructure and communication systems can be lowered by using existing infrastructures and reducing the administration and user costs. For example, P2P storage networks avoid the necessity for operating a central server for the purpose of storing the entire data volume for a network. By utilizing existing resources, the relative costs for each peer are reduced in respect of data storage or distributed computing. In addition, the operation of central or, as the case may be, additional storage servers or high performance computers, is no longer necessary to produce the same total performance within the network. In the context of collaboration, P2P groupware applications, such as Groove Virtual Office, frequently make the central administration of a server or the central distribution of rights when forming working groups obsolete.
- Scalability: Implementations have shown that P2P networks reduce the dependency on focal points and thereby reduce the potential for bottlenecks by spatially distributing information and creating replicates. This enhances their scalability. Hybrid file sharing networks, such as early Napster, have scalability advantages in comparison to client/server approaches. This results from the direct exchange of documents between peers without the

assistance of a server. With this method, early Napster was in a position to serve more than six million users simultaneously. New approaches such as OceanStore and PAST are aimed at providing their services for several billion users with a volume of files greater than 10^{14} (Milojicic et al., 2002, p. 13).

- Ad hoc networks: P2P networks are ideal for the ad hoc networking of peers because they tolerate intermittent connectivity. As a result, it is conceivable that the underlying philosophy of P2P networks will be increasingly utilized by approaches such as grid computing, mobile business, and ubiquitous computing—in particular when the task at hand is to establish communication between spontaneously networked peers or entities (PDAs, mobile telephones, computers, smart devices, or things in general) in the absence of coordinating, central authorities (Chtcherbina & Wieland, 2002).

These advantages are currently counteracted by some disadvantages. Security mechanisms, such as authentication and authorization as well as accountability, are easier to implement in networks with a central server. Equally, the availability of resources and services cannot always be guaranteed in networks with a small number of participants due to their intermittent connectivity. In file sharing networks, for example, in order to provide the desired level of availability, a correspondingly large number of replicates have to be created. This, however, results in increased use of storage and can have a negative effect on the resource advantages that could be otherwise be achieved.

At all three levels, two points become clear: the importance of P2P networks as information system architectures and the pressing need for research and development in this field. To the best knowledge of the authors, there has been no quantitative comparative study to date, which takes P2P networks into account when investigating the advantages of various information system architectures. However, it should be noted that the potential of P2P networks lies in the establishment of new application scenarios and a "simple" comparison of performance would probably be inadequate. According to this potential, the authors expect that knowledge management will especially benefit from P2P-based resource management. When thinking of knowledge as a further resource, we have to distinguish between implicit and explicit knowledge. Implicit knowledge is tied to individuals and therefore is shared face-to-face. Face-to-face communication can be supported very well by P2P knowledge management systems (Tiwana, 2003). P2P groupware, for example, facilitates self-organized and ad hoc networking of knowledge workers and therefore offers different communication channels. Explicit knowledge, however, is represented in documents. In order to enable access to this knowledge, a centralized knowledge management system has to create a repository of all available documents. This organizational effort can be omitted by using P2P knowledge management systems. Due to this, decentralized knowledge repositories enable access to up-to-date and as yet unrecognized knowledge better than centralized approaches can do (Susarla, Liu, & Whinston, 2003).

It remains to be seen which further application scenarios can be established and how effectively and how efficiently decentralized, in principle, self-controlled, P2P networks can fulfill the requirements for fair cost distribution, trustworthiness, and security (Schoder & Fischbach, 2003). In any case, alongside client/server structures, P2P networks offer a self-contained approach for the organization and coordination of resources. They have both advantages and disadvantages that will have to be evaluated in the relevant context.

References

Aberer, K., & Hauswirth, M. (2002). An overview on peer-to-peer information systems. Retrieved November 15, 2004, from http://lsirpeople.epfl.ch/hauswirth/papers/WDAS2002.pdf

Adar, E., & Hubermann, B.A. (2000). Free riding on Gnutella. *First Monday, 5*(10). Retrieved November 15, 2004, from http://www.firstmonday.dk/issues/issue5_10/adar/index.html

Adya, A., Bolosky, W.J., Castro, M., Cermak, G., Chaiken, R., Douceur, J.R., et al. (2002). FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. Retrieved November 15, 2004, from http://research.microsoft.com/sn/Farsite/OSDI2002.pdf

Anderson, D. (2001). SETI@home. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 67–76). Sebastopol, CA: O'Reilly.

Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@home: An experiment in public-resource computing. *Communications of the ACM, 45*(11), 56–61.

Armstrong, A., & Hagel, J. (1996). The real value of on-line communities. *Harvard Business Review, 74,* 134–141.

Baker, M., Buyya, R., & Laforenza, D. (2002). Grids and grid technologies for wide-area distributed computing. *International Journal on Software: Practice & Experience (SPE), 32*(15), 1437–1466.

Bal, H.E., Löhr, K.-P., & Reinefeld, A. (Eds.) (2002). *Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC.

Balakrishnan, H., Kaashoek M.F., Karger, D., Morris, R., & Stoica, I. (2003). Looking up data in P2P systems. *Communications of the ACM, 46*(2), 43–48.

Barkai, D. (2001). *Peer-to-peer computing. Technologies for sharing and collaboration on the net*. Hillsboro, OR: Intel Press.

Bellwood, T., Clément, L., & von Riegen, C. (2003). UDDI Version 3.0.1 - UDDI Spec Technical Committee Specification. Retrieved November 15, 2004, from http://uddi.org/pubs/uddi_v3.htm

Bolcer, G.A., Gorlick, M., Hitomi, P., Kammer, A.S., Morrow, B., Oreizy, P., et al. (2000). Peer-to-peer architectures and the Magi™ open-source infrastructure. Retrieved November 15, 2004, from http://www.endeavors.com/pdfs/ETI%20P2P%20white%20paper.pdf

Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J., et al. (2000). A national-scale authentication infrastructure. *IEEE Computer, 33*(12), 60–66.

Castro, M. (2001). Practical Byzantine fault tolerance. Retrieved November 15, 2004, from http://www.lcs.mit.edu/publications/pubs/pdf/MIT-LCS-TR-817.pdf

Chtcherbina, E., & Wieland, T. (2002). In the beginning was the peer-to-peer. Retrieved November 15, 2004, from http://www.drwieland.de/beginning-p2p.pdf

Clarke, I. (2003). Freenet's Next Generation Routing Protocol. Retrieved November 15, 2004, from: http://freenet.sourceforge.net/index.php?page=ngrouting

Clarke, I., Miller, S.G., Hong, T.W., Sandberg, O., & Wiley, B. (2002). Protecting free expression online with Freenet. *IEEE Internet Computing, 6*(1), 40–49.

Cohen, B. (2003). Incentives Build Robustness in BitTorrent. Retrieved Nevember 15, 2004, from: http://www.bitconjurer.org/BitTorrent/bittorrentecon.pdf

Crowcroft, J., & Pratt, I. (2002). Peer to peer: Peering into the future. *Proceedings of the IFIP-TC6 Networks 2002 Conference,* 1–19.

Dabek, F., Brunskill, E., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I., et al. (2001). Building peer-to-peer systems with Chord. A distributed lookup service. *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, 81–86.

Dabek, F., Kasshoek, M.F., Karger, D., Morris, R., & Stoica, I. (2001). Wide-area cooperative storage with CFS. *Proceedings of the 18th ACM Symposium on Operating Systems Principles,* 202–215.

Damker, H. (2002). Sicherheitsaspekte von P2P-Anwendungen in Unternehmen. In D. Schoder, K. Fischbach, & R. Teichmann (Eds.), *Peer-to-peer – Ökonomische, technologische und juristische Perspektiven* (pp. 209–228). Berlin: Springer.

Dasgupta, R. (2001). *Microsoft .NET as a development platform for peer-to-peer applications*. Intel Corporation.

Druschel, P., & Rowstron, A. (2001). Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany,* 329–350.

Druschel, P., Kaashoek, F., & Rowstron, A. (Eds.). *Peer-to-peer systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7–8, 2002, Revised Papers.* Springer.

Edwards, J. (2002). *Peer-to-peer programming on Groove*. Indianapolis, IN: Addison-Wesley.

Eggs, H. (2001). *Vertrauen im Electronic Commerce. Herausforderungen und Lösungsansätze*. Wiesbaden, Germany: Deutscher Universitätsverlag.

Eggs, H., Sackmann, S., Eymann, T., & Müller, G. (2002). Vertrauengenerierende Institutionen für P2P-Netzwerke. In C. Weinhardt & C. Holtmann (Eds.), *E-Commerce – Netze, Märkte, Technologien* (pp. 43–54). Heidelberg, Germany: Physica.

Foster, I. (2002). The grid: A new infrastructure for 21st century science. *Physics Today, 55*(2). Retrieved November 15, 2004, from http://www.aip.org/pt/vol-55/iss-2/p42.html

Foster, I., & Kesselman, C. (Eds.) (2004). *The grid: Blueprint for a new computing infrastructure* (2nd edition). San Francisco: Morgan Kaufmann.

Foster, I., Kesselman, C., & Tuecke, S. (2002). The anatomy of the grid. In D. Schoder, K. Fischbach, & R. Teichmann (Eds.), *Peer-to-peer – Ökonomische, technologische und juristische Perspektiven* (pp. 119–152). Berlin: Springer.

Foster, I., Kesselman, C., Tsudic, G., & Tuecke, S. (1998). A security architecture for computational grids. *Proceedings of the 5th ACM Conference on Computer and Communication Security,* 83–92.

Global Grid Forum (GGF) (n.d). Retrieved November 15, 2004, from http://www.gridforum.org/

The Globus Alliance (2004). Retrieved November 15, 2004, from http://www.globus.org/

Golle, P., Leyton-Brown, K., & Mironov, I. (2001). Incentives for sharing in peer-to-peer networks. *Proceedings of the ACM Conference on Electronic Commerce, Tampa, FL,* 264–267.

Goldberg, A., & Yianilos, P. (1998). Towards an archival intermemory. In *Proccedings of the IEEE International Forum on Research and Technology Advances in Digital Libraries, Santa Barbara*, CA, 147–156.

Gong, L. (2001). JXTA: A network programming environment. *IEEE Internet Computing, 5*(3), 88–95.

Grid Security Infrastructure (2004). Retrieved November 15, 2004, from http://www-unix.globus.org/toolkit/docs/3.2/gsi/index.html

Groove networks (2003). Retrieved October 28, 2003, from http://groove.net/

Hagel, J., & Armstrong, A. (1997). *Net gain. Expanding markets through virtual communities*. Boston: Harvard Business School Press.

Hummel, J., & Lechner, U. (2001). The community model of content management, a case study of the music industry. *Journal of Media Management, 3*(1), 4–14.

Hummel, J., & Lechner, U. (2002). Business models and system architectures of virtual communities. From a sociological phenomenon to peer-to-peer architectures. *International Journal of Electronic Commerce, 6*(3), 41–53.

Hummel, T. (2002). Instant messaging – Nutzenpotenziale und Herausforderungen. In D. Schoder, K. Fischbach, & R. Teichmann (Eds.), *Peer-to-peer – Ökonomische, technologische und juristische Perspektiven* (pp. 59–70). Berlin: Springer.

Jabber (2004). Retrieved November 15, 2004, from http://www.jabber.org/about/overview.php

Kontiki (n.d.). Retrieved November 15, 2004, from http://kontiki.com/

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels D., et al. (2000). OceanStore: An architecture for global-scale persistent storage. *Proceedings of Architectural Support for Programming Languages and Operating Systems, New York,* 190–201.

Langley, A. (2001). Freenet. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 123–132). Sebastopol, CA: O'Reilly.

Lethin, R. (2001). Reputation. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 341–353). Sebastopol, CA: O'Reilly.

Leuf, B. (2002). *Peer-to-peer. Collaboration and sharing over the Internet*. Boston: Addison-Wesley.

Licklider, J.C.R., & Taylor, W. (1968). The computer as a communication device. In *Science and Technology* (pp. 21–40).

Loesgen, B. (n.d.). XML interoperability. Retrieved November 15, 2004, from http://www.topxml.com/conference/wrox/2000_vegas/Powerpoints/brianl_xml.pdf

Lui, S.M., Lang, K.R., & Kwok, S.H. (2002). Participation incentive mechanisms in peer-to-peer subscription systems. *Proceedings of the 35th Hawaii International Conference on System Sciences,* 302b.

Lv, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th ACM International Conference on Supercomputing,* 84–95.

Miller, J. (2001). Jabber – Conversational technologies. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 77–88). Sebastopol, CA: O'Reilly.

Miller, M. (2001). *Discovering P2P*. San Francisco: Sybex.

Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., et al. (2002). Peer-to-peer computing. Retrieved November 15, 2004, from http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf

Minar, N. (2001). Distributed systems topologies: Part 1. Retrieved November 15, 2004, from http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html

Minar, N. (2002). Distributed systems topologies: Part 2. Retrieved November 15, 2004, from http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html

Minar, N., & Hedlund, M. (2001). A network of peers – Peer-to-peer models through the history of the Internet. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 9–20). Sebastopol, CA: O'Reilly.

Moreton, T., Pratt, I., & Harris, T. (2002). Storage, mutability and naming in Pasta. Retrieved November 15, 2004, from http://www.cl.cam.ac.uk/users/tlh20/papers/mph-pasta.pdf

Napster protocol specification (2000). Retrieved November 15, 2004, from http://opennap.sourceforge.net/napster.txt

Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., & Löser, A. (2003). *Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-to-Peer Networks*. Retrieved November 15, 2004, from: http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2002/www2003_superpeer.pdf

Next Page Inc. (n.d.). NXT 4 build and deployment white paper. Retrieved November 15, 2004, from http://www.nextpage.com/pdfs/collateral/wp/nxt4_build_wp090903lo.pdf

Oberholzer, F., & Strumpf, K.(2004). The Effect of File Sharing on Record Sales - An Empirical Analysis. Retrieved November 15, 2004, from: http://www.unc.edu/~cigar/papers/FileSharing_March2004.pdf

Oram, A. (Ed.) (2001). *Peer-to-peer: Harnessing the benefits of a disruptive technology*. Sebastopol, CA: O'Reilly.

P2P-Radio (2004). Retrieved November 15, 2004, from: http://p2p-radio.sourceforge.net/

Pandurangan, P.R.G., & Upfal, E. (2001). Building low-diameter p2p networks. *Proceedings of the 42nd annual IEEE Symposium on the Foundations of Computer Science*, 1–8.

Planet DESCENT (n.d.). Retrieved October 28, 2003, from http://www.planetdescent.com/

Project JXTA (n.d). Retrieved November 15, 2004, from http://www.jxta.org/

Ramaswamy, L., & Liu, L. (2003). Free riding: A new challenge to peer-to-peer file sharing systems. *Proceedings of the 36th Hawaii International Conference on System Sciences,* 220.

Ratnasamy, S., Francis, P., Handley, H., Karp, R., & Shenker, S. (2001). A scalable content-addressable network. *Proceedings of the ACM SIGCOMM'01 Conference,* 161–172.

Ripeanu, M. (2001). Peer-to-peer architecture case study: Gnutella network. *Proceedings of IEEE 1st International Conference on Peer-to-Peer Computing*.

Ripeanu, M., Foster, I., & Iamnitchi, A. (2002). Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, *6*(1), 51–57.

Rowstron, A., & Druschel, P. (2001). Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Proceedings of ACM SOSP'01*, 188–201.

Schoberth, T., & Schrott, G. (2001). Virtual communities. *Wirtschaftsinformatik, 43*(5), 517–519.

Schoder, D., & Fischbach, K. (2002). Peer-to-peer. *Wirtschaftsinformatik, 44*(6), 587–589.

Schoder, D., & Fischbach, K. (2003). Peer-to-peer prospects. *Communications of the ACM*, *46*(2), 27–29.

Schoder, D., Fischbach, K., & Teichmann, R. (Eds.) (2002). *Peer-to-peer – Ökonomische, technologische und juristische Perspektiven*. Berlin: Springer.

Schollmeier, R. (2002). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. *Proceedings of the First International Conference on Peer-to-Peer Computing*, 27–29.

SCVI.NET(n.d.). Retrieved November 15, 2004, from: http://www.scvi.net/.

Shirky, C., Truelove, K., Dornfest, R., Gonze, L., & Dougherty, D. (Eds.) (2001). *2001 P2P networking overview*. Sebastopol, CA: O'Reilly.

Smith, H., Clippinger, J., & Konsynski, B. (2003). Riding the wave: Discovering the value of P2P technologies. *Communications of the Association for Information Systems, 11*, 94–107.

Stump, M. (2002). Peer-to-peer tracking can save cash: Ellacoya. Retrieved November 15, 2004, from http://www.ellacoya.com/news/pdf/10_07_02_mcn.pdf

Susarla, A., Liu, D., & Whinston, A.B. (2003). Peer-to-peer enterprise knowledge management. In C.W. Holsapple (Ed.), *Handbook on knowledge management 2 – Knowledge directions* (pp. 129–139). Berlin: Springer.

Tiwana, A. (2003). Affinity to infinity in peer-to-peer knowledge platforms. *Communications of the ACM*, *46*(5), 77–80.

Udell, J., Asthagiri, N., & Tuvell, W. (2001). Security. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 354–380). Sebastopol, CA: O'Reilly.

W3C World Wide Web Consortium (2004). Retrieved November 15, 2004, from *http://www.w3.org/*

Web services activity (2004). Retrieved November 15, 2004, from http://www.w3.org/2002/ws/

Wiley, B. (2001). Interoperability through gateways. In A. Oram (Ed.), *Peer-to-peer: Harnessing the benefits of a disruptive technology* (pp. 381–392). Sebastopol, CA: O'Reilly.

Wojciechowski, R., & Weinhardt, C. (2002). Web services und Peer-to-peer-Netzwerke. In D. Schoder, K. Fischbach, & R. Teichmann (Eds.), *Peer-to-peer – Ökonomische, technologische und juristische Perspektiven* (pp. 99–118). Berlin: Springer.

Yang, B., & Garcia-Molina, H. (2001). *Comparing hybrid peer-to-peer systems*. Retrieved November 15, 2004, from http://www-db.stanford.edu/~byang/pubs/hybridp2p_med.pdf

Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., & Kubiatowicz, J. (2004). Tapestry: A Resilient Global-scale Overlay for Service Deployment. In IEEE Journal on Selected Areas in Communications, Vol 22, No. 1.

Chapter II

Peer-to-Peer Networks for Content Sharing

Choon Hoong Ding
The University of Melbourne, Australia

Sarana Nutanong
The University of Melbourne, Australia

Rajkumar Buyya
The University of Melbourne, Australia

Abstract

Peer-to-peer (P2P) systems are popularly used as "file swapping" networks to support distributed content sharing. A number of P2P networks for file sharing have been developed and deployed. Napster, Gnutella, and Fasttrack are three popular P2P systems. This chapter presents a broad overview of P2P computing and focuses on content sharing networks and technologies. It also emphasizes on the analysis of network topologies used in popular P2P systems. In addition, this chapter also identifies and describes architecture models and compares various characteristics of four P2P systems—Napster, Gnutella, Fasttrack, and OpenFT.

Introduction

Peer-to-peer (P2P) content sharing technologies such as Napster, Gnutella, and Kazaa are applications that have been astonishingly successful on the Internet. P2P has gained tremendous public attention through Napster, which is a system supporting music sharing on the Web. It is an emerging and interesting research technology with a promising product base.

Intel P2P working group gave the definition of P2P as "The sharing of computer resources and services by direct exchange between systems" (Kan, 2001). This thus gives P2P systems two main key characteristics:

- Scalability: There is no algorithmic or technical limitation to the size of the system, for example, the complexity of the system should be somewhat constant regardless of the number of nodes in the system.
- Reliability: The malfunction on any given node will not affect the whole system (or maybe even any other nodes).

File sharing networks such as Gnutella are good examples of scalability and reliability. In Gnutella, peers are first connected to a flat overlay network in which every peer is equal. Peers are connected directly without the need of a master server's arrangement and the malfunction of any node does not cause any other nodes in the system to malfunction as well.

P2P can be categorized into two groups by the type of model: pure P2P and hybrid P2P. The pure P2P model, such as Gnutella and Freenet, does not have a central server. The hybrid P2P model, such as Napster, Groove, and Magi, employs a central server to obtain meta-information such as the identity of the peer on which the information is stored or to verify security credentials. In a hybrid model, peers always contact a central server before they directly contact other peers.

P2P Networks Topologies

According to Peter, Tim, Bart, and Piet (2002), all P2P topologies, no matter how different they may be, have one common feature. All file transfers made between peers are always done directly through a data connection that is made between the peer sharing the file and the peer requesting it. The control process prior to the file transfer, however, can be implemented in many ways. As stated by Minar (2001), P2P file sharing networks can be classified into four basic categories: centralized, decentralized, hierarchical, and ring systems. Although these topologies can exist on their own, it is usually the practice for distributed systems to have a more complex topology by combining several basic systems to create what is known now as hybrid systems. We will give a brief introduction to the four basic systems and later delve deeper into the topic of hybrid systems.

Centralized Topology

The concept of a centralized topology shown in Figure 1 is very much based on the traditional client/server model. A centralized server must exist which is used to manage the files and user databases of multiple peers that log onto it (Peter et al., 2002). The client contacts the server to inform it of its current IP address and names of all the files that it is willing to share. This is done every time the application is launched. The information collected from the peers will then be used by the server to create a centralized database dynamically that maps file names to sets of IP addresses.

All search queries are sent to the server, who will perform a search through its locally maintained database. If there is a match, a direct link to the peer sharing the file is established and the transfer executed (Kurose & Ross, 2003). It should be noted here that under no circumstances will the file ever be placed on the server. Examples of applications that make use of such a network are seti@home, folding@home, and Napster (which will be discussed in greater detail in the Napster section).

Figure 1. An illustration of the centralized topology

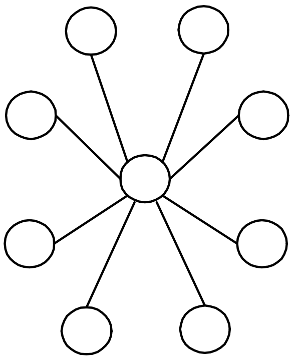
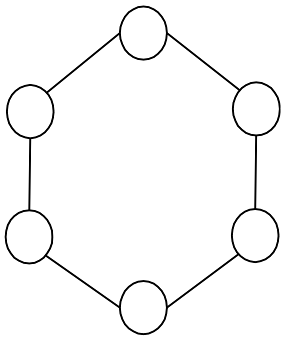


Figure 2. An illustration of the ring topology



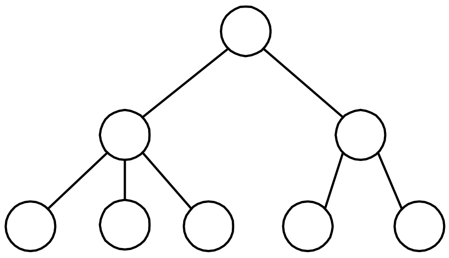
Ring Topology

It should be relatively clear that the drawback of a centralized topology is that the central server can become a bottleneck (when load becomes heavy) and a single point of failure. These are some of the main contributing factors that led to emergence of the ring topology shown in Figure 2. It is made up of a cluster of machines that are arranged in the form of a ring to act as a distributed server (Minar, 2001). This cluster of machines work together to provide better load balancing and higher availability. This topology is generally used when all the machines are relatively nearby on the network, which means that it is most likely owned by a single organization where anonymity is not an issue. Figure 2 shows a simple illustration of a ring topology.

Hierarchical Topology

Hierarchical systems have been in existence since the beginning of human civilization. Everything from a simple family to businesses and to the government basically functions in a hierarchical manner. Today, many of the Internet applications function in a hierarchical environment. The best example of a hierarchical system on the Internet is the Domain Name Service (DNS) (Minar, 2001). Authority flows from the root name servers to the servers of the registered name and so on. This topology is very suitable for systems that require a form of governance that involves delegation of rights or authority. Another good example of a system that makes use of the hierarchical topology is the Certification Authorities (CAs) that certify the validity of an entity on the Internet. The root CA can actually delegate some of its authoritative rights to companies that subscribe to it, so that those companies can, in turn, grant

Figure 3. An illustration of the hierarchy topology



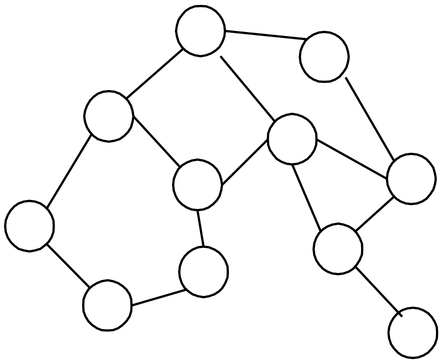
certificates to those that reside underneath it. Figure 3 provides a brief illustration of what a hierarchical system looks like.

Decentralized Topology

In a pure P2P architecture, no centralized servers exist. All peers are equal, thus creating a flat, unstructured network topology (Peter et al., 2002) (see Figure 4). In order to join the network, a peer must first contact a bootstrapping node (node that is always online), which gives the joining peer the IP address of one or more existing peers, officially making it a part of the ever dynamic network. Each peer, however, will only have information about its neighbors, which are peers that have a direct edge to it in the network.

Since there are no servers to manage searches, queries for files are flooded through the network (Kurose & Ross, 2003). The act of query flooding is not exactly the best solution as it entails a large overhead traffic in the network. An example of an application that uses this model is Gnutella. Details of how it searches and shares files in a pure P2P network will be discussed in the Gnutella section.

Figure 4. An illustration of the decentralized topology



Hybrid Topology

Having discussed the basic topologies of P2P networks, we now come to the more complex real-world systems that generally combine several basic topologies into one system. This is known as the hybrid architecture (Yang & Garcia-Moline, 2002). We will discuss several of such examples in this section to give a brief idea of this sort of architecture. In such a system, nodes will usually play more than one role.

Centralized and Ring Topology

This hybrid topology is a very common sight in the world of Web hosting (Minar, 2001). As mentioned previously in the ring topology section, heavy-loaded Web servers usually have a ring of servers that specializes in load balancing and failover. Thus, the servers themselves maintain a ring topology. The clients,

Figure 5. An illustration of the centralized and ring topology

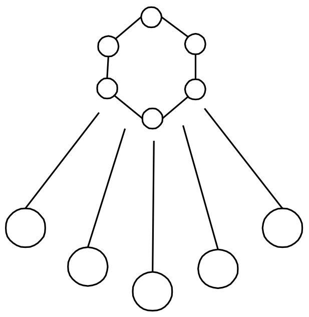
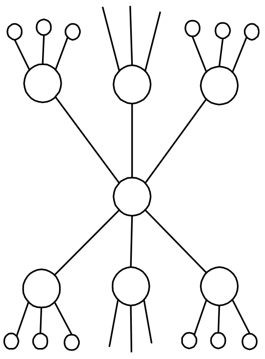


Figure 6. An illustration of the centralized and centralized topology



however, are connected to the ring of servers through a centralized topology (i.e., client/server system) as shown in Figure 5. Therefore, the entire system is actually a hybrid, a mixture between the sturdiness of a ring topology with the simplicity of a centralized system.

Centralized and Centralized Topology

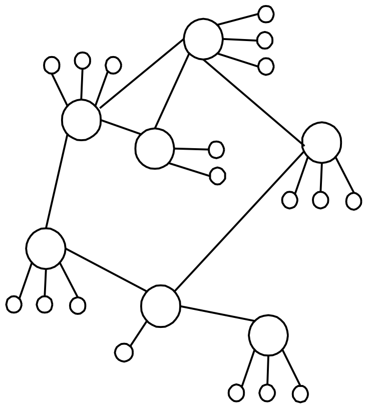
It is often the case that the server of a network is itself a client of a bigger network (Minar, 2001). This sort of hybrid topology shown in Figure 6 is a very common practice in organizations that provide Web services. A simple example that will help illustrate this point is when a Web browser contacts a centralized Web server. The Web server may process and format the results so that they can be presented in HTML format, and in the process of doing that, these servers might themselves contact other servers (for example, database server) in order to obtain the necessary information (Minar, 2002).

Centralized and Decentralized Topology

In this topology, peers that function as group leaders are introduced (Kurose & Ross, 2003). They have been known by many names. Some call them Group Leader Nodes, Super Nodes, or even Ultra Nodes. To keep things simple and consistent with the following sections about Fasttrack (Sharma, 2002), we will refer to them as Super Nodes hence forth.

These Super Nodes perform the task of a centralized server as in the centralized topology, but only for a subset of peers. The Super Nodes themselves are tied together in a decentralized manner. Therefore, this hybrid topology actually

Figure 7. An illustration of the centralized and decentralized topology



introduces two different tiers of control. The first is where ordinary peers connect to the Super Nodes in a centralized topology fashion. The second is where the Super Nodes connect to each other in a decentralized topology fashion as shown in Figure 7.

As with the centralized topology, the Super Nodes maintain a database that maps file names to IP addresses of all peers that are assigned to it (Yang & Garcia-Moline, 2002). It should be noted here that the Super Node's database only keeps track of the peers within its own group. This greatly reduces the scope of peers that it needs to serve. So any ordinary peer with a high speed connection will qualify to be a Super Node. The best example of a P2P application that utilizes such a topology is Kazaa/FastTrack. Another good example of such a topology is the common Internet e-mail. Mail clients have a decentralized relationship to specific mail servers. Like the Super Nodes, these mail servers share e-mails in a decentralized fashion among themselves.

Other Potential Topologies

It should be noted that the hybrid topologies mentioned so far are just the common ones. As can be seen, there is a great deal of different combinations of hybrid topologies that can be achieved from the basic topologies. However, if one is to make too many combinations, the resulting topology may become too complex, hence making it difficult to manage.

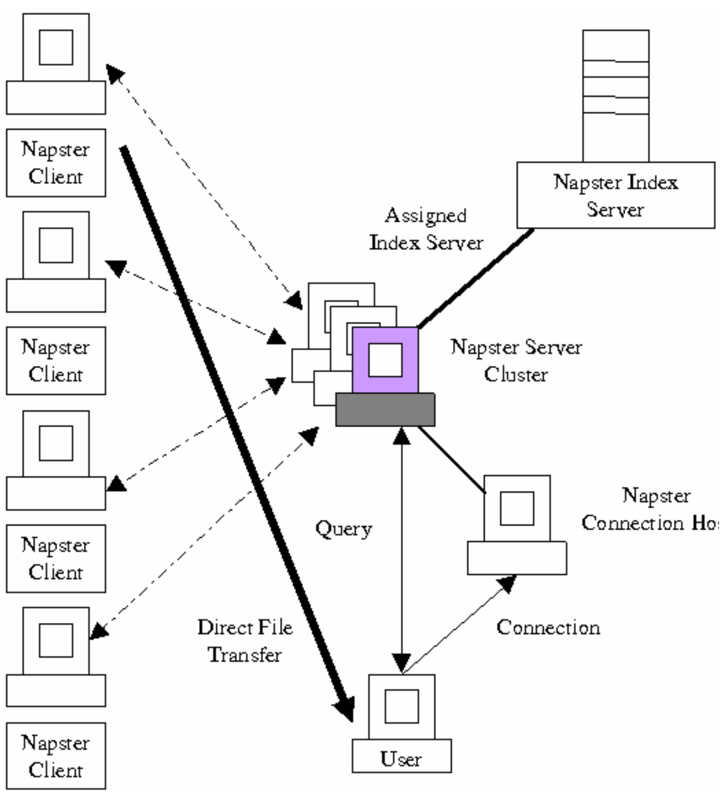
Napster

Napster is a file sharing P2P application that allows people to search for and share MP3 music files through the Internet. It was single-handedly written by a teenager named Shawn Fanning (Tyson, 2000; Shirky, 2001). Not only did he develop the application but he also pioneered the design of a protocol that would allow peer computers to communicate directly with each other. This paved the way for more efficient and complex P2P protocols by other organizations and groups.

The Napster Architecture

The architecture of Napster, shown in Figure 8, is based on the centralized model of P2P file sharing (Hebrank, 2000). It has a server-client structure where there

Figure 8. An illustration of the Napster architecture



is a central server system which directs traffic between individual registered users. The central servers maintain directories of the shared files stored on the respective PCs of registered users of the network. These directories are updated every time a user logs on or off the Napster server network. Clients connect automatically to an internally designated "metaserver" that acts as common connection arbiter. This metaserver assigns at random an available, lightly loaded server from one of the clusters. Servers appear to be clustered about five to a geographical site and Internet feed, and able to handle up to 15,000 users each. The client then registers with the assigned server, providing identity and shared file information for the server's local database. In turn, the client receives information about connected users and available files from the server. Although formally organized around a user directory design, the Napster implementation is very data-centric. The primary directory of users connected to a particular server is only used indirectly, to create file lists of content reported as shared by each node (Barkai, 2001).

Users are almost always anonymous to each other; the user directory is never queried directly. The only interest is to search for content and determine a node from which to download. The directory, therefore, merely serves as a background translation service, from the host identity associated with particular content to the currently registered IP address needed for a download connection to this client. Each time a user of a centralized P2P file sharing system submits a request or search for a particular file, the central server creates a list of files matching the search request by cross-checking the request with the server's database of files belonging to users who are currently connected to the network. The central server then displays that list to the requesting user. The requesting user can then select the desired file from the list and open a direct HTTP link with the individual computer that currently possess that file. The download of the actual file takes place directly, from one network user to the other, without the intervention of the central server. The actual MP3 file is never stored on the central server or on any intermediate point on the network.

The Napster Protocol

Due to the fact that Napster is not an open source application, it was only possible to build up a similar application to reveal the Napster protocol through reverse engineering (Turcan, 2002). In other words, no one will ever be totally sure what the Napster protocol specification is, except for the creator of Napster himself. Project OpenNap has made it possible to run a Napster server on many platforms without using the original Napster application and the index server. The following are the protocol specifications for Napster with reference to Turcan (2002).

Napster works with a central server which maintains an index of all the MP3 files of the peers. To get a file, you have to send a query to this server which sends you the port and IP address of a client sharing the requested file. With the Napster application, it is now possible to establish a direct connection with the host and to download a file.

The Napster protocol also uses a whole lot of different types of messages. Every state of the hosts, acting like clients toward the server, is related to the central Napster server. Thus the Napster protocol makes anonymity impossible. At first, one may think that this is a drawback, but this complex protocol actually makes a lot of services possible. Some examples are:

- Creating Hotlists: Notifying when users of your own hotlist sign on or off the server
- List of ignored user

- Instant Messaging: Sending public or private messages to other users; creating and joining channels of shared interests

Napster Message Data Structures

The format of each message that flow to/from the Napster central server is shown below:

<Length> (2 Bytes)	<Function> (2 Bytes)	<Payload> (n Bytes)

Where:

- Length specifies the length of the payload
- Function defines the message type of the packet (see next paragraph)
- Payload of this portion of the message is a plain ASCII string

Every block of header and payload is separated by "blanks" which make the synchronization of the incoming messages possible. Most of blocks have no fixed length. The blanks make separation of data blocks in incoming bit streams possible.

Initialisation

A registered Napster host, acting like a client, sends to the server a LOGIN message in the following format:

<Nick>	<Password>	<Port>	<Client_Info>	<Link_Type>

Where:

- Nick & Password identify the user
- Port is the port that the client is listening on for data transfer
- Client_Info is a string containing the client-version info
- Link_Type is an integer indicating the client's bandwidth

Table 1. Integer representations for Link_Type

Representation	Bandwidth		Representation	Bandwidth
0	Unknown		1	14.4 kbps
2	28.8 kbps		3	33.6 kbps
4	56.7 kbps		5	64k ISDN
6	128k ISDN		7	Cable
8	DSL		9	T1
10	T3 or greater			

The details of what the integers in Link_Type represent are given in Table 1.

The host's IP address need not be added to the message. However, the server can extract it automatically from the TCP packet in which the message is packed for the transmission.

An unregistered host sends a New User Login which is similar to the format of Login, with the addition of the e-mail address at the end. The server sends a Login Acknowledgment to the client after a successful login. If the nick is registered, the e-mail address given at registration time is returned, else, a dummy value will be returned.

Client Notification of Shared File

With the Client Notification of Shared File message, the client sends successively all the files it wants to share. It uses the following message notification format:

<Filename>	<MD5>	<Size>	<Bitrate>	<Frequency>	<Time>

Where:

- Filename is the name of the file to be shared
- MD5 (Message Digest 5) is the hash value of the shared file. The MD5 algorithm produces a 128-bit "fingerprint" of any file. It is nearly computationally infeasible to produce two messages having the same hash value. The MD5 algorithm is intended to provide any user the possibility to secure the origin of his/her shared file, even if the file is lying in drives of other Napster users
- Size is the file size in bytes
- Bitrate is the bit rate of the MP3 in kbps (kilobits per second)

- Frequency is the sample rate of the MP3 in Hz (Hertz)
- Time is the duration of the music file in seconds

File Request

The downloading client will first issue either a Search or Browse. The first search message has the following format:

<Artist Name>	<Title>	Bit-rate	<Max Results>	<Line-type>	<Frequency>

Where:

- Artist Name is the name of the artist of the MP3 song
- Title is the title of the MP3 song
- Bitrate is the range of bitrates to be used
- Max Results is the maximum number of results
- Link-Type is the range of link types
- Frequency is the range of sample frequencies in Hz

The artist name and the song title are checked from the file name only. Napster does not make use of the ID3 in MP3 files in its search criteria. The payload of the Browse message contains only the nick of the host. It requests a list of the host's shared files.

Response and Browse Response

The server answers respectively with a Search Response or a Browse Response with the formats below:

<Filename>	<MD5>	<Size>	<Bit-rate>	<Frequency>	<Time>	<Nick>	<IP>	<Link-type>

Where:

- Filename is the name of the file that is found
- MD5 is the hash value of the requested file

- Size is the file size in bytes
- Bitrate is the bitrate of the MP3 in kbps
- Frequency is the sample rate of the MP3 in Hz
- Time specifies the length of the file
- Nick is to identify the user who shares the file
- IP is a four-byte integer representing the IP address of the user with the file
- Link-Type refers to the Login Message

Download Request

In order to request a download, a DOWNLOAD REQUEST message is sent to the server. This message has the following payload format:

<Nick>	<Filename>

Where:

- Nick is to identify the user who shares the file
- Filename is the name of the file to be downloaded

Download ACK

The server will answer with a DOWNLOAD ACK containing more information about the file (Line Speed, Port Number, etc.). This message has the following payload format:

<Nick>	<IP>	<Port>	<Filename>	<MD5>	<Link-type>

Where:

- Nick is to identify the user who shares the file
- IP is a four-byte integer representing the IP address of the user with the file
- Port is the port on which the client is listening on for data transfer
- Filename is the name of the file to be downloaded

- MD5 is the hash value of the requested file
- Link-Type is the range of link types

Alternate Download Request

This is like the normal "Download Request." The only difference is that this request is used only when the person sharing the file can only make outgoing TCP connection because of the firewall that is blocking the incoming messages. The ALTERNATE DOWNLOAD REQUEST message should be used to request files from users who have specified their data port as "0" in their login message.

Alternate Download Ack

ALTERNATE DOWNLOAD ACK message is sent to the uploader when its data port is set to 0 to indicate that it is behind a firewall and need to push all data. The uploader is responsible for connecting to the downloader to transfer the file.

File Transfer

From this point onward, the hosts do not send messages to the central server any longer. The host requesting the file makes a TCP connection to the data port specified in the 0xCC message from the server. To request for the file that the client wishes to download, it sends the following HTTP messages: a string "GET" in a single packet and a message with the format:

<Nick>	<Filename>	<Offset>

Where:

- Nick is the client's nick
- Filename is the name of the file to be downloaded
- Offset is the byte offset in the file at which to begin the transfer. It is needed to resume prior transfer.

The remote host will then return the file size and, immediately following, the data stream. The direct file transfer between Napster's hosts uses a P2P architecture. Once the data transfer is initiated, the downloader should notify the server that they are downloading a file by sending the DOWNLOADING FILE message. Once the transfer is completed, the client sends a DOWNLOAD COMPLETE message.

Firewalled Downloading

Napster also has a method to allow clients behind firewalls to share their contents. As described, when the file needs to be pushed from a client behind a firewall, the downloader sends a message ALTERNATE DOWNLOAD REQUEST message to the server. This causes an ALTERNATE DOWNLOAD ACK to be sent to the uploader, which is similar to the DOWNLOAD REQUEST message for a normal download.

Once the uploader receives the message from the server, it should make a TCP connection to the downloader's data port (given in the message). Upon connection, the downloader's client will send one byte, which is the ASCII character "1." The uploader should then send the string "SEND" in a single packet, and then the message (format was shown before).

Upon receipt, the downloading client will either send the byte offset at which the transfer should start, or an error message such as "INVALID REQUEST." The byte offset should be sent as a single packet in plain ASCII digits. A 0 byte offset indicates the transfer should begin at the start of the file.

Implementation

The Napster protocol was a closed one, meaning no one knows for sure how file searching and transfer is done. So when Napster was first introduced, there was only one client implementation, which was called Napster.

Napster focuses exclusively on MP3-encoded music files. Although no other file types were supported, an intriguing subculture of client clones and tools soon arose, reverse engineered from Napster's closed-source clients.

The intent behind the development was to have greater user control. For instance, a form of MP3 spoofing implemented by tools such as Wrapster could enclose an arbitrary file with a kind of wrapper that made it look like an MP3 file to Napster servers. It would then appear in the server databases and be searchable by other clients wishing to download files other than music. An obstacle to this effort was that the artist–title description field allowed little information about the nonmusic file. This, the low ratio of nonmusic to music files, and the normal random distribution of connecting nodes conspired to make Napster's scope-limited searches highly unlikely to find special content. Tools such as Napigator were developed to allow users to connect to specific servers, bypassing metaserver arbitration. In this way, certain servers became known as Wrapster hangouts-primary sites for nonmusic content. Users looking for this kind of content were then more likely find it.

Nonmusic exchanges over Napster were never more than marginal, at least compared to alternative, content-agnostic systems such as Gnutella. Some alternative Napster servers, such as OpenNap which started as “safe havens” for Napster users when Napster began filtering content, did for a while begin to fill the gap, tying together former Napster clients, clones, and variations with a new kind of server that extended the original Napster protocol to all file types. No matter how much the Napster model was reengineered, however, the fundamental requirement of a Napster-compatible central server remained a serious constraint for a network based on this technology or any of its clones. To transcend this limitation, other protocols and architecture models were needed, for example, serverless networks in the style of Gnutella.

Gnutella

In early March 2000, Gnutella was created by Justin Frankel and Tom Pepper, who were both working under Gnullsoft, which is one of AOL’s subsidiaries. Gnutella’s development was later halted by AOL shortly after it was published, but the short duration where Gnutella was made online was enough to allow curious programmers to download and later reverse engineer Gnutella’s communication protocol. As a result, a number of Gnutella clones with improvements were introduced (for example, LimeWire, BearShear, Gnucleus, XoloX, and Shareaza).

The Gnutella Architecture

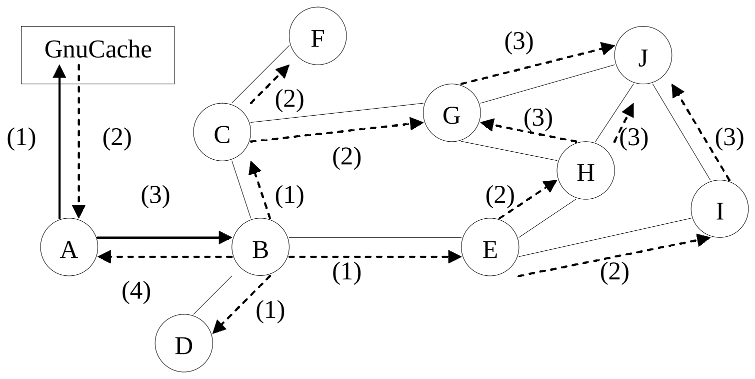
Instead of using a centralized index directory like Napster, Gnutella uses a flat network of peers called servents, to maintain the index directory of all of the content in the system.

In a Gnutella network, servents are connected to each other in a flat ad hoc topology. A servent works both as a client and a server by itself. As a server, it responds to queries from another peer servent. As a client, it issues queries to other peer servents.

For a servent to join the Gnutella network, it must find the address of a servent that is already connected to the network. This can be done by using host caches, such as GnuCache, which caches Gnutella servents (hosts) that always connect to the Gnutella network. After an address is found, it then sends a request message GNUTELLA CONNECT to the already-connected servent. The requested servent may either accept the request by sending a reply message GNUTELLA OK, or reject the request message by sending any other response back to the requesting servent. A rejection can happen due to different reasons such as an exhaustion of connection slots, having different versions of the protocol, and so forth (Limewire). Once attached to the network, the servent periodically pings its neighbors to discover other servents. Typically, each servent should connect to more than one servent since the Gnutella network is dynamic, which means any servent can go off-line or disconnect any time.

It is thus important to stay in contact with several servents at the same time to prevent from being disconnected from the network. Once a server receives the ping message, it sends back a pong message to the server that originated the ping message using the same path that the ping message came from. A pong message contains the details of the servent such as port, IP address, the number of files shared, and the number of kilobytes shared. Gnutella does not use any directory servers, as each servent maintains its local index directory. To search for a file, a node sends a query to all its neighbors that are directly connected to it. Once a query is received by a neighbor, the query criterion is checked against the local index directory and the message is in turn propagated to all its neighbors and so forth.

Figure 9. Example of Gnutella protocol



User A connects to the GnuCache to get the list of available servents already connected in the network.
GnuCache sends back the list to User A.
User A sends the request message GNUTELLA CONNECT to User B.
User B replies with the GNUTELLA OK message, granting User A to join the network.

If the check matches the local data in any servent, the servent sends back a queryHit message to the servent that initiated the query along the same path that carried the query message. However, when the servent generating the queryHit message is behind a firewall, the requesting servent cannot create a connection to it. In this case, the push message is sent by the requesting servent to the servent that generates the queryHit message and stays behind the firewall to initiate the connection instead. Note that file transfer is done using the HTTP protocol, and not the Gnutella protocol.

Since Gnutella broadcasts its messages, in order to prevent flooding the network with messages, the TTL (Time-to-Live) field is included in the header of every Gnutella message. The TTL field is decremented by one every time the message is passed through one servent. The servent decrementing the TTL and finding that its value equals to zero will drop that message. Each servent also needs to maintain a list of recently seen messages by storing the Descriptor ID and the Payload Descriptor of each incoming message to prevent forwarding the same message repeatedly. A detailed description of the Gnutella message and protocol will be explained in the next section. Figure 9 illustrates the concept of the Gnutella protocol.

As shown in Figure 9, suppose that User A that is connected to the network wants to search for some files. He/she sends a query message to his/her neighbor, User B. User B first checks that the message is not an old one. He/she then checks for a match with his/her local data. If there is a match, he/she sends the queryHit message back to User A. User B then decrements TTL by 1 and forwards the query message to Users C, D, and E. Users C, D, and E performs the same steps as user B and forwards the query message further to Users F, G, H, and I who will again repeat the same process mentioned earlier. Suppose that User H is the first one to forward the query message over to User J. Any subsequent query messages forwarded by Users G and I to user J will be discarded since checks against its local table shows that it has already received those messages. This holds for User G as well when User H forwards the query message to him/her. Suppose now that User J finds a match against his/her local data. He/she responds by sending a queryHit message back to User A, following the same path the query message was carried through which is from J, to H, to E, to B, and to A. Now User A can initiate the file download directly with User J using the HTTP protocol.

The Gnutella Protocol

This section describes in detail the messages that are used in the Gnutella protocol with reference to Kan (2001). The message used to communicate between the servents is called Gnutella descriptors. Gnutella descriptors consist

Table 2. Five descriptors used in the Gnutella protocol

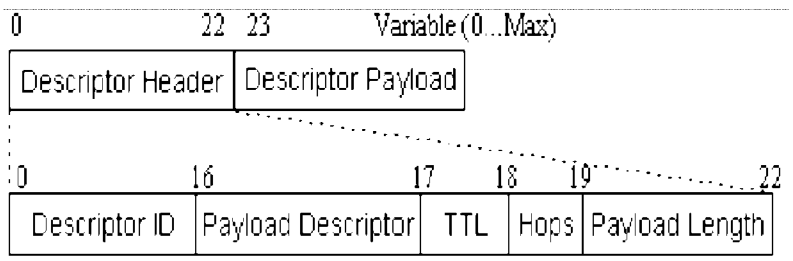
Descriptor	Description
Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.
QueryHit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.

of the Descriptor Header and the Descriptor Payload. There are five types of Gnutella descriptors: Ping, Pong, Query, QueryHit, and Push, as discussed in Table 2.

Descriptor Header

The Descriptor Header consists of five parts: Descriptor ID, Payload Descriptor, TTL, Hops, and Payload Length. (See the figure below.)

- Descriptor ID is a unique identifier for the descriptor on the network.
- Payload Descriptor identifies the type of each descriptor: 0x00 for Ping, 0x01 for Pong, 0x40 for Push, 0x80 for Query, and 0x81 for QueryHit.
- TTL represents the number of times the descriptor will be forwarded by Gnutella servents before it is discarded from the network.
- Hops represent the number of times the descriptors has been forwarded. The result of the current hops value plus the current TTL value always equals to the initial TTL value.
- Payload Length is used to locate the next descriptor. The next descriptor header is located exactly Payload Length bytes from the end of this header.



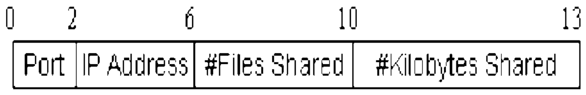
Descriptor Payload

There are five types of descriptor payload: Ping, Pong, Query, QueryHit, and Push.

1. Ping

   Ping descriptor has no associated payload and is of zero length. A servent uses Ping descriptors to actively probe the network for other servents.

2. Pong

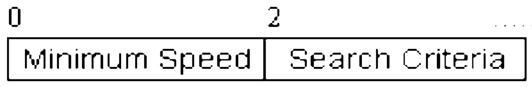


   Pong descriptor payload has four parts: port, IP address, the number of files shared, and the number of kilobytes shared. Pong descriptors are only sent in response to an incoming Ping descriptor. One Ping descriptor can be replied with many Pong descriptors.

   Port is the port number on which the responding host can accept incoming connections.

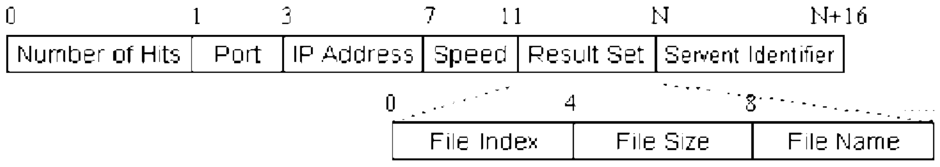
   - IP Address is the IP address of the responding host.
   - #Files Shared is the number of files that the servent with the given IP address and port is sharing on the network.
   - #Kilobytes Shared is the number of kilobytes of data that the servent with the given IP address and port is sharing on the network.

3. Query



   The Query descriptor which is Gnutella's search message format, consists of two parts: Minimum Speed and Search Criteria. Minimum Speed is the minimum speed in KBytes/sec of the servent that should respond to this message. Search Criteria contains the search criteria of the requesting servent. The maximum length of the search criteria is bounded by the Payload Length field of the descriptor header.

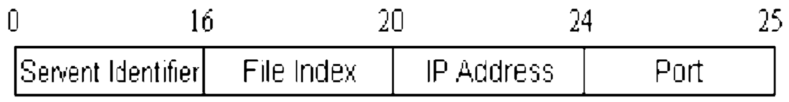
4. QueryHit



QueryHit descriptors are the responses to the Query descriptor mentioned above. These descriptors are sent by the servents when they find a match against their local data. The descriptor ID field in the Descriptor Header of the QueryHit should be the same as that of the associated Query descriptor. This will allow the requesting servent to identify the QueryHit descriptor associated with the Query descriptor it generated.

- Number of Hits is the number of query hits in the result set.
- Port is the port number on which the responding host can accept incoming connections.
- IP Address is the IP address of the responding host.
- Speed is the speed in KBytes/sec of the responding host.
- Result Set is a set of responses to the corresponding Query. This set contains the Number of Hits elements, each of which has the structure comprising file index, file size, and file name.
- Servent Identifier is a 16-byte string uniquely identifying the responding servent on the network.

5. Push (0x40)



The Push descriptor is used by the requesting servent to ask for a responding servent behind the firewall to initiate the connection.

- Servent Identifier is a 16-byte string uniquely identifying the servent that is being requested to push the file with index File_Index. This servent identifier is set to be the same as the servent identifier returned in the corresponding QueryHit descriptor sent by the servent behind the firewall.

- File Index uniquely identifies the file to be pushed from the requested servent.
- IP Address is the IP address of the requesting host.
- Port is the port number of the requesting host.

Rules

There are generally five rules in the Gnutella protocol for servents to follow in order to maintain desirable network traffics.

- Rule 1: All servents must memorize the unique 128-bit Descriptor ID every time a message is delivered or originated. If these memorized messages are received again, it will not be forwarded. This helps eliminate looping in the network thereby reducing unnecessary traffic.
- Rule 2: Pong, QueryHit, and Push descriptors may only be sent along the same path that carried the incoming Ping, Query, and QueryHit descriptors, respectively. This ensures that only those servents that routed the Ping (Query, and QueryHit) descriptor will see the Pong (QueryHit, and Push) descriptor in response. A servent that receives a Pong (QueryHit, and Push) descriptor with Descriptor ID = n, but has not seen a Ping (Query, and QueryHit) descriptor with Descriptor ID = n should discard the Pong (QueryHit, and Push) descriptor from the network.
- Rule 3: A servent forwards incoming Ping and Query descriptor to all of its directly connected servents, except for the one that sent the incoming Ping or Query.
- Rule 4: A servent decrements a descriptor header’s TTL field, and increment the Hops field, before it forwards the descriptor to any directly connected servent. If after decrementing the header’s TTL field, the TTL field is found to be zero, the descriptor is discarded.
- Rule 5: If a servent receives a descriptor with the same Payload Descriptor and Descriptor ID as the one it has received before (check by comparing with the ones the servent stores in the table), a servent discards this descriptor.

Gnutella Protocol Analysis and Improvement Methods

As with other P2P file sharing protocols, Gnutella was intentionally designed to achieve four basic goals:

- Flexibility: Since the Gnutella network is ad hoc, any servent can join or leave the network at anytime it wants and this occurs very frequently. Therefore, the Gnutella protocol must be designed to be flexible enough to keep operating efficiently despite the constantly changing set of servents.
- Performance and Scalability: Performance is measured in terms of the throughput that any servent initiating query messages will get the QueryHit messages replying back in an acceptable time. Scalability implies that the Gnutella protocol is able to handle a large number of servents without much degradation in performance.
- Reliability: Reliability focuses on security issues that external attacks should not be able to degrade significant data or result in performance loss.
- Anonymity: Anonymity concerns mainly with protecting the privacy of participants and the identity of the people seeking or providing the information.

Flexibility

With regards to the issue of dynamic environments in Gnutella, since a servent periodically pings other servents, this will prevent it from being cut from the network and will keep the network functioning as long as there is a connection between servents. However, the strength of Gnutella comes from the fact that there is a huge amount of users being online concurrently and most of them share their information.

Performance and Scalability

Performance and scalability can be considered as one of the biggest concerns in the Gnutella community. Problems relating to the issue of performance arise from many servents connecting to the network via low-speed modems. These servents are usually scattered all over the network and they get overflooded with messages until they become unresponsive, which is no more different from their being off-line. This causes the network to be highly fragmented (peers appear to be in isolated clusters), thus causing query messages to be limited and not to go beyond each clustered fragment. This drastically degrades the search results.

The second problem concerns ping and pong messages that are sent between servents throughout the network. Since every servent has to constantly ping other servents in order to obtain their addresses; the very fact that there is usually a large number of servents indicates that the number of ping and pong messages would be enormous as well. An investigation conducted by Dimitri, Antonio, and Bill (2002) shows that the number of pong messages can consume up to about

50% of all Gnutella traffic. This will prevent other more important messages such as Query, QueryHit, and Push to route through the network. The consequence is that the performance of the network will eventually be degraded because most of the network bandwidths are used to send ping and pong messages (note that ping messages are broadcasted to every directly connected peer).

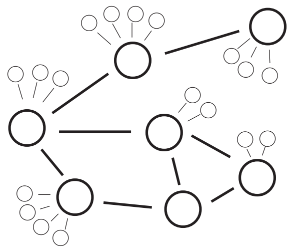
The third performance problem stems from the fact that Gnutella does not enforce its users to share files. This encourages most users to be freeloaders as most of them just join the network without sharing any of their files. Suppose that the TTL of the query messages is seven and every servent is connected to four other servents. Theoretically, the search will go through 16,384 servents, but only if 10% of those servents share their files, the search will drop to only around 1,600 servents. This reduces the probability to find the desired files dramatically, and this still does not consider the fact that some servents may go off-line or may drop the query messages due to network congestion problems.

The fourth problem involves download failures which may very well have been caused by the sharing of partial files or the limitation of upload speed by the users who share those files. Back to the issue of scalability, Ritter (2001) argues that the Gnutella network is not as scalable as it claims to be. This is due to the propagating nature of the Gnutella protocol which consumes a lot of network bandwidth, hence causing the number of Gnutella users to be limited by the underlying network bandwidth (network congestion because of the large number of messages transmitted over the network). This contradicts the original design intention of Gnutella which was to be able to support an unlimited number of users. The TTL field in the Descriptor Header, the cache of previously seen message kept by each servent, and the rules imposed by the Gnutella protocol are designed to help reduce this problem, but more powerful techniques are needed to ensure true scalability.

There are several approaches (LimeWire, 2003; Ivkovic, 2001) that are proposed to help achieve higher performance and scalability, encourage content sharing, dishearten freeloaders, reduce unnecessary network traffic, create and maintain a healthy network structure, and so forth. Encouragement of content sharing may be done by making it a default feature to automatically share completed downloads. This means that when a download is completed, the newly downloaded file should be shared automatically by the user. Download failures can be resolved by having an incomplete download directory to prevent the sharing of partial files.

There are two types of freeloaders: users who only download files for themselves without ever providing files for download to others, and users who provide low-quality contents to others for download. The problem of freeloaders can be solved by blocking Web-based downloads, blocking queries from Web-based indexing tools or through reward-and-punish methods (this may require slight changes in the protocol).

Figure 10. Centralized and decentralized network topology (Minar, 2001)



To reduce unnecessary network traffic would mean to reduce the number of ping and pong messages. One approach to accomplish this is to avoid periodically pinging the network by setting a rule whereby a node can only start sending ping messages when it detects that the peers directly connected to it are two or fewer. The task of creating and maintaining the healthy network can be achieved by dropping a connection that has a high potential to get clogged (this may remove the slower-modem users from the center or fast areas of the networks) and using the prioritization scheme to discard unnecessary messages if necessary. The rule of dropping a connection may be in the following form: Drop a connection to a node if it has not been able to receive or send the message for about 10 seconds, and does not respond to a ping with TTL = 1. A prioritization scheme can be used to selectively drop the most useless message first if necessary. The priorities of messages are ordered from high to low: Push > QueryHit > Query > Pong > Ping.

Another approach to improve performance is through the use of Super-peers (this is used in newer P2P architectures such as FastTrack and OpenFT) and caching of the results. One of the main causes of performance degradation comes from the assumption that every servent has the same resource capacity. The truth, in fact, is that many servents have weak connections (low-speed modem), and participate in the network only for a short time. This makes these slow servents unsuitable for taking an active role in the protocol. Super-peers can be used as proxies for less powerful servents. Super-peers, which typically have high bandwidth, act as local search hubs on behalf of the less powerful peers connecting to them and they stay online permanently. Super-peers can also help reduce network traffic by caching query routing information. Actually, Gnutella v0.6 protocol already proposed the use of Super-peers. The use of Super-peers will make the Gnutella network topology a mix of centralized and decentralized network as illustrated in Figure 10.

In order to reduce the time needed to find a match for a query, each servent can cache a QueryHit message that is sent back through it, which is associated with the query sent to it before. In the case of when there is the same query associated with that queryHit sent to it again, that servent can send back the QueryHit corresponding to that Query immediately. This method will perform best when repeated queries are common. We can create this effect by arranging the servents interested in the same contents stay close together in the network. Note that pong messages can also be cached in the same way so that the next time the ping messages come, a servent can return the cached pong messages immediately, eliminating the need to forward the ping message further. The idea is that each servent remembers all pong messages that it has received. When its neighbor sends a ping message with TTL of n to it, the servent will reply with a pong message and all its cached pongs with hop counts less than n. This will result with each ping message propagated to only the directly connected neighbors, and the servents will not need to repeatedly forward the same pong messages.

Reliability

The issue of reliability has been a heated discussion topic recently. Reliability focuses on the security issues because malicious users can render the network to malfunction causing reliability to be compromised. Since the Gnutella network is fully distributed, the contents are spread throughout many nodes which make attacking a specific machine useless, as that cannot bring the Gnutella network down. However, with the introduction of Super-peers, the reliability of the Gnutella network may be questionable as attackers can target the Super-peers. Even though this will not make the entire Gnutella network go down, some parts of the network can suffer significantly. Apart from that, there are three kinds of attacks that can be done easily in a loosely structured, highly dynamic P2P network such as Gnutella: flooding, malicious or fake contents, and hijacking queries. This will be discussed based on Dimitri, Antonio, and Bill (2002). In fact, Gnutella network opens many security risks since the design focus is primarily on functionality and efficiency.

Flooding the network with a large number of query messages or supplying erroneous replies to all queries is a form of denial of service attack. Note that the flooding of pong or QueryHit messages is useless since they will be discarded if there are no matching ping or query messages previously sent over the same network. Flooding of ping messages is also unfruitful because of the caching of pong messages as discussed before. Dimitri, Antonio, and Bill (2002) claim that flooding problem cannot be prevented but can be minimized by load balancing.

Providing malicious or fake contents, such as files that contain viruses, files with altered or corrupted content, or advertising and misleading others to believe a file to be something else can be a real problem. What is worse is that the virus files, if they exist in the network, tend to propagate quickly. Therefore, it is essential to have a way to authenticate the content of the file before receiving it. Dimitri, Antonio, and Bill (2002) suggest the Fourier Transforms technique to determine the content in a file that are not identical bit-wise to see whether it is what is expected.

Hijacking queries can be done easily in the Gnutella network. This stems from the trust of third parties in the network. The use of Super-peers also makes it easy to hijack the queries since each Super-peer can see a large portion of queries. Point-to-point encryption, Distributed Hash Tables, and Selection mechanism are all techniques proposed by Dimitri, Antonio, and Bill (2002) to help resist flooding and query interception attacks.

According to Ripeanu (2001), the current Gnutella network follows a multimodal distribution, which is a combination of a power-law and quasi-constant distribution. This topology keeps the network reliable almost as a power-law distribution topology, which allows the network to tolerate random node failures.

Unfortunately, Gnutella network topology can be acquired easily through the use of a crawler and analyzing the data gathered. This permits highly efficient denial of service attacks by educated attackers.

Anonymity

The last issue, anonymity, can be achieved naturally in the Gnutella network. The requested servents do not know who the requesters are since the requesters can be anyone along the query paths. In the same way, the requesting servents do not know who the requested servents are either. This can be a problem as anonymity makes it possible for many security threats to easily occur.

On the other hand, although one may think that the Gnutella protocol provides total anonymity in the network, unfortunately a hacker can still invade one's privacy by scanning the Descriptor ID in the Descriptor header to trace back on Gnutella messages. So no servent is totally anonymous—at least not yet!

FastTrack

One of the newer and more innovative peer-to-peer architectures is the FastTrack network. It came as a solution to the problems that both Napster and Gnutella were facing. The FastTrack network is by nature a hybrid architecture, which as mentioned in the earlier sections, is a cross between two or more basic network topologies. For FastTrack, it is namely the cross between centralized and decentralized topologies (Yang & Garcia-Moline, 2002).

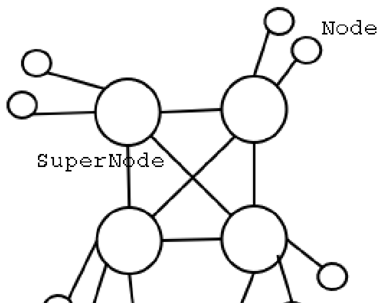
The FastTrack Protocol

The FastTrack protocol is a proprietary architecture, where rights to use the network has to be obtained through a company called Sherman Networks (Bergner, 2003). Therefore, very little is known of the actual protocol used. Many attempts have been made to reverse engineer the FastTrack protocol. The most well known to date would be the giFT project (Source Forge, 2002), as they were the closest to finally cracking the protocol. FastTrack reacted by changing its encryption to the point where it was virtually impossible to reverse engineer (Bergner, 2003). The work done by the giFT project, however, was sufficient to give a rough outline of how the FastTrack protocol actually works, even if the information may now very well be outdated. The following section contains a description of the architecture used by all FastTrack clients.

This technology uses two tiers of control in its network as shown in Figure 11. The first tier is made up of clusters of ordinary nodes that log onto Super Nodes (ordinary machines with high-speed connection). As discussed previously, this sort of connection mimics the centralized topology. The second tier consists of only Super Nodes that are connected to one another in a decentralized fashion.

The number of peers that can be designated as Super Nodes can vary from ten to several thousand. This is because these Super Nodes themselves are just ordinary nodes that can and will join or leave the network as they please. Therefore, the network is dynamic and always changing. In order to ensure the constant availability of the network, there exists a need for a dedicated peer (or several of these peers) that will monitor and keep track of the network. Such a peer is called a bootstrapping node (Kurose & Ross, 2003) and it should always be available online. When a FastTrack client, for example Kazaa, is executed on a peer, it will first contact the bootstrapping node. The bootstrapping node will then determine if that particular peer qualifies to be a Super Node. If it does, then it will be provided with some (if not all) IP addresses of other Super Nodes. If

Figure 11. An illustration of the FastTrack topology



it qualifies to be only an ordinary peer, then the bootstrapping node will respond by providing the IP address of one of the Super Nodes (Balakrishnan, Kaashoek, Karger, Morris, & Stoica, 2003).

Certain FastTrack clients such as Kazaa use a method known as the "Reputation System," where the reputation of a certain user is reflected by their participation level (a number between 0 and 1,000) in the network. The longer the user stays connected to the network, the higher its participation level will be, which in turn means that it will be more favoured in queuing policies and hence should receive better service. This is mainly to encourage users to share files and thus effectively reduce the number of "passenger clients" on the network.

Resource discovery is accomplished through the act of broadcasting between Super Nodes. When a node from the second tier makes a query, it is first directed to its own Super Node, which will in turn broadcast that same query to all other Super Nodes to which it is currently connected. This is repeated until the TTL of that query reaches zero (Sharma, 2002). So, if for example, the TTL of a query is set to 7 and the average amount of nodes per Super Node is 10, a FastTrack client is able to search 11 times more nodes on a FastTrack network as compared to Gnutella (Aitken, Bligh, Callanan, Cocoran, & Tobin, 2001). This provides FastTrack clients with a much greater coverage and better search results. However, there is one drawback to such a broadcasting method, and that is the daunting amount of data that needs to be transferred from Super Node to Super Node. This is the very same problem that has been plaguing the Gnutella network. This, however, is not a serious problem for Kazaa as opposed to Gnutella, therefore the Super Nodes are nodes that are guaranteed to have fast connections.

Each of the Super Nodes that received the query performs a search through its indexed database that contains information of all the files shared by its connected nodes. Once a match is found, a reply will be sent back following the same path the search query was propagated through until it reaches the original node that issued the query. This method of routing replies is similar to Gnutella, and hence runs the risk of facing the same problem of losing replies as it is routed back through the network. This is due to the fact that the Gnutella network backbone, as mentioned previously, is made up of peers that connect and disconnect from the network very sporadically. This would mean that reply packets that are being routed back may be lost as the path it took is no longer there because one or more of the nodes making up the link disconnected itself. The aforementioned problem, however, is not as serious for Kazaa users as the FastTrack network backbone is made up of peers that have high-speed connections (Super Nodes) and hence the return paths can be considered more reliable.

OpenFT

As mentioned in the previous section, the FastTrack network protocol is proprietary, hence not much about its protocol is known. This very fact has become the source of motivation for various individuals and groups to try to break the FastTrack protocol. The most well known of all is the giFT project. According to Bergner (2003), the initial meaning of the abbreviation giFT was Generic Interface to FastTrack or giFT isn't FastTrack. The giFT project came very close to actually breaking the FastTrack protocol. However, FastTrack reacted by changing its encryption, making it impossible to reverse engineer.

Since the FastTrack protocol is no longer feasible to crack, the aim of the giFT project was changed to develop a system that can connect many different heterogeneous networks and still be able to share files between them. Therefore, the meaning of its abbreviation was also changed to giFT Internet File Transfer (Bergner, 2003). The project led to the development of a new and improved network protocol that was very much like FastTrack; it was called OpenFT.

The OpenFT Architecture

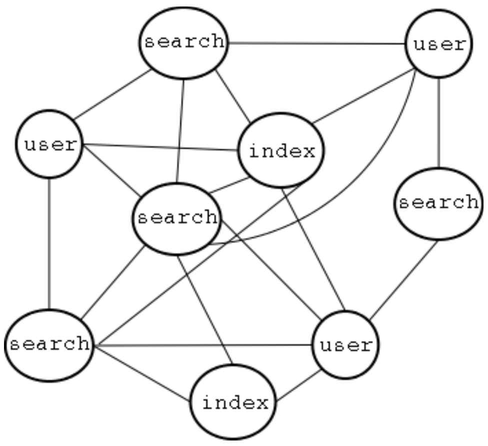
Like FastTrack, the OpenFT protocol also classifies the nodes in its network into different roles, but instead of a two-tier control architecture, OpenFT added an extra tier, making it a three-tier control architecture as shown in Figure 12. The classification of nodes is based on the speed of its network connection, its processing power, its memory consumption, and its availability (SourceForge, 2002).

The first tier is made up of clusters of ordinary machines which we refer to as User Nodes. These nodes maintain connections to a large set of Search Nodes (ordinary machines with high speed connection). The user nodes then update a subset of the search nodes to which it is connected with information regarding files that are being shared (SourceForge, 2002).

The second tier is made up of machines that are referred to as Search Nodes. These nodes are the actual servers in the OpenFT network. These servers have the responsibility to maintain indices of files that are shared by all the User Nodes under them. On default, a Search Node can manage information about files stored at 500 User Nodes (SourceForge, 2002).

The third tier is made up of a group that it much smaller, because the requirements to qualify for this group are much more stringent. One has to be a very reliable host that has to be up and available most of the time. These nodes are referred to as Index Nodes as their main purpose is to maintain indices of existing Search Nodes. They also perform tasks such as collecting statistics and

Figure 12. An illustration of the OpenFT topology



monitoring network structure. Basically, this group can be seen as the administrator group that ensures that all participating nodes are working fine and are up to expectation (SourceForge, 2002). It should be noted that the second and third tier of control can actually be performed by the same node. In other words, a node can function both as a Search Node and as an Index Node simultaneously.

By default, each User Node has to select three Search Nodes to maintain its shared file information. If accepted, the selected Search Node becoms a Parent to that particular User Node, which will now have to send the list of its shared files to it (Bergner, 2003). In addition, all nodes also have to maintain a list of available Search Nodes. When a query is sent, it will be sent to all nodes that are found in this list (SourceForge, 2002).

The OpenFT Protocol

The OpenFT network protocol uses a simple packet structure to make it easy to parse. There are altogether 28 packet types and all of them have the following packet header.

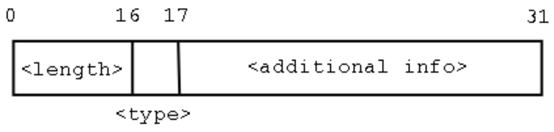


Table 3. Summary of packets used in the OpenFT network protocol

Packet	**Description**
version	Used to indicate protocol version and check to see if it is outdated
nodeinfo	Used to communicate node information such as IP address, port numbers, and OpenFT node category
nodelist	Used to list down other nodes
nodecap	Used to indicate node capabilities
ping	Used to keep connection alive
session	Used to establish session
child	Used to request to be the child of a particular Search Node
addshare	Used to add information about the files that are shared in a User Node to the registry of a Search Node
remshare	Used to remove information about the files that are shared in a User Node to the registry of a Search Node
modshare	Used to modify information about the files that are shared in a User Node to the registry of a Search Node
stats	Used to request network statistics from Index Nodes
search	Used to query for files
browse	Used to browse the files shared by a particular node
push	Used to perform HTTP PUSH (such as the one mention in Gnutella) through firewalls

Length describes the length of the packet excluding the header. The current implementation uses bits 16–31 to store both flags and packet type information (Bergner, 2003).

As mentioned in the previous section, each node is assigned a particular role in the network. At the moment of writing, there is yet a distributed algorithm that can be implemented to perform the role assignment task. For now, the users themselves choose the role they want to play in the network. Table 3 provides a summary of the packets used by the OpenFT network protocol (Bergner, 2003; Guilfoyle & Lempsink, 2003).

All the packets mentioned can basically be categorized into several categories so that their usage and function can be seen more clearly. With reference made to Bergner (2003), they are:

- Session Establishment: version, nodeinfo, nodelist, nodecap, session
- Connection Maintenance: ping
- File Sharing Management: child, addshare, remshare, modshare
- Searching for files: search, browse
- Misc: stats, push

Comparisons

P2P systems use different architectural models and techniques for resource discovery and handling performance and security issues. A comparison between various P2P systems is given in Table 4.

Table 4. A comparison of various P2P systems

Characteristics	System	Description
Purpose	Napster	MP3 file sharing
	Gnutella	File sharing of all types
	FastTrack	File sharing of all types
	OpenFT	File sharing of all types
Architecture	Napster	Peers connected to a centralized server
	Gnutella	Flat/Ad hoc network of peer servents (pure P2P)
	FastTrack	Decentralized two-level hierarchical network of group-leader peers and ordinary peers
	OpenFT	Decentralized three-level hierarchical network of search peers, index peers, and ordinary peers
Lookup Method	Napster	Using central directory
	Gnutella	Query flooding: Broadcast queries to peers and make a direct connection when downloaded
	FastTrack	Using group-leader peers
	OpenFT	Using search peers
Decentralization	Napster	The system is highly centralized. Peers are connected directly to the central index server.
	Gnutella	The system is highly centralized. The topology is flat and each peer is truly equal.
	FastTrack	The system is decentralized in the sense that there is no explicit central server. However, in each group, the ordinary peers are still connected to their group-leader peer in a centralized manner.
	OpenFT	The network structure is very much like FastTrack, only OpenFT is slightly more optimized for performance.
Scalability	Napster	The system is not scalable because having everyone connected to the directory server is bottleneck prone.
	Gnutella	Theoretically, the system can easily expand indefinitely, but in practice, it may be limited by the underlying bandwidth. Traffic jams of a large number of messages transmitted over the network can cause the network to stop functioning properly.
	FastTrack	The scalability of the system is medium. Although the architecture is fairly decentralized, it is not entirely serverless. A bottleneck may occur locally within a group.
	OpenFT	The scalability of the system is just like FastTrack. It is easily scalable, but bottlenecks may occur within a group since it is not entirely serverless.

Table 4. A comparison of various P2P systems, cont.

Anonymity	Napster	Hardly any anonymity since all users have to sign up onto the central server before they can connect to the network. Users are almost always anonymous to each other, though, since the user directory is never queried directly.
	Gnutella	Anonymity is preserved naturally by the protocol itself since the messages may come from any peer in the local paths. However, the Descriptor ID in the Descriptor header may be used to trace the messages. Nosy nodes can record queries and responses.
	FastTrack	Level of anonymity is slightly better than Napster but worse than Gnutella. Users are not fully anonymous in the FastTrack network.
	OpenFT	Level of anonymity is similar to FastTrack.
Security	Napster	Moderate since Napster is managed by a centralized server. This allows Napster to exert much more control which makes it much harder for clients to fake IP addresses, port numbers, and so forth. And since Napster only allows the sharing of MP3 files, this makes security threats that are related to fake content, viruses, and so forth not too common and more traceable as compared to the other three protocols.
	Gnutella	Low since the protocol design primarily focuses on functionality and efficiency. Prone to several security threats: flooding, malicious or fake content, virus spread, hijacking of queries, denial of service attacks. Security threats mainly focus on services rather than host. In the case of one user staying behind the firewall, download can be achieve by requesters sending a push message to ask the firewalled user to initiate a connection. In the case of two users both staying behind the firewall, a connection cannot be created.
	FastTrack	Low, but is still better than Gnutella, since it maintains a hybrid architecture. Security threats such as flooding, malicious or fake content, viruses, and so forth can be reduced as all the Super Nodes actually function as centralized servers to nodes that are under their domain. Another threat is the integration of spyware into the popular FastTrack client, Kazaa. These spywares monitor the activities of users in the background. Any information from surfing habits to favourite Web sites can be gathered and sent back to the server, which will then use this information for targeted marketing, and so forth.
	OpenFT	Very similar to threats faced by FastTrack. One of the main differences is that it does not have spywares integrated into its clients.
Self-Organization	Napster	A highly self-organized system is not necessary. Organization of nodes/resources are handled by the central servers.
	Gnutella	The system is highly self-organized. It adapts gracefully to the dynamic nature of the Internet through the use of ping messages to periodically find available peers connected to the network. But this comes with a drawback, which is the staggering amount of data transfer involved.

Table 4. A comparison of various P2P systems, cont.

	FastTrack	The system is very much self-organized as well. It adapts to the dynamic nature of the Internet, just like Gnutella. The slight difference in its protocol allows FastTrack to achieve a highly self-organized system with much less data transfer than Gnutella.
	OpenFT	Self-organization in OpenFT is similar to FastTrack.
Lookup Completeness	Napster	The lookup method is complete because search is done by the central server which has the complete index directory.
	Gnutella	The lookup method is incomplete because a query may not reach all servents. It has a much greater coverage than Napster, but it takes much longer to perform a lookup.
	FastTrack	The lookup method in FastTrack is incomplete as well, but it is able to search many more nodes than to Gnutella for the same time span.
	OpenFT	The lookup method on OpenFT is similar to FastTrack. There may be some minor adjustments to improve performance, but fundamentally, they are both the same.
Fault Resilience	Napster	The malfunction of the central server can cause a system-wide malfunction.
	Gnutella	The malfunction of some nodes would not cause the system to stop functioning as long as there are enough nodes connected to the network at a given time. However, this will definitely degrade performance.
	FastTrack	The malfunction of an ordinary peer would not hurt the system. The malfunction of a group-leader peer is taken care of by reassigning all ordinary peers connected to other group-leader peers.
	OpenFT	Fault resilience for OpenFT is similar to FastTrack, where the malfunction of a user node would not hurt the system. If any of the index nodes or search nodes fails, all that needs to be done is to just reassign all affected user nodes to other index or search nodes.

Summary

It should be relatively clear by now that P2P technology is still in the infant stage of its development. There is still great potential for growth and improvements. From this chapter alone, we can see how P2P has evolved from a more centralized architecture such as Napster into a fully distributed architecture such as Gnutella, only to evolve again into hybrid architectures such as FastTrack and OpenFT. This evolution in technology is spurred mainly by the need to achieve better efficiency and speed in content sharing as well as the need to survive law suits against these architectures. Much more innovative architectures will surface as the race toward network efficiency and survival continues.

References

Aitken, D., Bligh, J., Callanan, O., Corcoran, D., & Tobin, J. (2001). *Peer-to-peer technologies and protocols.*

Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., & Stoica, I. (2003, February). Looking up data in P2P systems. *Communications of the ACM, 46*(2).

Barkai, D. (2001). An introduction to peer-to-peer computing. From http://www.intel.com/update/departments/initech/it02012.pdf

Bergner, M. (2003). *Improving performance of modern peer-to-peer services*. UMEA University, Department of Computer Science.

Dimitri, D., Antonio, G., & Bill, K. (2002). Analysis of peer-to-peer network security using Gnutella. From http://www.cs.berkeley.edu/~daw/teaching/cs261-f02/reports/defig.pdf

Guilfoyle, J., & Lempsink, E. (2003). giFT's interface protocol. From http://gift.sourceforge.net/docs.php?document=interface.html

Hebrank, M. Gnutella & Napster. HUH? or What do I need to know to keep from looking like an idiot. Fom http://www.wacked.org/~heller/gnutella/

Ivkovic, I. (2001). Improving Gnutella protocol: Protocol analysis and research proposals. From http://www.swen.uwaterloo.ca/~iivkovic/Gnutella.Limewire2001.pdf

Kan, G. (2001). Gnutella. In A. Oram (Ed.), *Peer-to-peer: Harnessing the power of disruptive technologies.* Sebastopol, CA: O'Reilly.

Kurose, J.F., & Ross, K.W. (2003). *Computer networking: A top-down approach featuring the Internet*. Boston: Addison-Wesley.

LimeWire (n.d.). The Gnutella Protocol Specification v0.4, Document Revision 1.2. From http://www.clip2.com

LimeWire (2003). Improving the Gnutella network. From http://www.limewire.com/index.jsp/im_require

Minar, N. (2001). Distributed systems topologies: Part 1. From http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html

Minar, N. (2002). Distributed systems topologies: Part 2. From http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html

Peter, B., Tim, W., Bart, D., & Piet, D. (2002). *A comparison of peer-to-peer architectures*. Ghent, Belgium: Broadband Communication Networks Group (IBCN), Department of Information Technology (INTEC), Ghent University.

Ripeanu, M. (2001, July). Peer-to-peer architecture case study: Gnutella network, technical report TR-2001-26, University of Chicago. From http://people.cs.uchicago.edu/~matei/PAPERS/gnutella-rc.pdf

Ritter, J. (2001, February). Why Gnutella can't scale. No, really. From http://www.darkridge.com/~jpr5/doc/gnutella.html

Sharma, A. (2002, September). The FastTrack network. *PC Quest Magazine*, India. From http://www.pcquest.com/content/p2p/102091205.asp

Shirky, C. (2001). Listening to Napster. In A. Oram (Ed.), *Peer-to-peer: Harnessing the power of disruptive technologies*. Sebastopol, CA: O'Reilly.

Source Forge (2002, September 14). What is the giFT project? From http://cvs.sourceforge.net/viewcvs.py/gift/giFT/README?rev=1.9

Turcan, E. (2002). Peer-to-peer: The third generation Internet. From http://csdl.computer.org/comp/proceedings/p2p/2001/1503/00/15030095.pdf

Tyson, J. (2000). Marshall Brain's HowStuffWorks, How Napster worked. From http://www.howstuffworks.com/napster1.htm

Yang, B.H., & Garcia-Moline (2002, February). *Designing a super-peer network*. Stanford, CA: Stanford University Press.

Chapter III

Using Peer-to-Peer Systems for Data Management

Dinesh C. Verma
IBM T.J. Watson Research Center, USA

Abstract

This chapter describes a peer-to-peer approach for managing data backup and recovery in an enterprise environment. Data management systems in enterprises constitute a significant portion of the total cost of management of data in enterprise systems. Maintaining data with a high degree of availability and reliability is typically done by having a centralized backup system that maintains backup copies of data. The maintenance of a large dedicated backup server for data management requires a highly scalable network and storage infrastructure, leading to a major expense center within the enterprise. With the current trends in workstation disk storage, an alternative peer-to-peer paradigm for data management can offer an approach that provides equivalent performance at a fraction of the cost of the centralized backup system. The author hopes that the contents of the chapter would lead to the development of more solutions that harness the power of peer-to-peer networks.

Introduction

Peer-to-peer (P2P) technology has become mainstream media story due to its widespread use in exchanging popular files, and has almost become synonymous with music swapping over the Internet. The popularity of peer-to-peer systems on the Internet underscores the amount of scalability and ease of administration that is offered by that technology. These features enable peer to peer technology to be exploited for many other applications, both over the Internet as well as within enterprise networks. In this chapter, we look at one such application of peer-to-peer systems which can result in substantial cost savings for many enterprises.

One of the key support functions expected of an IT department in most medium- and large-scale companies is that of backup and recovery of files stored on the various computers within the enterprise. Data stored in a computer may be lost due to various causes, such as user mistakes, power outages, disk crashes, accidental erasures, viruses, and so forth. Data management systems within an enterprise facilitate the recovery of lost data. The standard modus operandi of data management systems is to make automated periodic copies of data, restoring files from the most recent backup copy when required.

Most enterprises operate a backup server to maintain backup copies of data across the different machines in an enterprise. Such backup servers must have the capacity to store all of the data present in a large number of client computers while providing a very high level of availability. Due to the volume of data in an enterprise, backup servers tend to be high-end, expensive machines with high-capacity data storage devices. Many vendors provide software products that provide data backup and restore functions.

The operation and maintenance of backup servers is an expensive operation for most enterprises. For every dollar spent on new storage hardware, enterprises are estimated to spend three dollars to manage data (Clark, 2002). In many enterprises, the growing cost of IT operations is a significant concern. Furthermore, industrial surveys indicate that up to 60% of storage capacity in an enterprise typically remains unused.

The excess storage capacity available in the devices can be exploited to provide data backup services to other devices in the enterprise. This can enable machines in an enterprise to provide collaborative backup service to each other, eliminating the need for an expensive backup server. The caveat, of course, would be to ensure that the collaborative backup remains as reliable as the traditional backup server.

This chapter describes a peer-to-peer system for data backup and recovery services, and analyzes its reliability and availability. Our calculations show that such a system can obtain performance levels comparable to that of traditional backup systems.

The rest of the chapter is structured as follows: In section 2, we discuss the structure of a traditional backup system. This is followed in section 3 by the design of a peer-to-peer system for backup and restoration. In section 4, we analyze the availability and reliability of the different approaches to building a data management system. Finally, we present our conclusions and areas for future investigation.

The Traditional Data Management System

A traditional backup system typically consists of one backup site supporting a large number of clients (desktop machines, PCs, and laptops). The backup site usually consists of a cluster of backup servers. It is commonly implemented as one or more high-end servers running with a very large amount of storage space. Logically, however, the cluster appears as a single server to the backup clients. Large enterprises may have several backup sites, with a possible peer-to-peer mirroring capability among them (Castes, Kozakos, & Laing, 2004).

Backup clients in the system run software that allows users to back up and restore files from the server. Typically, client software has a mechanism to run backups at scheduled times, for example, each client may automatically be backed up once a day. Metadata associated with a backup, for example, the last time the files were backed up, is kept on the backup server. Two modes of backup are standard: a full backup in which all the files on a machine are copied over to the backup server, and an incremental mode in which only files modified since the last backup are copied over to the backup server.

Due to the need to make the backup server available around the clock, the server needs maintenance and support personnel to ensure continued operation. This makes the backup storage many times more expensive than the cost of the client storage. This leads to a paradoxical situation in that the total cost of ownership of a megabyte of storage increases as the cost of a megabyte of storage on client systems falls. This is because the enterprise needs to provision for increased storage at the backup site as the capacity of disks on individual workstations increases. On the one hand, users and application developers feel no need to worry about storage efficiency due to the large disk space available on their computers. On the other hand, they come under increased pressure from the enterprise IT staff to reduce the amount of storage used in order to reduce the costs associated with data management. To make matters worse, backup sites tend to be upgraded at a much slower pace than the turnaround time of PCs and laptops. We will soon be encountering an odd situation where storage space at backup sites will be scarce and expensive, while there is excess disk space on each client.

The Peer-to-Peer Data Management System

Peer-to-peer systems have been used successfully for a variety of applications within the Internet, such as sharing and exchanging files (Gnutella; Kazaa), and building large distributed storage systems (Rhea et al., 2003). Two approaches toward building peer-to-peer systems have been proposed, the first being the building of applications on top of an application-level multicasting and query-searching mechanism, and the second being the implementation of systems for distributed hash-tables. In the first approach, files are searched by means of a query broadcast to the peers. The second approach maps files to a unique key, and provides efficient method to look up that key in a distributed environment is implemented. The multicasting approach provides for a more general type of query, and has a proven record of scalability and robustness in the wide-area network.

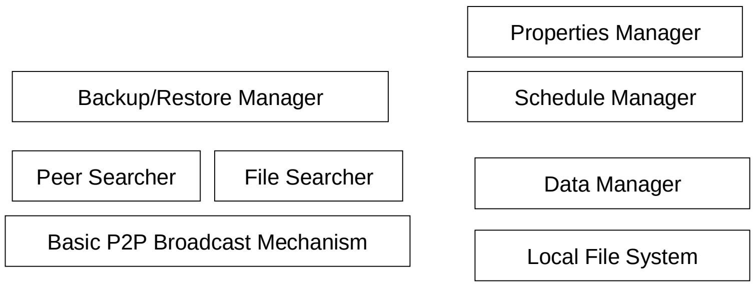
In a peer-to-peer backup system, each file is copied over to another peer in the system, rather than to a central backup site. In order to obtain a higher availability for each file, a file would need to be copied to more than one peer. The peer-to-peer data backup architecture creates an independent area on each peer to be used for data backup from other peers. Users can choose configuration properties such as the maximum fraction of the disk space to be used for backup functions and the location of the backup directory on the disk.

System Architecture

We propose a system for data management in which a software package is installed on each of the computers within an enterprise. The structure of the backup software consists of the following components shown in Figure 1.

- *Basic P2P Broadcast Mechanisms* are the basic components that perform an application-level multicast on a peer-to-peer network. The basic P2P broadcast mechanism provides the ability to search for a file with a given name and a set of attributes on a peer-to-peer network. Libraries providing this capability are generally available as components in most peer-to-peer software distributions.

Figure 1. Structure of P2P data management software



- *The Peer Searcher* is used to search for a peer that would be a suitable candidate for maintaining backups of a specific file on the local machine. When a file needs to be backed up, the peer searcher component sends a query to all the computers, looking for computers that would be able to store a backup copy of the file. The peer searcher components on other machines respond to such queries. The peer searcher would then select a subset of the responding peers as suitable repository for backup copies.
- *The File Searcher* is used to search for peers that are holding a backup copy of a file and are currently operational.
- *The Backup/Restore Manager* is the component responsible for performing the backup and restore operations, that is, the component that actually copies the data across machines.
- *The Properties Manager* is a component that keeps on tracking the properties of a current machine to assess its suitability to act as a backup copy for a peer. Properties tracked by the properties manager includes when the peer is typically up and connected to the enterprise network, the disk utilization on the system, and the speed of the processor on the system.
- *The Data Manager* is the component that maintains copies of backup files on the local system. In addition to the traditional role of managing files, the component has to maintain the security and privacy so that the backed up data is only visible to the owner of the data.
- *The Schedule Manager* is responsible for initiating the backup of the files on the local machine on a preset schedule. The function is identical to that in corresponding traditional backup clients running on an automated schedule.

Each of these components is further described in the following section.

The Backup/Restore Manager

The backup/restore manager creates the backup copy of files to one or more peers, and allows the restoration of the local file from one of those copies. Our system copies each file independently of the other files on the system.

When backing up a file, the backup/restore manager first identifies a set of suitable peers by invoking the peer search module. The backup/restore manager then contacts one or more of the peers in the set to copy the file over. If the other peer already has a previous copy of the file, an incremental backup of the file is made to the peer. Otherwise, the full file is copied to the backup server.

For restoring files, the date of desired restoration and file name are sent to the file searcher module. The file searcher module looks for the most recent copy of the file that was backed up prior to the desired restoration date, which is then copied back to the local system. The backup/restore manager is also responsible for performing the features of traditional backup features, such as identifying the set of files that do not need to be backed up (e.g., application files or temporary files), or the set of files that need to be backed up with encryption support, and so forth.

The Peer Searcher

The peer searcher is responsible for locating one or more peers that can act as potentially good sites to create a backup copy of a file. In our solution, the peer searcher is implemented using a peer-to-peer application-level multicast paradigm.

When searching for suitable peers, the peer searcher module floats a broadcast query. The broadcast query contains the name of the file being backed up, the size of the file, the name of the local machine, and the uptime cycle to which the current machine belongs. The peer searcher modules on the other peer receives the query and compose a response consisting of the following properties: the uptime cycle of the peer, the amount of free space on the peer, and a Boolean flag indicating if a copy of the file already belongs in the peer being contacted. The peer searcher module on the originating node collects all the responses, and assigns a weight to each of the responding peers. The weight is computed to give preference to peers with an existing copy of the file, larger free space, and the same uptime cycle. The peers with highest weights are selected as potential backup sites.

The Properties Manager

The search for suitable peers by the peer searcher is performed on the basis of the properties of the other peers. Some of the properties maintained for this selection are the amount of free space at the peer, network connectivity of the machine, and the uptime cycle of the machine. These properties are managed by a properties manager module running on each peer.

The properties manager keeps track of the times during the day when the local machine tends to be up. All machines maintain an uptime cycle property. The uptime cycle is an array with 24 entries, each being the probability of the machine being up during that hour of the day. The properties manager computes the probabilities over the relatively long duration, for example, a month. If sufficient data is not available, the uptime cycle is initialized to a reasonable default, for example, a probability of 1 during 9–5 local time, and a probability of 0 during other times.

The properties manager keeps track of the disk space allocated on the local peer for the task of backing up files from other peers. As this disk space fills up, the peer is less likely to respond to requests from other peers to create backup copies of files. The properties manager also maintains an indicator of the type of network connectivity that the peer is likely to have to other servers. This is determined by looking at the characteristics of the network interface that the server has active at each hour. Hours when the peer has slow network connectivity (e.g., the only interface active is a dial-up modem), the uptime cycle is marked to indicate the machine has a low probability of being available.

The Data Manager

The data manager is responsible for keeping copies of the backup files on the local disk. The data manager maintains a timestamp for each time being backed up, such timestamp being the reading of the clock on the peer with the original file. The data manager also maintains an index of all the files that are backed up on the local peer.

Files that are backed up using incremental differences from the older versions are stored locally with the incremental changes. The data manager manages the differences and delivers the complete file to any requesting user.

Files are kept in a compressed format to reduce the storage requirements. Files being backed up are also compared to see if they are identical to other files present on the system. A hash computed over the contents of the files is used as a quick check for equality. If the file is identical, then only the metadata (timestamp, owning node, etc.) information is created for the new file, with only a pointer made to the copy of the existing file.

Most enterprises maintain a limited document retention policy in order to contain the amount of storage needed at the backup site. Thus, files would typically not be backed up beyond the period of a few years. Some types of data, for example, billing records, accounts records, tax-related information, are maintained for a larger period of time, for example, seven years, depending on government regulations and operating practices. The data manager maintains the time period for which a file needs to be maintained in the metadata and deletes files that are past their document retention period.

Other Components

The schedule manager performs automatic backup of the contents of the local file system at regular intervals. The function is similar to that of traditional data management systems and implemented using existing system scheduling mechanisms.

The file searcher module is responsible for finding existing copies of a file to be restored. The identity of the local peer and the file name are used to locate the copies of a file. The peer looking for the backup copy broadcasts a query and each peer with a backup with the time when its backup copy was created. The file searcher module on the originating peer selects the backup that was most recently created prior to the desired time of restoration. The backup/restore manager performs the actual restoration of the file from the selected peer.

The basic P2P search module implements an application-level multicast approach to searching for files in the system, identical to other well-known schemes (Gnutella; Kazaa).

Security Issues

Users may not wish to share all of the files on their computers with other users in the network, and the backup copies of those files would need to be encrypted before being stored at other peers. The encryption can be done using a key known only to the owner of the file. One approach for keeping files safe would be to use public key cryptography. Each user creates a public key and a private key for him-/herself, and files before being backed up are encrypted with the public key. Only the user with the knowledge of the private key can decrypt the backup copy. Within an enterprise with a public key infrastructure such as PKI (Ellison, 1999) deployed, the above mechanism would work well.

Since secure backup does not require interpretation of file by anyone other than the original machine, it is possible to provide security in the backup system by using the more efficient secret key encryption. The backup software can create a secret key for each user or create a separate secret key for each file. The key generation is done by a deterministic algorithm which takes as input (1) a password specified by the user, (2) the identity of the system where the file is created, (3) optionally, the name of the file being backed up, and (4) the time when the copy is being made. Thus, the originating host, when the user supplies the password, can regenerate the key for the file during restoration period and retrieve the file. The algorithm for key generation can be made so as to generate strong cryptographic keys. A typical key generation algorithm could take the concatenation of the four input parameters, compute the exponent of the concatenated binary number, and then take the result modulo a maximum key size as the secret key for encryption. Any secret key-based encryption scheme can then be used to encrypt the file.

Some aspects of key management systems are still needed for the above approach. Users may forget their password for backup and need a way to recover or reset the lost password. In order to assist them in recovering the password, the system maintains a known text file which is then encrypted and stored locally. These two files are not copied to the other peers. When a user forgets his/her password, the system provides utility tools that would help him/her recover the password by trying to compare different possible combinations against the known file. The approach is time-consuming and only works if the known file is not corrupted on the disk.

Note that the password for a file cannot be changed since that would not match with the keys used for prior backups. To work around this limitation, the data management system maintains a local file containing all of the previous passwords and the periods of their usage. This file is always encrypted using the current key and backed up to other machines. On a change of password, the old password is appended to the password file, and the password file is re-encrypted using a key generated by the new password.

Hybrid Data Management Approach

Handling security and managing metadata in peer-to-peer systems is more difficult in peer-to-peer approach than in a centralized backup system. One could use a hybrid approach to obtain the advantages of centralized security management and the cost savings of a peer-to-peer data backup systems.

In the hybrid approach, the metadata for each backup file is maintained at a central repository, while the actual data is maintained at the peers. The central repository is used to manage passwords. The central repository uses a public key cryptographic approach to encrypt the files on the peers. All files are encrypted using the public key of the repository. Thus, the issues of key management reduces to the management of repository keys.

The repository maintains an index of the peers that have the backup copy of the different files, thereby creating a database that can be quickly searched to determine the set of peers with backup copies of a file. Locating peers with backup copies of a file can be done much faster than using a broadcast query search.

All backup and restoration requests are made to the repository. The repository authenticates the identity of the user making the request and searches for a suitable set of peers for the machine. The repository identifies the peers that will make the backup copy and the files are copied directly between the peers. After the successful backing up of a file, the metadata information at the repository is updated.

The repository used in the peer-to-peer approach would require much less bandwidth and data storage than a full-fledged centralized backup solution.

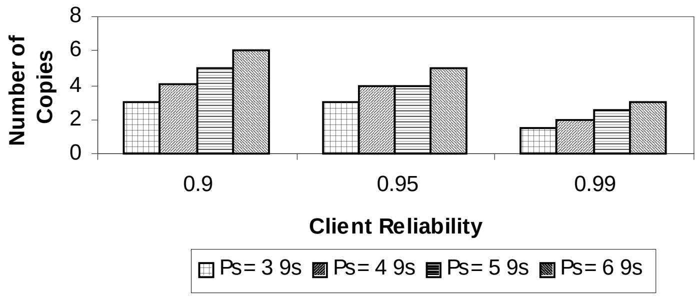
Evaluation of Peer-to-Peer Approach

In this section, we examine the effectiveness of the peer-to-peer approach with some simple system models. The system models we present are more akin to back-of-the-envelop calculations rather than a formal system performance evaluation model. However, they would help in identifying the situations where the peer-to-peer data management approach would be better than the centralized approach and where it would be worse.

In our system model, we assume that there are N machines in the system, and that they are randomly connected. We assume that each of the machines has a disk with the capacity of D and that each machine has a fraction f of its capacity used to create local files. We further assume that b copies of each file are made in the case of the peer-to-peer approach. Furthermore, each client has the probability p_c of being available at any given time, and that each client operates independently of the other clients.

We assume that all optimizations that can be made for data management in peer-to-peer systems can be mimicked in a centralized server approach and vice versa. Thus, the key comparison is between the storage space and bandwidth required for the centralized solution as compared to the peer-to-peer approach. Let us assume that the centralized backup solution provides an availability of p_s.

Figure 2



With *b* copies being made of the system, a client would be able to recover a file when needed as long as one of the *b* peers with a copy of the file is available for it to recover from. Thus, the peer-to-peer system would have a reliability better than that of the centralized system, as long as

$$p_s < 1 - (1 - p_c)^b$$

which is equivalent to

$$b > \log(1 - p_s)/\log(1 - p_c)$$

Figure 2 shows the typical number of copies that would need to be made with some typical availability numbers of clients and the backup server. The different bars show the number of copies needed to obtain the availability expected of a centralized backup system for a given reliability of the clients. For a client reliability of only 90%, 4 copies of a file are able to obtain the availability equivalent to that of a backup server with 4 9s of availability. For a more reliable client, which is available 99% of the times one could obtain availability equivalent to 6 9s at the centralized server with only three copies.

The additional copies of servers come at a cost, since the total storage capacity needed in the peer-to-peer approach is higher than that of the centralized server. Since *b* copies of each file are being made, the space required is *b* times that of a centralized backup server. In order for the peer-to-peer approach to be successful, the peers must have adequate disk space to handle the requisite number of copies. This requires that the following relation hold true:

$$b * f < \alpha$$

where α is a number less than 1 due to practical considerations such as having enough space capacity of clients, or the need to make multiple versions of a file. Thus, a peer-to-peer approach would be feasible as long as

$$\log(1- p_s)/\log(1- p_c) < \alpha/f$$

This would indicate the peer-to-peer approach for data management is feasible provided the utilization of the disk storage at each of the peers is reasonably small. For a utilization of each client of about 20%, and an a of 0.6, Figure 2 would imply that one could obtain 3 9s of availability using clients with an availability of 0.9, and one could obtain 6 9s of availability using clients with an availability of 0.99. If the trend of client workstations with large disk capacity and relatively little usage continues to hold, it would appear that the availability and reliability of the peer-to-peer approach would be comparable to centralized backup approaches for data management.

Conclusion

In this paper, we have presented an peer-to-peer approach to managing data in an enterprise. The approach provides for a way to back up and restore data in enterprise systems without the need for a centralized backup server. We have presented the architecture of a system that would implement this approach, and analyzed the feasibility of the approach using some simple assumptions. The analysis indicates the approach would work well for systems where there are many clients with large amount of disk space capacity. Under the present trend of having PCs and laptops with larger amount of disk storage, this approach appears to be an effective, low-cost method for managing data in an enterprise.

References

Castets, G., Kozakos, G., & Laing, D. (2004). IBM Enterprise Total Storage Server, IBM ITSO Redbook SG-24-5757.

Clark, E. (2002, October). Emerging technology, keeping storage costs under control. *Network Magazine*. From http://www.networkmagazine.com/article/NMG20020930S0004

Ellison, E. (1999). The nature of a usable PKI. *Computer Networks*, *31*, 823–830.

Gnutella. *http://gnuella.wego.com*

Kazaa. *http://www.kazaa.com*

Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., & Kubiatowicz, J. (2003, March). Pond: The OceanStore Prototype. *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*,

Chapter IV

Peer-to-Peer Information Storage and Discovery Systems

Cristina Schmidt
Rutgers University, USA

Manish Parashar
Rutgers University, USA

Abstract

This chapter investigates the current landscape of peer-to-peer (P2P) storage and discovery systems. A classification of existing P2P discovery systems, outlining the advantages and disadvantages of each category, and surveying existing systems in each class, is presented. The design, operation and applications of Squid, a P2P information discovery system that supports flexible queries with search guarantees, are described.

Introduction

Recent years have seen an increasing interest in the peer-to-peer (P2P) computing paradigm, where entities at the edges of the network can directly interact as equals (or peers) and share information, services, and resources without centralized servers. While the P2P architecture is not new (the original Internet was conceived as a P2P system), the proliferation of the World Wide Web and advances in computation and communication technologies have enabled the development and deployment of a large number of P2P applications. These applications take advantage of resources (for example, storage, computing cycles and content) available at user machines at the edges of the Internet. Properties such as decentralization, self-organization, dynamism, and fault tolerance make them naturally scalable and attractive solutions for applications in all areas of distributed computing and communication. Usenet and DNS represent an earlier generation of successful applications with P2P characteristics. More recent P2P applications include content sharing and distribution systems such as Napster (Napster Webpage) and Gnutella (Gnutella Webpage), communication and collaboration systems such as Jabber (Jabber Webpage), Groove (Groove Webpage) and ICQ (ICQ Webpage), systems for anonymity such as Freenet (Clarke, Sandberg, Wiley, & Hong, 2000), censorship-resistant systems such as Publius (Waldman, Rubin, & Cranor, 2000) and embarrassingly parallel computing such as SETI@home (SETI@home Webpage). Furthermore, P2P systems have been recently proposed for resource discovery in computational grids (Andrzejak & Xu, 2002; Foster & Iamnitchi, 2003; Iamnitchi, Foster, & Nurmi, 2002; Li, Xu, Dong, & Zhang, 2002; Schmidt & Parashar, 2003a) and for Web Service Discovery (Schlosser, Sintek, Decker, & Nejdl, 2002; Schmidt & Parashar, 2003b).

This chapter investigates the current landscape of P2P information storage and discovery systems. In this chapter, we first present an overview of P2P storage and discovery systems. We then present a classification of existing systems based on their design and target applications, and survey existing systems in each class. Specifically, we present and compare P2P data lookup and search systems. Finally, we presents Squid (Schmidt & Parashar, 2003a), an information discovery system that supports flexible queries, while providing search guarantees.

P2P Information Storage and Discovery Systems

An important area where P2P technologies continue to be successfully applied is information storage, discovery, and retrieval. Key strengths of the P2P architecture, including the ability to harness peer resources, self-organization and adaptation, inherent redundancy, fault tolerance and increased availability, make it an attractive solution for this class of applications.

P2P storage systems enable persistence and information sharing in a network of peers. The network of peers is typically an overlay network, defined at the application layer of the protocol stack. A peer can have multiple roles. It can be a server (it stores data and responds to query requests), a client (it requests information from other peers), a router (it routes messages based on its routing table), and a cache (it caches data to increase the system's availability). Note that these characteristics describe pure P2P storage and retrieval systems. In hybrid systems, a peer may have fewer roles (for example, some peers are servers, routers, and caches, and others are only clients).

Efficient information discovery in the absence of global knowledge of content and/or naming conventions is a fundamental issue in large, decentralized, distributed sharing environments. The heterogeneous nature and large volume of data and resources, their dynamism, and the size and dynamism of the P2P sharing environment (with nodes joining and leaving) make information discovery a challenging problem. An ideal information discovery system has to support flexible searches (using keywords, wildcards, range queries), be efficient, fault tolerant, self-organizing, and offer guarantees. It should also be able to deal with the scale, dynamism, and heterogeneity of these environments. However, maintaining global knowledge about the current nodes and the content stored in the system is not feasible, and existing P2P information discovery systems implement design trade-offs. These systems are either easy to maintain and allow very complex queries but do not offer any guarantees, or offer guarantees but at the cost of maintaining a complex and constrained structure and supporting less expressive queries.

Another factor is the heterogeneity of the system. Not all nodes have the same capabilities (for example, storage, processing power, bandwidth), and they may choose to share only a fraction of their resources. These capabilities have to be taken into account while distributing content across the network.

Existing P2P information storage/discovery systems can be broadly classified as unstructured or structured based on the underlying network of nodes. While unstructured systems are relatively easy to maintain, can support complex queries, and have been used for information discovery (Iamnitchi et al., 2002; Li et al., 2002), they do not guarantee that all matches to a query in a practical-sized system will be found. Hybrid P2P systems (*Napster Webpage*) provide search guarantees by using centralized directories, which in turn can limit their scalability and may not always be feasible. Structured data lookup systems (Ratnasamy, Francis, Handley, Karp, & Shenker, 2001; Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001) also provide search guarantees and bound search costs, but at the cost of maintaining a rigid overlay structure. Furthermore, these systems only support lookup using unique identifiers (for example, filenames). Structured systems can be extended to support keyword searches using a distributed inverted index (Reynolds & Vahdat, 2003) built on top of the data lookup protocol. In these systems, a data element[1] is mapped to multiple peers, one for each keyword attached to it. As a result, when resolving queries with multiple keywords, all responsible nodes are contacted, all data elements matching each keyword are retrieved, and the partial results are merged and filtered to match the entire query. Typically, range queries are not supported by these systems.

Squid: Flexible Information Discovery with Guarantees

Squid (Schmidt & Parashar, 2003a; Schmidt & Parashar, 2003b) is a P2P information discovery system that supports complex queries containing partial keywords and wildcards, and range queries. It guarantees that all existing data elements that match a query will be found with bounded costs in terms of number of messages and number of nodes involved. The key innovation is a dimension-reducing and locality-preserving indexing scheme that effectively maps the multidimensional information space to physical peers.

The overall architecture of Squid is a distributed hash table (DHT), similar to typical data lookup systems (Ratnasamy et al., 2001; Stoica et al., 2001). The key difference is in the way data elements are mapped to the index space. In existing systems, this is done using consistent hashing, which uniformly maps data element identifiers to indices in the index space. As a result, data elements are randomly distributed across peers without any notion of locality. Squid attempts to preserve locality while mapping the data elements to the index space. In Squid, all data elements are described using a sequence of keywords (common words in the case of P2P storage systems, or values of globally defined attributes—such as memory and CPU frequency—for resource discovery in computational grids). These keywords form a multidimensional keyword space where the keywords are the coordinates and the data elements are points in the space. Two data elements are local if their keywords are lexicographically close or they have common keywords. Thus, we map data elements to a one-dimensional index such that indices that are local in the one-dimensional index space are also local in this multidimensional keyword space. Further, they are mapped to the same node or to nodes that are close together in the overlay network. This mapping is derived from a locality-preserving mapping called Space Filling Curves (SFCs) (Bially, 1967; Butz, 1971; Sagan, 1994).

Note that locality is not preserved in an absolute sense in this keyword space; documents that match the same query (i.e., share a keyword) can be mapped to disjoint fragments of the index space called clusters. These clusters may in turn be mapped to multiple nodes and a query will have to be efficiently routed to these nodes. Using an optimization based on successive refinement and pruning of queries the number of nodes clusters generated for a query is significantly reduced, and as a result, the number of messages sent into the system.

Unlike the consistent hashing mechanisms, SFCs do not necessarily result in a uniform distribution of data elements in the index space—certain keywords may be more popular and hence the associated index subspace will be more densely populated. As a result, when the index space is mapped to nodes, load may not be balanced. Load-balancing techniques are defined to reduce the amount of load imbalance.

Organization of This Chapter

The organization of the rest of this chapter is as follows: Section "The Current P2P Landscape" offers an overview of existing storage and information discovery P2P systems. Section "Squid: Keyword Search with Guarantees in P2P Environments" describes the architecture and operation of Squid. Section "Squid: Application Domain" summarizes the application domain where Squid can be used. Finally, Section "Summary and Conclusions" summarizes and concludes this chapter.

The Current P2P Landscape

Existing P2P systems can be classified, based on their capabilities and the nature of overlay network of peers used, as systems for data lookup and systems for searching. These classes are described and discussed.

Data Lookup Systems

Data lookup systems guarantee that if information exists in the system, it will be found by the peers within a bounded number of hops. These systems build on structured overlays and essentially implement Internet-scale distributed hash tables (DHT).

P2P Overlay Networks and Structured Overlays

The nodes[2] in a typical P2P system self-organize into an overlay network at the application level. Search systems such as Gnutella use an unstructured overlay that is easy to maintain. However, these systems do not scale well as they use a flooding-based search mechanism where each node forwards the query request to all or some of the neighbors. Furthermore, storage systems built on unstructured overlays do not guarantee that all existing information that matches a query will be found. Structured overlay networks address these issues.

In structured overlays, the topology of the peer network is tightly controlled. The nodes work together to maintain the structure of the overlay in spite of the dynamism of the system (i.e., nodes dynamically join, leave and fail). Maintaining the overlay structure can lead to non-negligible overheads. However, data lookup in these systems is very efficient and can be guaranteed. For example, the Chord system (Stoica et al., 2001) organizes nodes in a ring. The cost of maintaining the overlay when a node joins or leaves is $O(\log^2 n)$ in this case, where n is the number of nodes in the system. In addition to this cost, each node runs checks to verify the validity of its routing tables and repairs them if necessary. Similarly, CAN (Ratnasamy et al., 2001) organizes nodes in a d-dimensional toroidal space. Each node in CAN is associated with a hypercube region (zone) and has the nodes that "own" the adjacent hypercubes as its neighbors. The cost of a node join is $O(d)$, and a background algorithm that reassigns zones to nodes makes sure that the structure of the overlay is maintained in case of node failures.

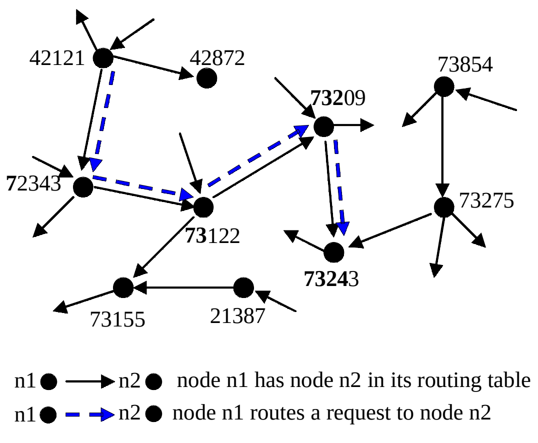
Distributed Hash Tables (DHTs) and DHT Routing

A hash table is a natural solution for a lookup protocol: given an identifier (a key), the corresponding data is fetched. Data stored in hash tables must be identified using unique numeric keys. A hashing function is used to convert the data identifier into a numeric key. In a P2P lookup system, the hash table is distributed among the peers, each peer storing a part of it. Typically, consistent hashing (Karger, Lehman, Leighton, Levine, Lewin, & Panigrahy, 1997) is used to ensure a uniform distribution of load among the peers.

As DHTs build on structured overlays, the placement of data and the topology of the overlay network are highly structured and tightly controlled, allowing the system to provide access guarantees and bounds. The mapping of data elements to indices in a DHT-based system is based on unique data identifiers (for example, filenames) and only these identifiers can be used to retrieve the data. Keyword searches and complex queries are not efficiently supported by these systems. Examples of data lookup systems include Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron & Druschel, 2001), and Tapestry (Zhao, Kubiatowicz, & Joseph, 2001). These systems have been used to construct persistent storage systems, cooperative distributed file systems, distributed name systems, cooperative Web caches, and multicast systems. For example, the PAST (Druschel & Rowstron, 2001) persistent distributed storage system is built on Pastry and the CFS (Dabek, Kaashoek, Karger, Morris, & Stoica, 2001), a cooperative distributed file system, builds on Chord. However, the lack of support for keyword searches can significantly limit the applicability and scalability of these systems, as the data identifiers have to be unique and globally known.

A DHT essentially implements one operation, lookup (key), which routes the request to the peer responsible for storing the key. The routing algorithm depends on the overlay used and its efficiency has a direct impact on the scalability of the system. Each node in a DHT-based P2P lookup system has a unique numeric identifier, which is a string of bits of a specified length. The data keys are also strings of bits and are of the same length. Furthermore, each node in the system maintains a routing table containing pointers to a small number of other nodes. When a node receives a query for a key that is not stored locally, it routes the query to the node in its routing table that places the query "closest" to the destination. "Closeness" is measured differently from system to system, and it is typically a function of the identifier of the current node and the key. Selected P2P DHT systems are summarized.

Figure 1. Example of routing from node 42121 to node 73243 in a Plaxton mesh


42121
42872
73854
73209
72343
73275
73122
73243
73155
21387
n1 ● → n2 ● node n1 has node n2 in its routing table
n1 ● - - → n2 ● node n1 routes a request to node n2

Plaxton et al: Plaxton et al. (Plaxton, Rajaraman, & Richa, 1997) developed an algorithm for accessing replicated shared objects in a distributing system where each access is satisfied by a nearby copy. In the Plaxton mesh, each node is assigned a unique integer label. Furthermore, each object (data) has a unique identifier and is replicated at multiple nodes in the system. The node whose identifier shares the longest prefix with the object identifier is considered the "root" for that object. Object replicas form a virtual height-balanced tree, which is embedded one-to-one into the overlay. Each node of the tree keeps information associated with the copies of the object residing in its subtree.

Each node in the system maintains a routing table and a pointer list. The pointer list contains pointers to copies of some objects in the network. The pointer list is updated only when an object is inserted or deleted. An insert request for object *A* originated at node *x* will update the pointer lists at nodes whose labels share a prefix subsequence with *A*'s identifier.

The Plaxton routing algorithm is based on prefixes. In this algorithm, a single digit of the destination's identifier is resolved at a time. For example, if the node 72343 receives a request with a key **7**3241 (the key and the node identifier have the first digit in common), the node will route the message to a neighbor that shares the first two digits with the key (for example, **73**122). Figure 1 illustrates the routing of a message from node 42121 to node 73243. To enable this routing, each node has to maintain a routing table organized into routing levels, with each level pointing to nodes that match a prefix of its own identifier and each level increasing the length of the common prefix. For example the node 72343 will have the following nodes in its routing table: (level0: 0…, 1…, 2…, etc.), (level1: **7**0..., **7**1..., **7**2..., etc.), (level2: **72**0..., **72**4..., **72**5..., etc.), ..., and so forth. The size of the routing table at each node is O(log *n*), where *n* is the number of nodes in the system, and a data lookup requires at most O(log *n*) hops.

Tapestry: Plaxton's algorithm (Plaxton et al., 1997) assumes a relatively static node population, which makes it unsuitable for P2P systems. Tapestry (Zhao et al., 2001) is a P2P overlay routing infrastructure that uses a variant of the Plaxton algorithm that is more appropriate for a dynamic node population. The basic routing mechanism in Tapestry follows the Plaxton algorithm described above. Each node maintains a neighbor table, organized into routing levels, and a backpointer list that points to nodes where the current node is a neighbor. The references to objects are stored in the same way as in the Plaxton system.

Additionally, Tapestry addresses fault tolerance using a soft-state approach: (1) heartbeat messages are exchanged to discover failed nodes, and a failed node is given a *second chance*: it is not removed from the routing tables of other nodes until after a period of time; (2) each object has multiple roots (the single root in the Plaxton scheme is a single point of failure); and (3) references to objects expire and have to be renewed periodically, so eventually there are no references that point to nonexistent objects.

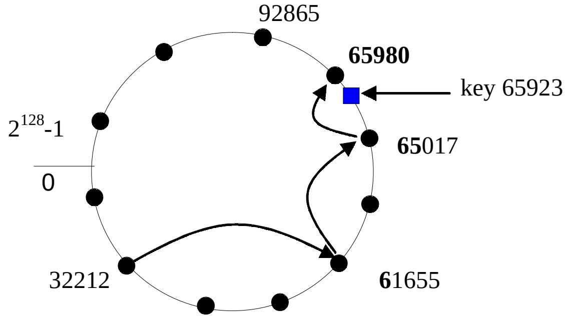
The management of a node in Tapestry is described below:

- Node Joins: The joining node has to know at least one node that is already part of the system. The node sends a join request, together with its identifier, into the system, which routes it to its root node. The root node helps the new node to create its routing table.
- Node Departures: Before a node leaves the system, it sends messages of its intention to all the nodes on its backpointer list, along with a replacement node for each routing level from its own routing table. The notified nodes update their routing tables accordingly.
- Node Failures: Tapestry addresses this problem by building redundancy into routing tables and object location references. When failures are detected, a repairing mechanism is triggered, which redistributes and replicates object location references.

Pastry: In Pastry (Rowstron & Druschel, 2001), each peer node has a unique identifier from a circular index space, ranging from 0 to 2^{128}-1. The unique node identifiers are randomly generated at node join, and uniformly distributed in the identifier index space. Data are mapped to nodes based on unique identifiers called keys. A data element is mapped to the node whose identifier is numerically closest to its key. For example, in Figure 2, data with key 65923 is mapped to node 65980. Similarly, data with key 54211 would be mapped to node 65017.

Each node in Pastry maintains a routing table, a neighborhood set, and a leaf set. The routing table is organized into levels (rows) similar to the Plaxton approach.

Figure 2. Example of routing in Pastry (the figure illustrates routing from node 32212 to the node whose identifier is closest to the key 65923)



The neighborhood set maintains information about nodes that are closest to the local nodes according to a proximity metric (for example, number of IP hops). The leaf set is the set of nodes with numerically closest identifiers (half larger and half smaller than the identifier of the current node).

The routing algorithm is prefix based. The Pastry node routes a message to the node in its routing table with an identifier that is numerically closest to the destination key (for example, it has the longest shared prefix with the key). Figure 2 illustrates the routing process.

The size of a routing table in Pastry is O(log *n*), where *n* is the number of active Pastry nodes. The expected number of routing hops is O(log *n*). Pastry attempts to minimize the distance traveled by the message by taking network locality into account.

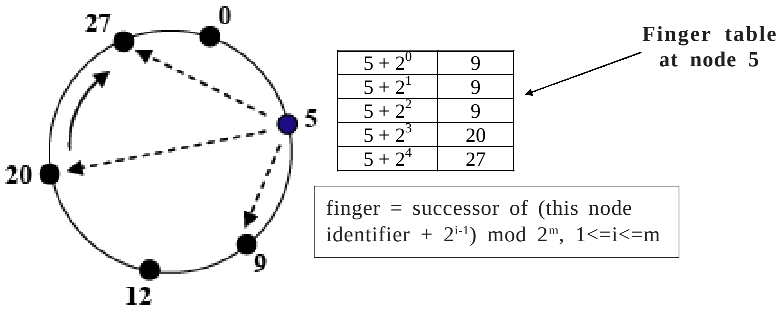
Node joins, departures, and failures are managed as follows:

- Node Joins: The joining node must know of at least another node already in the system. It randomly generates an identifier for itself, and sends a join request to the known node. The request will be routed to the node whose identifier is numerically closest to the new node's identifier. All the nodes encountered on route to the destination will send their state tables (routing table, neighborhood set, and leaf set) to the new node. The new node will initialize its own state tables, and it will inform appropriate nodes of its presence.
- Node Departures and Failures: Nodes in Pastry may fail or depart without any notice. Other nodes detect references to failed or departed nodes when they attempt to contact the failed or departed nodes. They repair their state tables using information from other nodes.

Chord: In the Chord overlay network (Stoica et al., 2001), each node has a unique identifier ranging from 0 to 2^m-1. These identifiers are arranged as a circle modulo 2^m, where each node maintains information about its successor and predecessor on the circle. Additionally, each node also maintains information about (at most) *m* other neighbors, called *fingers*, in a finger table. The i^{th} finger node is the first node on the circle that succeeds the current node by at least 2^{i-1}, where $1 \le i \le m$. The finger table is used for efficient routing.

Data elements are mapped to nodes based on their keys (identifiers). A data element is mapped to the first node whose identifier is equal to or follows its key. This node is called the *successor* of the key. Consider a sample overlay network with 6 nodes and an identifier space from 0 to 2^5-1, as shown in Figure 3. In this example, data elements with keys 6, 7, 8, and 9 will map to node 9, which is the

Figure 3. Example of Chord overlay network (each node stores the keys that map to the segment of the curve between itself and its predecessor node)



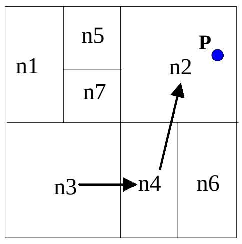
$5 + 2^0$	9
$5 + 2^1$	9
$5 + 2^2$	9
$5 + 2^3$	20
$5 + 2^4$	27

finger = successor of (this node identifier + 2^{i-1}) mod 2^m, $1<=i<=m$

successor of these keys. The management of node joins, departures, and failures using this overlay network and mapping scheme are as follows.

- Node Joins: When a node wants to join the overlay network, it has to know about at least one node that is already in the network. The joining node chooses an identifier from the identifier space (0 to 2^m-1) and sends a join message with this identifier to the known node. For example, the identifier may be computed based on the node's IP address. The join message is routed across the overlay network until it reaches the node that is the successor of the new node based on its chosen identifier. The joining node is inserted into the overlay network at this point and takes on part of the successor node's load. The new node constructs its finger table and the direct neighbor entries to its successor and predecessor nodes. It also updates the finger tables at other nodes in the system that should point to itself. The cost for a node joining the network is O($\log^2 n$) messages, where *n* is the number of nodes in the system.
- Node Departures: When a node leaves the system, the finger tables of nodes that have entries pointing to the leaving node have to be updated. The cost for updating these tables is O($\log^2 n$) messages where *n* is the number of nodes in the system, that is, the same as the cost of a join.
- Node Failures: When a node fails, the finger tables that have entries pointing to the failed node will be incorrect. In order to maintain correct finger tables, each node periodically runs a stabilization algorithm where it chooses a random entry in its finger table and updates it.
- Data Lookup: The data lookup protocol based on Chord can very efficiently locate nodes based on their content. A node forwards requests to the

Figure 4. Example of CAN overlay network with seven nodes (Data mapped to point P is stored at node n2. The figure illustrates routing from node n3 to node n2.)



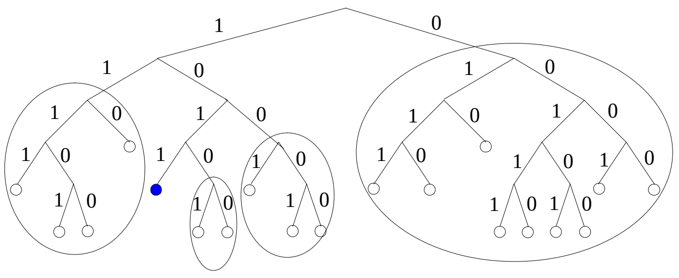
neighbor that is closest to the key. Data lookup takes O(log *n*) hops, where *n* is the total number of nodes in the system.

CAN: CAN (Ratnasamy et al., 2001) uses a *d*-dimensional Cartesian coordinate space on a *d*-torus. This coordinate space is dynamically partitioned among all the nodes in the system. Each node is associated with a zone in this coordinate space, and its neighbors are nodes that "own" adjacent zones. Figure 4 illustrates a system with seven nodes using a two-dimensional coordinate space.

Each data element is deterministically mapped to a point in this coordinate space using a uniform hash function. The data element is stored at the node that owns the zone in which the point lies. For example, in Figure 4, the data element mapped to point P is stored at node n2. The management of node joins, departures, and failures is done as follows:

- Node Joins: The joining node randomly chooses a point *P* in the coordinate space and sends a join message with the destination *P*. The CAN overlay routes the message to the node that is responsible for point *P*. The destination node splits its zone and gives half to the new node.
- Node Departures: The departing node's zone is merged with one of its neighbors if possible; otherwise, it is assigned to the neighbor with the smallest region. To avoid the fragmentation of the space, a background zone-reassignment algorithm is used.
- Node Failure: Nodes exchange periodic update messages. If a node does not send update messages to its neighbors for a prolonged period of time, the neighbors assume that that node has failed, and they initiate a takeover mechanism.

Figure 5. Kademlia binary tree (The leaves are nodes in the system; the black dot represents the node that has prefix 1011. The circles show the subtrees where the node with prefix 1011 must know some other node.)



- Data Lookup: Routing consists of forwarding the request to a neighbor that is closest to the key. Figure 4 illustrates routing from node **n3** to node **n2**. In CAN, nodes have O(d) neighbors and the data lookup cost is O($dn^{1/d}$),where n is the number of nodes in the system.

Kademlia: Kademlia (Maymounkov & Mazieres, 2002), like the systems presented above, uses identifiers for nodes and keys for data, both taken from a b-bit identifier space (in this case b = 160). The (key, data-value) pairs are stored at nodes with identifiers close to the key. The routing algorithm uses node identifiers to locate the node that stores a given key. A key innovation in Kademlia is the use of the XOR metric (a symmetric, non-Euclidean metric).

Kademlia treats nodes as leaves in a binary tree, where each node's position is determined by the shortest unique prefix of its identifier. Any node in the system views the tree as a set of successively lower subtrees that do not contain the node. The node must know about some other node in these subtrees. Figure 5 presents the system from the node with prefix 1011's perspective.

The routing table at each node consists of b k-buckets (i.e., 160 k-buckets). Each k-bucket keeps a list of k nodes at a distance between 2^i and 2^{i+1} from the current node, where $0 \leq i \leq 160$. k is a system-wide replication parameter, chosen such that any given k nodes are very unlikely to fail within an hour of each other. The entries in a bucket are ordered based on "last seen" time, the least recently seen nodes at the head, and the most recently seen nodes at the tail. The buckets are updated when the node receives a message (request or reply) from another node. The alive least recently seen nodes have priority; they are not replaced by most recently seen nodes if the bucket is full. This policy is based on the fact that the longer a node has been up, the more likely it is to remain up for another hour (S. Saroiu, 2002).

A (key, data-value) pair is stored at *k* nodes in the system that are closest to the key, according to the XOR metric. To find a data-value based on a key, a node initiates a recursive procedure to locate the *k* closest nodes to the key. The initiator sends lookup messages to α nodes from its closest *k*-bucket. These nodes respond with at most *k* known nodes which are closest to the destination. The initiator picks some of these nodes and sends them lookup requests. The process continues until the nodes with the data are located.

The (key, data-value) pairs need to be republished periodically to ensure their persistence in the system. Nodes that hold a copy of the pair can fail or leave the system, and/or other nodes with identifiers closer to the key can join the system. When a node joins the system, it must store any (key, data-value) pair for which it is one of the *k* closest nodes. These values are provided by the nodes that learn about the joining node.

- *Node Joins*: The new node borrows some neighbors from a node it knows about, and sends a lookup message with its identifier. The lookup procedure informs other nodes of the new node's existence, and helps the new node with its routing table update. The cost of a join is O(log *n*), where *n* is the number of nodes in the system.
- *Node Departures and Failures*: No action is needed.
- *Data Lookup*: As for all DHT-based systems, data lookup takes O(log *n*), and the request is forwarded to a neighbor that is closest to the destination, using the XOR metric to measure the closeness. While existing data is not guaranteed to be found, there is a high probability that any lookup is successful.

Discussion

The systems presented above represent only a subset of all the proposed data lookup systems and the number of these systems continues to grow. Most variants of DHT-based systems try to optimize the data lookup cost, the routing table size and the maintenance costs. These systems must be as efficient as possible while tolerating faults (i.e., a fraction of nodes can fail at the same time without data loss and routing anomalies). Bounds and trade-offs between the above-mentioned metrics are presented in Kaashoek and Karger (2003).

A serious problem in structured P2P systems is the discrepancy between the physical structure of nodes and the overlay. It is possible that two nodes running on machines in the same room do not have a link between them in the overlay, and the message exchange between those nodes involves geographically remote nodes spanning multiple continents. Ideally, the overlay network should match the physical network. Three solutions to the network proximity problem in the P2P overlays have been considered (Castro, Druschel, Hu, & Rowstron, 2002). In the first solution, proximity routing (Rowstron & Druschel, 2001; Xu, Tang, & Zhang, 2003), the overlay is constructed without regard for the physical network. During routing, when several neighbors exist, the routing algorithm selects the node that is close in the physical network and that moves the message closer to the destination. In the second approach, topology-aware overlay, the overlay is constructed such that neighboring nodes are also close in the physical network (Ratnasamy, Handley, Karp, & Shenker, 2002). This approach destroys the uniform distribution of the node identifier space. The third approach, proximity neighbor section, also constructs a topology-aware overlay, but to avoid the nonuniform assignment of node identifiers, a node selects the nearest nodes in each desired portion of the ID space for its routing table (Rowstron & Druschel, 2001). This approach works if the overlay is flexible about the neighbor selection (for example, Tapestry (Zhao et al., 2001) and Pastry (Rowstron & Druschel, 2001)). Chord (Stoica et al., 2001) and CAN (Ratnasamy et al., 2001) are rigid in this aspect, they require specific nodes as neighbors.

Search Systems

Search systems are P2P storage systems that allow complex keyword queries using partial keywords, wildcards, and ranges. P2P search systems can be classified based on their overlay topology as unstructured systems, which provide loose guarantees, structured systems, which provide strong guarantees, and hybrid server-based systems. Unstructured search systems are composed of Gnutella-type systems (Gnutella Webpage). Structured search systems are relatively recent, and are built on top of data lookup systems. P2P search systems are discussed.

Hybrid Search Systems

Hybrid P2P systems such as Napster (Napster Webpage) use centralized directories to index the information/data in the system, allowing them to support very complex queries. In these systems, the search is centralized while the data exchange is distributed. These systems are considered hybrid as they employ elements of both pure P2P systems and client/server systems. The key disadvantage of using centralized directories is that they limit system scalability. However, hybrid systems continue to be successful with recent systems such as KazaA (*KazaA Webpage*) and Morpheus (Morpheus Webpage).

Napster: In Napster (Napster Webpage), a server node indexes files held by a set of users. Users search for files at a server, and when they find a file of interest, they download it directly from the peer that stores the file. Users cannot search for files globally—they are restricted to searching at a single server that typically indexes only a fraction of the available files.

Morpheus: In Morpheus (Morpheus Webpage), nodes that have sufficient bandwidth and processing power are automatically elected as Super Nodes, using proprietary algorithms and protocols called the FastTrack P2P Stack (P2P FastTrack Stack). Super Nodes index the content shared by local peers that connect to it and proxy search requests on behalf of these peers. The file downloads are peer-to-peer as in Napster.

Unstructured Keyword Search Systems

Unstructured keyword search P2P systems such as Gnutella (Gnutella Webpage) are ad hoc and dynamic in structure and data location in these systems is random. The typical query mechanism in these systems consists of forwarding the query to as many peers as possible. Each node receiving a query processes it and returns results, if any. A query may return only a subset (possibly empty) of available matches, and information present in the system may not be found. Providing search guarantees in these systems requires that all peers in the system be queried, which is not feasible in most systems. Furthermore, the probability of finding a piece of data is directly proportional to its replication count. The mechanism used to route the query message in the network has direct impact on the scalability of the system. The approach described above is called limited-scope flooding, and does not scale. Other approaches such as Expanding Ring Search, Random Walks (Lv, Cao, Cohen, Li, & Shenker, 2002), and Routing Indices (Crespo & Garcia-Molina, 2002) improve search scalability. Furthermore, the number of results found can be improved using data replication. However, even with these optimizations, one cannot find all matches in a typical system. Unstructured keyword search systems are primarily used for community information sharing. Examples include Gnutella (Gnutella Webpage) and Freenet (Clarke et al., 2000).

Gnutella and Optimizations: Gnutella uses TTL[3]-based flooding to search for data. This scheme introduces a large number of messages, especially if each node has a large number of neighbors. Furthermore, selecting the appropriate TTL so that the requested data is found using a minimum number of messages is not trivial. To address the TTL selection problem, successive floods with different TTL values are used. A node starts the flood with a small TTL, and if the search is not successful, the TTL is increased and another flood is initiated. This process is called Expanding Ring Search. Random Walks (Lv et al., 2002) further improve the search efficiency. Instead of forwarding a query request to all the neighbors, a node forwards the query to *k* randomly chosen neighbors at each step until matching data is found. Another approach, Probabilistic Search Protocol (Menasce, 2003) forwards the query to each neighbor with a certain probability *p*, called the broadcast probability. The Routing Indices (Crespo & Garcia-Molina, 2002) approach allows nodes to forward queries to neighbors that are more likely to have the answers. If a node cannot answer a query, it forwards the query to a subset of its neighbors based on its local Routing Index, rather than selecting neighbors at random. The Routing Index is a list of neighbors, ranked according to their "goodness" for the query.

Freenet: Freenet (Clarke et al., 2000) was designed to prevent the censorship of documents and to provide anonymity to users. Unlike Gnutella, which uses a breadth-first search (BFS) with depth TTL, Freenet uses a depth-first search (DFS) with a specified depth. Each node forwards the query to a single neighbor and waits for a response from the neighbor before forwarding the query to another neighbor (if the query was unsuccessful) or forwarding the results back to the query source (if the query was successful).

Structured Keyword Search Systems

Structured keyword search P2P systems are built on top of a data lookup protocol. Their goal is to combine the complex queries supported by unstructured and hybrid systems with the efficiency and guarantees of data lookup systems. Four systems in this class, pSearch (Tang, Xu, & Mahalingam, 2003), a system proposed by Reynolds and Vahdat (Reynolds & Vahdat, 2003), Squid (Schmidt & Parashar, 2003a), and a system proposed by Andrzejak and Xu (Andrzejak & Xu, 2002) are summarized below.

pSearch: pSearch (Tang et al., 2003) is a P2P storage system that supports content- and semantics-based searches. It is built on top of CAN (Ratnasamy et al., 2001) and uses the Vector Space Model (VSM) and Latent Semantic Indexing (LSI) (Berry, Drmac, & Jessup, 1999) to index the documents. It uses LSI to store the documents in CAN using their semantic vectors as keys. The queries are solved using the query semantic vectors as keys, contacting the node that stores the key, and flooding the query to nodes within a radius *r* of the destination.

VSM represents documents and queries as vectors. Each component of the vector represents the importance of a word (term) in the document query. The weight of a component is usually computed using the TF*IDF (term frequency * inverse document frequency) (Berry et al., 1999) scheme. When a query is issued, the query vector is compared to all document vectors. The ones closest to the query vector are considered to be similar and are returned. One measure of similarity is the cosine between the vectors.

The VSM scheme computes the lexical similarity between the query and the documents. LSI is used to compute the semantic similarity. LSI uses singular value decomposition (SVD) (Berry et al., 1999) to transform and truncate the matrix of document vectors computed by VSM to discover the underlying semantic terms and documents. As a result a hi-dimensional document vector is transformed into a medium-dimensional semantic vector by projecting the former into a medium-dimensional semantic subspace. The similarities between the query and the documents are once again measured as the cosine of the angle between their vector representations. The system does not support partial keywords, wildcards, and range queries.

Reynolds and Vahdat: Reynolds and Vahdat (2003) propose an indexing scheme for P2P storage systems such as Chord (Stoica et al., 2001) and Pastry (Rowstron & Druschel, 2001). They build an inverted index, which is distributed across the nodes using consistent hashing (Karger et al., 1997), and use Bloom filters to reduce bandwidth consumption during querying. An inverted index is a data structure that maps a keyword to the set of documents that contain the keyword. A Bloom filter (Reynolds & Vahdat, 2003) is a hash-based data structure that summarizes membership in a set. The membership test returns false positives with a tunable, predictable probability and never returns false negatives. The number of false positives falls exponentially as the size of the Bloom filter increases.

In this system, data elements are described using a set of keywords. Each keyword is hashed to a node in the overlay, where a reference to the data element is stored in an inverted index. A query consisting of multiple keywords is directed to multiple nodes. The results are merged at one of the nodes, and the final result is sent to the user. The system does not support partial keywords, wildcards, and range queries.

Squid: Squid (Schmidt & Parashar, 2003a) is a P2P information discovery system that uses a structured overlay (for example, Chord) and a Space Filling Curve (SFC)-based indexing scheme. It guarantees that all existing data elements that match a query will be found with reasonable costs in terms of number of messages and number of nodes involved. It also supports searches with partial keywords, wildcards, and range queries.

In order to efficiently support complex queries, lexicographic locality must be preserved in the DHT index space. Typically, DHTs use hashing mechanisms that do not preserve data locality; Squid builds its index using SFCs, which are proven to preserve data locality.

In systems based on distributed inverted indices, queries with multiple keywords are typically directed to multiple nodes, and the number of the nodes involved in a query increases as the size of the query (for example, number of keywords) increases. In Squid, the reverse takes place: longer queries require fewer nodes to be contacted. Consequently, Squid is better suited for multi-keyword searches. The architecture and operation of Squid is discussed in more detail in Section "Squid – Keyword Search with Guarantees in P2P Environments."

Andrzejak and Xu: Andrzejak and Xu propose a discovery system based on Hilbert SFC (Andrzejak & Xu, 2002). Unlike Squid (Schmidt & Parashar, 2003a), this system uses the inverse SFC mapping, from a one-dimensional space to a *d*-dimensional space, to map a resource to peers based on a single attribute (for example, memory). It uses CAN (Ratnasamy et al., 2001) as its overlay topology, and the range of possible values for the resource attribute (one-dimensional) is mapped onto CAN's d-dimensional Cartesian space. This system is designed for resource discovery in computational grids, more specifically to enhance other resource discovery mechanisms with range queries. In contrast, Squid uses SFCs to encode the *d*-dimensional keyword space to a one-dimensional index space.

Summary

Table 1 summarizes the landscape of P2P information storage/discovery systems presented above, highlighting the advantages and disadvantages of each class of systems. An ideal search system should support complex queries, offer guarantees, be easy to maintain, and solve queries efficiently (for example, contact few peers, not waste bandwidth, have a good query response time, etc.). Since these properties can lead to conflicting requirements, existing systems implement trade-offs. The trade-offs depend on the nature of the application and the targeted user community. If the shared information is critical (for example, a medical database), the system must guarantee that all existing data that matches the query is found, even at the cost of maintaining a structured overlay. If the shared information is noncritical, for example, music files, the system may not offer guarantees, and in turn focus on the cost and ease of maintenance.

Table 1. A summary of the P2P information storage/discovery landscape

P2P SYSTEM	PROS & CONS	EXAMPLES
Data Lookup: Use structured overlay networks and consistent hashing to implement Internet-scale distributed hash tables. Data placement and overlay topology are highly structured. Data lookup only, using unique data identifiers (for example, filenames)	**Pros:** Provide efficient lookups with guarantees and bounded search costs **Cons:** Complex queries (keywords, wildcards, ranges) not supported. High structured overlay maintenance costs	Plaxton (Plaxton et al., 1997), Chord (Stoica et al., 2001), CAN (Ratnasamy et al., 2001), Pastry (Rowstron & Druschel, 2001), Tapestry (Zhao et al., 2001), Kademlia (Maymounkov & Mazieres, 2002)
Hybrid: Unstructured system using centralized directories to address search guarantee problem	**Pro:** Can support complex queries **Cons:** Limited scalability. May not be feasible for very large systems	Napster (Napster Webpage), Morpheus (Morpheus Webpage), Kazaa (Kazaa Webpage)
Unstructured: Gnutella like systems. Use flooding techniques to process queries by forwarding them to neighboring peers. Flooding algorithms used include "limited-scope flooding," "expanding ring search," "random walks" (Lv et al., 2002), "best neighbor," "learning based" (Iamnitchi et al., 2002), and "routing indices" (Crespo & Garcia-Molina, 2002). Data replication is used to improve number of results.	**Pros:** Overlay networks are easy to maintain. Can support complex queries and different querying system **Cons:** No search or cost guarantees. All machining information for a query may not be found in a practical-sized system	Gnutella (Gnutella Webpage), Freenet (Clarke et al., 2000), A. Iamnitchi et al. (Iamnitchi et al., 2002), Li et al. (Li et al., 2002), Crespo and Garcia-Moline (Crespo & Garcia-Molina, 2002), Yang and Garcia-Molina (Yang & Garcia-Molina, 2002).
Structured Keyword Search: Build on data lookup systems and construct distributed indices to support keyword searches. These systems use schemes such as Vector Space Model (VSM), Latent Semantic Indexing (LSI), Distributed Inverted Indices, and Space-Filling Curves.	**Pros:** Can support keyword searches with guarantees and bounded costs. Some of them support wildcards and range queries **Con:** High-structured overlay maintenance costs	pSearch (Tang et al., 2003), Reynolds and Vahdat (Reynolds & Vahdat, 2003), Squid (Schmidt & Parashar, 2003a), Andrzejak and Xu (Andrzejak & Xu, 2002)

Squid: Keyword Search with Guarantees in P2P Environments

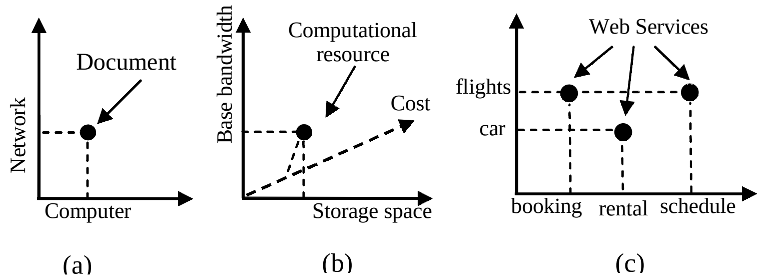
Squid (Schmidt & Parashar, 2003a) is an information discovery system that supports flexible querying using keywords, partial keywords, wildcards and ranges, while providing search guarantees and bounded costs. It is based on data lookup systems (Ratnasamy et al., 2001; Stoica et al., 2001), and essentially implements an Internet-scale distributed hash table (DHT). The key difference is in the way the data elements are mapped to the DHT key space. In existing systems, this is done using consistent hashing, which restricts querying to data element lookup using exact identifiers. Squid uses a locality preserving, dimension-reducing mapping called a Space Filling Curve (SFC) (Bially, 1967), (Sagan, 1994), allowing it to support more complex queries.

The Squid architecture consists of the following components: (1) a locality-preserving mapping that maps data elements to indices, (2) an overlay network topology, (3) a mapping from indices to nodes in the overlay network, (4) load-balancing mechanisms, and (5) a query engine for routing and efficiently resolving keyword queries using successive refinements and pruning. These components are described below.

Constructing an Index Space: Locality-Preserving Mapping

A key component of a data lookup system is defining the index space and deterministically mapping data elements to this index space. To support complex keyword searches in Squid, each data element is associated with a sequence of keywords and a mapping that preserves keyword locality is defined. The keywords are common words in the case of P2P storage systems, or values of globally defined attributes of resources in the case of resource discovery.

These keywords form a multidimensional keyword space where data elements are points in the space and the keywords are the coordinates. The keywords can be viewed as base-*n* numbers, for example *n* can be 10 for numeric keywords or 26 if the keywords are English words. Two data elements are considered local if they are close together in this keyword space. For example, their keywords are lexicographically close (*comput*er and *comput*ation) or they have common keywords. Not all combinations of characters represent meaningful keywords, resulting in a sparse keyword space with nonuniformly distributed clusters of data elements. Examples of keyword spaces are shown in Figure 6. A key requirement for efficiently supporting complex queries using partial keywords, *Figure 6. (a) A 2-dimensional keyword space for a storage system. The data element* "Document" *is described by keywords* "Computer:Network". *(b) A 3-dimensional keyword space for storing computational resources, using the attributes: storage space, base bandwidth and cost. (c) A 2-dimensional keyword space. There are three web services described by keywords: (booking, flights), (schedule, flights), (rental, car).*



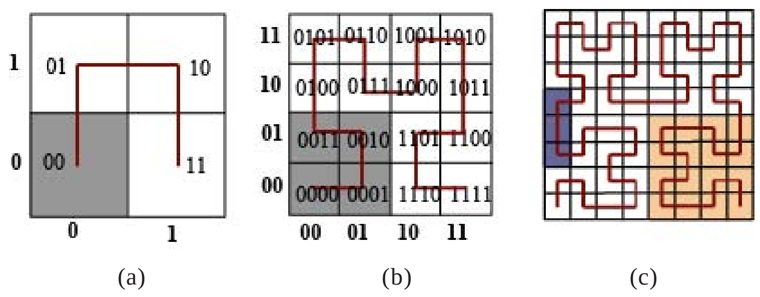
wildcards, and ranges in a DHT-based system is that the DHT index space preserve locality. Furthermore, a recursive index space enables these queries to be optimized using successive refinement and pruning. Such an index space is constructed using the Hilbert SFC as described below.

Space Filling Curves

A Space Filling Curve (SFC) is a continuous mapping from a *d*-dimensional space to a one- dimensional space (f: $N^d \rightarrow N$). The *d*-dimensional space is viewed as a *d*-dimensional cube partitioned into sub-cubes, which is mapped onto a line such that the line passes once through each point (sub-cube) in the volume of the cube, entering and exiting the cube only once. Using this mapping, a point in the cube can be described by its spatial coordinates, or by the length along the line, measured from one of its ends. The construction of SFCs is recursive, as illustrated in Figure 7. Each refinement of the curve is called an approximation.

An important property of SFCs is digital causality, which follows from its recursive nature. A unit length curve constructed at the k^{th} approximation has an equal portion of its total length contained in each sub-cube; it has n^{k*d} equal segments. If distances along the line are expressed as base-*n* numbers, then the numbers that refer to all the points that are in a sub-cube and belong to a line segment are identical in their first $(k-1)*d$ digits. This property is illustrated in Figure 7 (a) and (b). In (a), the sub-cube (0, 0) with SFC index 00 is refined, resulting in four sub-cubes in (b), each with the same first two digits as the original sub-cube.

Figure 7. Space-filling curve approximations for d = 2: n = 2 (a) First order approximation (b) Second order approximation (c) 3rd order approximation. The colored regions represent three-cell and 16-cell clusters.



SFCs are also locality preserving. Points that are close together in the one-dimensional space (the curve) are mapped from points that are close together in the *d*-dimensional space. The reverse property is not true; adjacent sub-cubes in the *d*-dimensional space may not be adjacent or even close on the curve. A group of contiguous sub-cubes in *d*-dimensional space will typically be mapped to a collection of segments on the SFC. These segments called clusters are shown in Figure 7 (c).

In Squid, SFCs are used to generate the one-dimensional index space from the multidimensional keyword space. Applying the Hilbert mapping to this multi-dimensional space, each data element can be mapped to a point on the SFC. Any range query or query composed of keywords, partial keywords, or wildcards can be mapped to regions in the keyword space and corresponding clusters in the SFC.

Mapping Indices to Peers

The next step consists of mapping the one-dimensional index space onto an overlay network of peers. Squid currently builds on the Chord (Stoica et al., 2001) overlay network topology. In Chord, each node has a unique identifier ranging from 0 to 2^m-1. These identifiers are arranged as a circle modulo 2^m. Each node maintains information about (at most) *m* neighbors, called *fingers*, in a *finger table*. The finger table is used for efficient routing and enables data lookup with O(log *n*) cost (Stoica et al., 2001), where *n* is the total number of nodes in the system.

In Squid, node identifiers are generated randomly. Each data element is mapped, based on its SFC-based index or key, to the first node whose identifier is equal to or follows the key in the identifier space. This node is called the *successor* of the key.

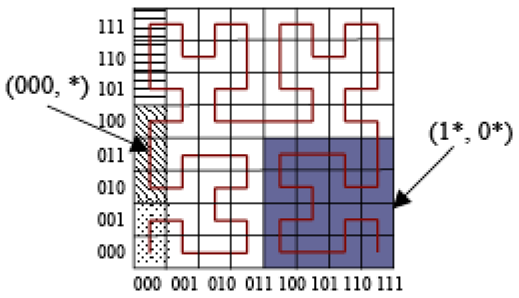
The Query Engine

The primary function of the query engine is to efficiently process user queries. As described above, data elements in the system are associated with a sequence of up to *d* keywords, where *d* is the dimensionality of the keyword space. The queries can consist of a combination of keywords, partial keywords, or wildcards. The expected result of a query is the complete set of data elements that match the user's query. For example, (computer, network), (computer, net*) and (comp*, *) are all valid queries. Another query type is a range query where at least one dimension specifies a range. For example, if the index encodes memory, CPU frequency and base bandwidth resources, the following query (256 - 512MB, *, 10Mbps - *) specifies a machine with memory between 256 and 512 MB, any CPU frequency, and at least 10Mbps base bandwidth.

Query Processing: Processing a query consists of two steps: translating the keyword query to relevant clusters of the SFC-based index space, and querying the appropriate nodes in the overlay network for data elements. For example, in Figure 8, the query (000, *) identifies three clusters and the query (1*, 0*) identifies one cluster.

Each cluster may contain zero, one or more data elements that match the query. Depending on its size, an index space cluster may be mapped to one or more adjacent nodes in the overlay network. A node may also store more than one cluster. Once the clusters associated with a query are identified, straightforward

*Figure 8. Regions in 2-dimensional space defined by the queries (000, *) and (1*, 0*)*



query processing consists of sending a query message for each cluster. A query message for a cluster is routed to the appropriate node in the overlay network as follows. The overlay network provides a data look-up protocol: given an identifier for a data element, the node responsible for storing it is located. This mechanism can be used to locate the node responsible for storing a cluster using a cluster identifier. The cluster identifier is constructed using SFC's digital causality property. The digital causality property guarantees that all the cells that form a cluster have the same first *i* digits. These *i* digits are called the cluster's *prefix* and form the first *i* digits of the *m* digit identifier. The rest of the identifier is padded with zeros.

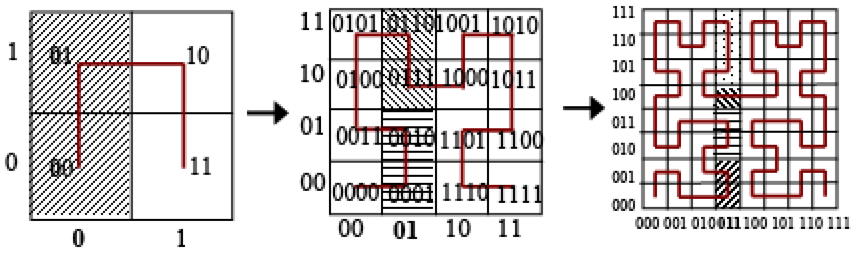
Query Optimization: For complex queries (for example, wildcards, ranges), the number of clusters can be very high, and sending a message for each cluster is not a scalable solution. Query processing can be made scalable using the observation that not all the clusters that correspond to a query represent valid keywords, as the keyword space and the clusters are typically sparsely populated with data elements. The number of messages sent and nodes queried can be significantly reduced by filtering out the useful clusters early. However, useful clusters cannot be identified at the node where the query is initiated. The solution is to consider the recursive nature of the SFC to distribute the process of cluster generation at multiple nodes in the system, the ones that might be responsible for storing them.

Since the SFC generation is recursive and clusters are segments on the curve, these clusters can also be generated recursively. This recursive process can be viewed as constructing a tree. At each level of the tree the query defines a number of clusters, which are refined, resulting in more clusters at the next level. The tree can be embedded into the overlay network such that the root performs first query refinement, and each node further refines the query, sending the resulting subqueries to the appropriate nodes in the system. Query optimization consists of pruning clusters from the tree during the construction phase.

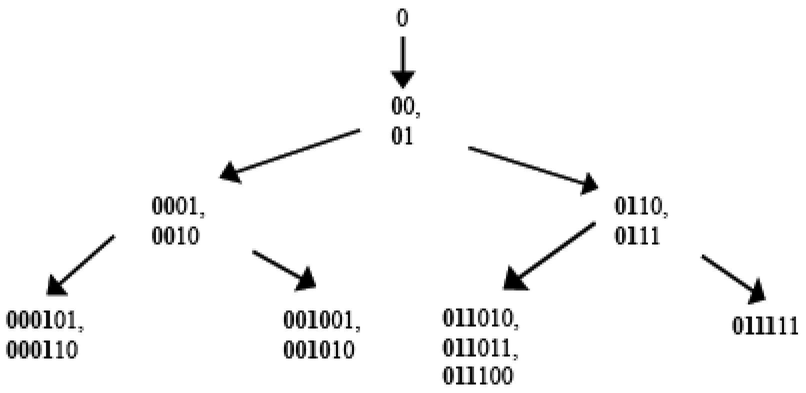
Figure 9 illustrates the query processing, using the sample query (011, *), in a two-dimensional space using base-2 digits for the coordinates. Figure 9 (a) shows the successive refinement for the query and Figure 9 (b) shows the corresponding tree. The leaves of the tree represent all possible matches for the query.

Figure 9 (c) illustrates the optimization process. The leftmost path (solid arrows) and the rightmost path (dashed arrows) of the tree are embedded into the ring overlay network topology. The overlay network uses six digits for node identifiers. The arrows are labeled with the prefix of the cluster being resolved. The query is initiated at node 111000. The first cluster has prefix 0, so the cluster's identifier will be 000000. The cluster is sent to node 000000. At this node the

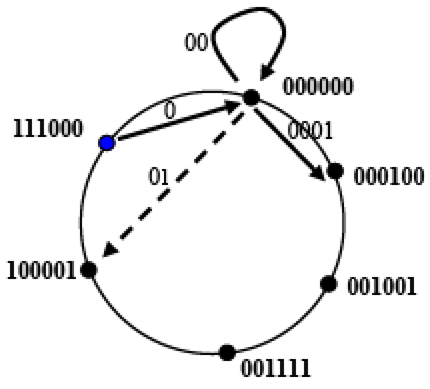
*Figure 9: Resolving query (011, *): (a) recursive refinement of the query – first, second and third order Hilbert curve; (b) the refinement process, viewed as a tree – each node is a cluster, and the characters in bold represent cluster's prefixes; (c) embedding the leftmost tree path (solid arrows) and the rightmost path (dashed arrows) onto the overlay network topology.*



(a)



(b)



(c)

cluster is further refined, generating two subclusters with prefixes 00 and 01. The cluster with prefix 00 remains at the same node. After processing, the subcluster 0001 is sent to node 000100. The cluster with prefix 01 and identifier 010000 is sent to node 100001 (dashed line). This cluster will not be refined because the node's identifier is greater than the query's identifier, and all matching data elements for the subclusters (the whole subtree) should be stored at the node.

Load Balancing

Unlike the consistent hashing mechanisms used by data lookup systems, SFCs do not necessarily result in uniform distribution of data elements in the index space—certain keywords may be more popular and hence the associated index subspace will be more densely populated. As a result, when the index space is mapped to nodes, load may not be balanced. Additional load balancing has to be performed. Two load-balancing steps are described below.

- Load Balancing at Node Join: At join, the incoming node generates several identifiers (for example, 5 to 10) and sends multiple join messages using these identifiers. Nodes that are logical successors of these identifiers respond reporting their load. The new node uses the identifier that will place it in the most loaded part of the network. In this way, nodes will tend to follow the distribution of the data from the very beginning. With *n* identifiers being generated, the cost to find the best successor is O(*n**log N), and the cost to update the tables remains the same, $O(\log^2 N)$. However, this step is not sufficient by itself. The runtime load balancing algorithms presented below improve load distribution.
- Load Balancing at Runtime: The runtime load-balancing step consists of periodically running a local load-balancing algorithm between few neighboring nodes. Two load-balancing algorithms are proposed. In the first algorithm, neighboring nodes exchange information about their loads, and the most loaded nodes give a part of their load to their neighbors. The cost of load-balancing at each node using this algorithm is $O(\log^2 N)$. As this is expensive, this load-balancing algorithm cannot be run very often.

  The second load-balancing algorithm uses virtual nodes. In this algorithm, each physical node houses multiple virtual nodes. The load at a physical node is the sum of the load of its virtual nodes. When the load on a virtual node goes above a threshold, the virtual node is split into two or more virtual nodes. If the physical node is overloaded, one or more of its virtual nodes can migrate to less loaded physical nodes (neighbors or fingers).

Squid: Application Domain

Squid (Schmidt & Parashar, 2003a) can be used to support flexible information discovery in a number of different application domains. A selection of applications domains are presented below.

P2P Information Sharing Systems

P2P information sharing systems enable persistence and sharing of information in a network of peers. These systems enable the end users to interact directly and share content (for example, music, software, data). The systems are made possible by the recent advances in information technology (storage, computational power, and bandwidth) and the associated drop in costs. All participants in these systems are equal: they all share content stored locally, request content shared by others, and process queries issued by other participants. A key component of the P2P sharing system is the search engine, which must support searches based on complex queries (for example, keywords, wildcards, ranges), and provide some guarantees as to the completeness of the search results.

Squid satisfies these requirements. Content described using keywords can be indexed and stored in the P2P network so that locality is maintained. The content can then be queried using keywords, partial keywords, and wildcards.

Resource Discovery in Computational Grids

Grid computing is rapidly emerging as the dominant paradigm of wide-area distributed computing (Foster & Kesselman, 1999). Its overall goal is to realize a service-oriented infrastructure for the sharing of autonomous and geographically distributed hardware, software, and information resources (for example, computers, data, storage space, CPU, software, instruments, etc.) among user communities. Grids present similar challenges as P2P systems: large scale, lack of globally centralized authority, heterogeneity (resources, sharing policies), and dynamism (Iamnitchi et al., 2002).

A fundamental problem in grid systems is the discovery of information and resources. A number of P2P resource discovery systems for grid environments have been recently proposed. These include both unstructured Gnutella-like systems (Iamnitchi et al., 2002) and structured systems that offer guarantees (Schmidt & Parashar, 2003a; Andrzejak & Xu, 2002). Squid can be used to complement current grid resource discovery mechanisms by enhancing them with range query capabilities. Computational resources can be described in multiple ways, for example, as URIs or XML files. Typically, they have multiple attributes such as memory and CPU frequency and the values of these globally defined attributes of computational resources are considered as keywords. These keywords are used to index the descriptions of the computational resources, and to store and query the resources in Squid.

Web Service Discovery

Web services are enterprise applications that exchange data, share tasks, and automate processes over the Internet. They are designed to enable applications to communicate directly and exchange data, regardless of language, platform and location. A typical Web service architecture consists of three entities: service providers that create and publish Web services, service brokers that maintain a registry of published services and support their discovery, and service requesters that search the service broker's registries.

Web service registries store information describing the Web services supported by the service providers, and can be queried by the service requesters. These registries are critical to the ultimate utility of the Web services and must support scalable, flexible and robust discovery mechanisms. Registries have traditionally had a centralized architecture (for example, UDDI (Universal Description, Discovery and Integration, 2000)) consisting of multiple repositories that synchronize periodically. However, as the number of Web services grows and become more dynamic, such a centralized approach quickly becomes impractical. As a result, there are a number of decentralized approaches that have been proposed. These systems (for example, (Schlosser et al., 2002)) build on P2P technologies and ontologies to publish and search for Web service descriptions.

Squid (Schmidt & Parashar, 2003b) can be used for building dynamic, scalable, decentralized registries with real-time and flexible search capabilities to support Web service discovery. Web services are described using WSDL, and can be characterized by a set of keywords. These keywords can then be used to index the Web service description files and the index is stored at peers in Squid.

Discovery of Newsgroups in Bulletin Board News Systems

Bulletin Board Systems (BBSs) are worldwide distributed discussion systems. A BBS consists of a set of newsgroups. Newsgroups are typically classified hierarchically by subject and contain collections of messages with a common theme. Users can discover and connect to newsgroups (for example, interest groups), and subscribe to and publish messages on the newsgroups. Usenet is one of the earliest BBSs. Recently, the number of BBSs has grown and they have become a popular means of communication.

Squid can be used to discover interest groups in a flexible manner. A newsgroup is indexed based on keywords, where the keywords are the components in its hierarchical name.

Decentralized Information Sharing for Investigative Research and Discovery

The emerging data-/information-driven scientific and engineering applications are based on an ad hoc and opportunistic integration, aggregation, and sharing of diverse (simulation, observational, and experimental) and distributed data spread across multiple campuses, institutions, cities, and states. However, the robust and scalable discovery of this scientific data remains a challenge. Raw scientific data can be characterized using automated vision and visualization techniques and can be described using domain-specific metadata consisting of keywords, keyword-value pairs, and keyword-range pairs. This keyword-based metadata can then be used to index and discover the data using P2P information discovery techniques. Squid is being used to realize such a decentralized information discovery and sharing collaboratory for cancer tissue micro-array and chemical combustion research (Schmidt, Parashar, Chen, & Foran, 2004).

Summary and Conclusions

This chapter presented an overview of the existing landscape of P2P storage and information discovery systems. It classified existing systems based on their capabilities and the overlay network they use and discussed the advantages, disadvantages, and trade-offs of each class. Examples of systems from each class of systems were also presented. Squid (Schmidt & Parashar, 2003a), a P2P information discovery system supporting flexible querying with search guarantees was described.

P2P systems offer a potential solution for implementing scalable, distributed storage and discovery systems. These systems take advantage of resources such as storage, computing cycles and content available at user machines at the edges of the Internet. Properties such as decentralization, self-organization, dynamism and fault tolerance make them naturally scalable and attractive solutions for applications in all areas of distributed computing and communication.

Data lookup systems implement Internet-scale distributed hash tables, use structured overlays, offer guarantees, and allow data location based on exact identifiers. The data location is very efficient in these systems, but they have to pay the price of maintaining a structured overlay. These systems are naturally load balanced since they employ consistent hashing to map data to peers. Search systems, on the other hand, allow more complex queries and may use structured or unstructured overlays. Hybrid systems are very successful in today's Internet, and they are used mainly for file sharing. Pure P2P systems based on unstructured overlay (for example, Gnutella), can be inefficient and expensive; however, ongoing research is addressing this problem. These systems are easy to maintain but do not offer guarantees.

The final category of search systems consists of distributed indices built on top of data lookup systems. These systems are very promising as they move toward the ideal: allow complex queries and offer guarantees. They build on data lookup systems and inherit their efficient data location characteristics as well as the costs of maintaining the structured overlay.

References

Andrzejak, A., & Xu, Z. (2002, September). Scalable, efficient range queries for grid information services. *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing,* Sweden, 33–40.

Berry, M., Drmac, Z., & Jessup, E. (1999). Matrices, vector spaces and information retrieval. *SIAM Review, 41*(2), 335–362.

Bially, T. (1967). A class of dimension changing mapping and its application to bandwidth compression. Unpublished PhD thesis, Polytechnic Institute of Brooklyn.

Butz, A.R. (1971). Alternative algorithm for Hilbert's space-filling curve. *IEEE Transactions on Computers, April,* 424–426.

Castro, M., Druschel, P., Hu, Y.C., & Rowstron, A. (2002, June). Exploiting network proximity in peer-to-peer overlay networks. *Proceedings of the International Workshop on Future Directions in Distributed Computing,* Bertinoro, Italy, 52–55.

Clarke, I., Sandberg, O., Wiley, B., & Hong, T. (2000, June). Freenet: A distributed anonymous information storage and retrieval system. *Proceed-*

ings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, 311–320.

Crespo, A., & Garcia-Molina. (2002, July). Routing indices for peer-to-peer systems. *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria, 23–32.

Dabek, F., Kaashoek, M., Karger, D., Morris, R., & Stoica, I. (2001, October). Wide cooperative storage with CFS. *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Banff, Canada, 202–215.

Druschel, P., & Rowstron, A. (2001, May). Past: A large-scale, persistent peer-to-peer storage utility. *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, Schoss Elmau, Germany.

Foster, I., & Iamnitchi, A. (2003, February). On death, taxes, and the convergence of peer-to-peer and grid computing. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA.

Foster, I., & Kesselman, C. (1999). *The GRID – Blueprint for a new computing infrastructure*. San Francisco: Morgan Kaufmann.

Gnutella Webpage. Retrieved from http://gnutella.wego.com/

Groove Webpage. Retrieved from http://www.grove.net/

Iamnitchi, A., Foster, I., & Nurmi, D. (2002). *A peer-to-peer approach to resource discovery in grid environments* (Technical Report TR-2002-06). Chicago: University of Chicago Press.

ICQ Webpage. Retrieved from http://web.icq.com/

Jabber Webpage. Retrieved from http://www.jabber.org/

Kaashoek, M.F., & Karger, D.R. (2003, February). Koorde, a simple degree-optimal distributed hash table. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA.

Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., & Panigrahy, R. (1997, May). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, El Paso, TX, 654–663.

KazaA Webpage. Retrieved from http://www.kazaa.com/

Li, W., Xu, Z., Dong, F., & Zhang, J. (2002). Grid resource discovery based on a routing-transferring model. *Proceedings of the 3rd International Workshop on Grid Computing*, Baltimore, MD, 145–156.

Lv, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002, June). Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th International Conference on Supercomputing*, New York, 84–95.

Maymounkov, P., & Mazieres, D. (2002, March). Kademlia: A peer-to-peer information system based on the XOR metric. *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, 53–65.

Menasce, D.A. (2003). Scalable P2P search. *IEEE Internet Computing, 7*(2).

Morpheus Webpage. Retrieved from http://www.morpheus-os.com/

Napster Webpage. Retrieved from http://napster.com/

P2P FastTrack Stack. Retrieved from http://www.fasttrack.nu/

Plaxton, C., Rajaraman, R., & Richa, A.W. (1997, June). Accessing nearby copies of replicated objects in a distributed environment. *Proceedings of the 9th ACM Symposium on Parallelism in Algorithms and Architectures*, Newport, RI, 311–320.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001, August). A scalable content-addressable network. *Proceedings of the ACM Special Interest Group on Data Communications Conference*, San Diego, CA, 161–172.

Ratnasamy, S., Handley, M., Karp, R., & Shenker, S. (2002, June). Topologically-aware overlay construction and server selection. *Proceedings of the Infocom*, New York, NY, 1190–1199.

Reynolds, P., & Vahdat, A. (2003, June). Efficient peer-to-peer keyword searching. *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brazil, 21–40.

Rowstron, A., & Druschel, P. (2001, November). Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 329–350.

Sagan, H. (1994). *Space-filling curves*. Springer-Verlag.

Saroiu, S., Gummandi, P.K., & Gribble, S.D. (2002, January). A measurement study of peer-to-peer file sharing systems. *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, CA.

Schlosser, M., Sintek, M., Decker, S., & Nejdl, W. (2002, September). A scalable and ontology-based P2P infrastructure for semantic Web services. *Proceedings of the Second International Conference on Peer-to-Peer Computing*, Linköping, Sweden, 104-111.

Schmidt, C., & Parashar, M. (2003a, June). Flexible information discovery in decentralized distributed systems. *Proceedings of the 12th High Performance Distributed Computing*, Seattle, WA, 226–235.

Schmidt, C., & Parashar, M. (2003b). A peer-to-peer approach to Web service discovery. *World Wide Web Journal, 7*(2), 211–229.

Schmidt, C., Parashar, M., Chen, W., & Foran, D. (2004, June). Engineering a peer-to-peer collaboratory for tissue microarray research. *Proceedings of the Challenges of Large Applications in Distributed Environments Workshop*, Honolulu, HI, 64-73.

SETI@home Webpage. Retrieved from http://setiathome.ssl.berkeley.edu/

Stoica, I., Morris, R., Karger, D., Kaashoek, F., & Balakrishnan, H. (2001, August). Chord: A scalable peer-to-peer lookup service for Internet applications. *Proceedings of the ACM Special Interest Group on Data Communications Conference*, San Diego, CA, 149–160.

Tang, C., Xu, Z., & Mahalingam, M. (2003). pSearch: Information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review, 33*, 89–94.

Universal Description, Discovery and Integration (2000). *UDDI technical white paper*. From http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

Waldman, M., Rubin, A.D., & Cranor, L.F. (2000, August). Publius: A robust, tamper-evident, censorship-resistant Web publishing system. *Proceedings of the 9th USENIX Security Symposium*, Denver, CO, 59-72.

Xu, Z., Tang, C., & Zhang, Z. (2003, May). Building topology-aware overlays using Global Soft-State. *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, RI, 500–508.

Yang, B., & Garcia-Molina, H. (2002, July). Improving search in peer-to-peer systems. *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 5-14.

Zhao, B.Y., Kubiatowicz, J., & Joseph, A.D. (2001). *Tapestry: An infrastructure for fault-tolerant wide-area location and routing* (Technical Report UCB/CSD-01-1141). Berkeley: Computer Science Division, University of California.

Endnotes

1 We will use the term *data element* to represent a piece of information that is indexed and can be discovered. A data element can be a document, a file, an XML file describing a resource, an URI associated with a resource, and so forth.

2 In this chapter “node” and “peer” are used interchangeably.

3 TTL (Time-to-Live), limits the living time of a message. Typically, TTL is the number of hops a message can travel until it is discarded.

Section I

Systems and Assets: Issues Arising from Decentralized Networks in Security and Law

Chapter V

Peer-to-Peer Security Issues in Nomadic Networks

Ross Lee Graham
Mid-Sweden University, Sweden

Abstract

The security concerns for a peer-to-peer implementation of nomadic networks are described with special emphasis on taxonomy and on their identification. The underpinning reference question is how to provide a wide scope of secure networking functions in a completely decentralized and unmanaged network (an ad hoc mobile nomadic network). Though many of the identified security concerns apply to any overlay network, the complete selection of concerns is based on their possible importance in an implementation such as the AmIGo Project collaboration for a ubiquitous nomadic network.

Introduction

For personal devices and ubiquitous interfaces, the problem to solve is how to provide a wide scope of secure networking functions in a completely decentralized and unmanaged network (an ad hoc mobile nomadic network).

There are two available network technology modes that address this problem; wireless Internet access (WAP, Bluetooth, wireless data, etc.) and wireless broadband access (LMDS, MMDS) (Barkai, 2001).

Peer-to-peer implementations are based on having every node treated as an access point (AP). The Internet access mode is typically based on an IP network. The broadband access mode is based on mobile radio networks. Both systems have similar structures in their routing algorithms.

The routing architecture is a central module within the unmanaged networks. The routing algorithms find paths on a dynamic topology. These routing architectures open new vulnerabilities to security (see below).

Both networks show similarities in their routing principles. These algorithms adapt to the dynamic self-organization characteristics that these networks exhibit (Barkai, 2001).

The Motivation for Peered Networks

If we look at the design of the Internet, the essentials of its first form were anticipated for the U.S. Department of Defense (DOD) in 1964, by Paul Baran at Rand Corporation. This preparatory work identified the general specifications and proposed solutions for the main concerns of the DOD. Lawrence G. Roberts from MIT was the main architect for ARPANET (Advanced Research Projects Agency Network). However, he states directly that the Rand work (including Baran) had no significant impact on the ARPANET plans and Internet history. Even so, Roberts admits to reading Baran's work and meeting with him (in 1967) prior to designing the overall network, network economic analysis, line optimization, and the selection of computer sites to be connected. Furthermore, ARPANET was consistent with the DOD requirements identified in the Rand work (Baran, 1964; Roberts, 2003).

ARPANET was a DOD project and the DOD had three major concerns:

- Concern 1: Security of the physical network

  Critical services must not be concentrated in a few vulnerable failure points. In Baran, this concern was expressed in terms of the possible destruction (through bombing) of the network communication system. The strategy was to design a network in which there are no central points of failure, a network where if part of the system were destroyed, the remainder would still function. In effect, Baran formulated a network reliability model in terms of physical destruction. For Roberts, reliability was also one of the key network issues.

Their second concern was sharing information between different makes and models of computers. This requirement demands interoperability.

- Concern 2: Interoperability of a variety of machines

  This implies that the network is functionally independent of the various architectures at its nodes. Therefore, either the networks speak the language of all the different machines or all the machines speak the language of the network, or some mix of these possibilities. In practice, it has proven more effective to make the machines speak the language of the network.

The third concern was for protection against interception of transmissions.

- Concern 3: Security of the information

  These three concerns functioned as guidelines for the DOD sponsorship of the ARPANET Project. Though Roberts denies the ARPANET design as having any relation to Baran's work at Rand, it is a fact that the DOD sponsored the Rand work before it sponsored the MIT work. This suggests that for the DOD, the Rand work defined a requirements basis for assessing the MIT work (Baran, 1964; Roberts, 2003).

In turn, the work on ARPANET led to defining the requirements for the development of TCP/IP (Roberts, 2003). The first version of TCP was written by Vint Cerf and Bob Kahn in 1974.

The original decentralized vision of ARPANET still survives in TCP/IP. There are two important features of TCP/IP that provide for this decentralization:

1. End node verification
2. Dynamic routing

For end node verification, all the network nodes operate with equal rights (like peers). It is the end nodes that verify the transmissions. There is no central control for overseeing the communications.

For dynamic routing, note that dynamic routing addresses the third concern as well, because connections can follow multiple paths. Baran devised the packet (message block) method to facilitate this implementation. There were three independent packet research efforts—Leonard Klinerock, MIT; Paul Baran, Rand Corporation; and Donald Davies, National Physical Laboratory in the UK—and the Davies name “packet” was chosen by Roberts (2003).

With no central control and the ability to follow multiple paths, if a sector is bombed (Baran starting point) or otherwise nonfunctional (Roberts starting point), TCP/IP packet routing can follow whatever other routes that are functional.

In effect, this is why congestion in large degree is handled by TCP/IP. Congestion at routing nodes puts the nodes into a nonoperational state for incoming packets, and so forth, therefore, another route is chosen for the packet. If the destination node is congested, this is not a feature that TCP/IP was originally designed to handle.

Today, there are new types of congestion because of the client/server domination in the Internet. In other words, the original notion of decentralization has given way to interconnected centralized servers (a topology that Bluetooth mimics in miniature). This offers challenges to TCP/IP for which it was not originally designed.

Leaf-node peer-to-peer in some respects is a step back in time, where decentralization reaches its ultimate end at the edges of the Internet. In the old days, the terminals were workstations for time sharing and not regarded as leaf-nodes. It was the mainframes that the terminals were connected to that may be placed as leaf-nodes. Leaf-nodes in peer-to-peer connectivity require computational power.

Peer-to-peer is in part a return to the original design specifications of TCP/IP. However, there are some differences today that require consideration.

We have taken this point of view to show that peer-to-peer and the original specifications of TCP/IP have much in common. This helps explain why the Internet carries peer-to-peer connectivity so well.

Finally, with respect to TCP/IP, peer-to-peer can use virtual addressing methods that allow peer-to-peer networks to ride on any network configuration that uses

IP addressing. In effect, this allows the formation of ad hoc overlay networks that are functionally independent of the networks they ride on (History, 2003).

Distributed Networks

Baran (1964) clearly conceived and defined a distributed network. He was given the task to model a network with specifications that answered to the requirements for the three concerns.

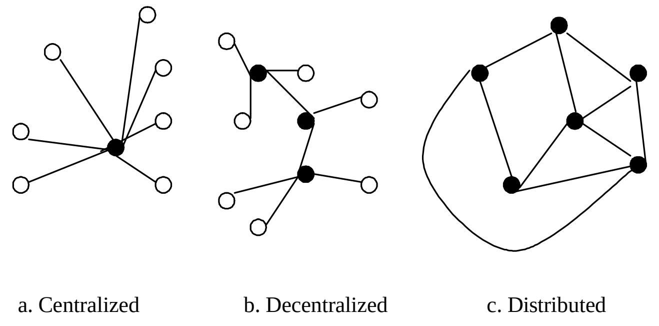
He classed computer networks into three categories: centralized, decentralized, and distributed (see Figure 1.1).

In the first two networks, the leaf-nodes are connected to access points. These access points are also called *centers* when leaf-nodes are connected to them for network access. A centralized network is a star structure of leaf-nodes with one center. If this center node fails, the entire network fails because it has only one center point of failure. A decentralized network is a cluster of stars with connected centers. It, therefore, has multiple center points of possible failure. In a purely distributed network, every node is an access point and there are no leaf-nodes, that is, no stars. Therefore, there are no center points of possible failure. Since every node is an access point, it might be argued that every point is a center point. We deny this argument by definition because the center points must have leaf-nodes.

Definition: In a purely distributed network where every node is an access point we have a peer-to-peer network.

Note that this definition implies that there is no central point of failure.

Figure 1.1. Paul Baran's original categories for network architectures



Distributed Computing

Distributed computing can be implemented on any network architecture. However, it needs to be kept clearly distinct from distributed networks. Distributed network denotes a network architecture type and is classed as an infrastructure. Distributed computing is a mode of using a network infrastructure and need not be restricted to distributed networks.

Bluetooth

Bluetooth operations are based on proximity RF. It allows transparent wireless communications in static or dynamic network topologies. This is not a new technology; it is a new use of existing technologies. The Bluetooth chip can operate on both circuit and packet switching modes. It provides both synchronous and asynchronous data services.

Bluetooth uses piconets in scatternets. In a piconet, a device is picked as master and the rest serve as slaves. Some of these slaves can function as masters in other piconets, and so forth. In a scatternet, multiple masters must establish links to each other. The scatternet concept is used as a clustering of piconets and piconets are used as a clustering of individual devices. From this arrangement we can see that we have the masters as the bottlenecks in a Bluetooth network (Bluetooth, 2003).

In a pure peer-to-peer network the client/server, master/slave, concepts do not apply. In pure peer-to-peer, there is no master nor slave. Therefore, Bluetooth may not be seen as P2P at the transmission level due to the master–slave architecture. However, the master–slave paradigm only addresses the physical exchange of information and may not relate to applications that run on top of Bluetooth-enabled devices. These applications may therefore employ P2P functionality. This is similar to the way that P2P rides on the Internet using virtual addressing.

P2P implementations are based on having every node treated as an AP. In the AmIGo (Ambient Intelligence to Go) Project, the use of Bluetooth may qualify as P2P connectivity because of the limitations imposed by the constraints of the individual communication devices (currently Nokia 7650), which limit communications to a one-to-one basis. In the Bluetooth architecture, every master can function as a slave to another master. Theoretically, this suggests that Bluetooth network connectivity could itself be trimmed to a P2P architecture if every slave can also function as a master. The piconets then become overlays on a network of nodes that are reversible according to their direction of connectivity (master $\rightarrow$ slave). This eliminates leaf slaves when every node is embedded in a distributed network, which can be represented as a directed graph structure without a leaf. This model needs investigation for its theoretical and practical limits.

Other Wireless Platforms

Wireless IP-based connectivity without cells opens the way for peer-to-peer implementations, both as overlay networks (virtual networks) as well as the hardware (physical networks) they ride on. Both the virtual and the physical wireless networks have topologies that may be static or dynamic. In wireless topologies both the overlay network and the physical network can be made congruent in their topologies and this can be done ad hoc.

Today, there are many choices for wireless networks (Toh, 2002), for example:

(Big LEO, 2003) (LEO = Low Earth Orbit):
<www.ee.surrey.ac.uk/Personal/L.Wood/constellations/tables/tables.html>

(Little LEO, 2003):
<www.itu.int/newsarchive/press/WRC97/little-leo.html>

(CSMA/AMPS Cellular, 2003):
<www.computer.org/proceedings/hicss/1435/volume9/14350299abs.htm>

(GSM-PCS, 2003):
<www.cellular.co.za/gsmoverviewpage.htm>

(Metricom, 2003):
<www.metricom-corp.com/>

(SkyTel 2-Way, 2003):
<www.skytel.com/>

(TDMA-PCS, 2003):
<www.iec.org/online/tutorials/pcs/topic01.html>

(Wireless LANs, 2003):
<www.computer.org/students/looking/summer97/ieee802.htm>

Ad Hoc Nomadic Mobile Applications

Mobile ad hoc devices can recognize the presence of other wireless devices through sensing the presence of neighborhood beacons. Here, neighborhood is defined as those devices within the RF range of the sensor. From this, we derive the concept of ad hoc nomadic mobile applications.

Examples:

- In the home: exiting or entering, set security alarm on or off, set temperature control, set lights control, set entertainment centers control, set computer telephone answering service, and so forth.
- In the car: receive e-mail, telephone calls, roadside shopping information, receive directions to various services or places, make reservations, and so forth.
- It can also cover speed control, toll collection, and synchronization of traffic lights.
- In the office: download e-mails, update schedules, track working hours, and so forth.
- While shopping: RF tags (they are cheap) can be used for price, sale, and stock information, as well as automatic shopping searches by location (distance) and prices.
- While traveling: information, schedules, reservations, payment, and automatic check-in all available in real time, no waiting in line except for luggage check-in/check-out or transport entry/exit.

P2P Security Issues (Trust Management)

P2P applications introduce a whole new class of security issues, security within the virtual P2P networks themselves and security with respect to the environments they ride on.

Leaf-nodes in mobile ad hoc networks communicate in a self-organizing dynamic topology. The properties of these networks generate new vulnerabilities that are unknown in the hard-wired infrastructures. This section identifies vulnerabilities.

In the case of P2P connections that form overlay networks (virtual networks), security must first be based on the infrastructure. The deeper we can run P2P on the TCP/IP stack, the less vulnerable the system is to attack. However, we are faced with a trade-off between vulnerability and interoperability. The deeper in the stack that we run P2P, the larger the interoperability problem when we have a many-standard system.

It is well documented that the wireless environment is prone to failures and disconnection and this makes the exchange of data less reliable. When this applies to security, the problems may be more critical with regard to the key management since situations such as lost keys or impossibility to communicate with the owner/distributor of keys may appear. Aside from the specific requirements on the security mechanisms, the use of security in wireless transmission also puts requirements on the communication characteristics. The overload of the communication channel, the need for high data rates, the efficient management of the connection, and communication time all need to be addressed.

We have chosen the threat modeling and risk assessment approach to security systems inspired by Bruce Schneier: "You first need to define the threat model, then create a security policy, and only then choose security technologies that suit" (Schneier, 2000).

Vulnerabilities in Wireless Routing Protocols

The wireless routing protocols have common features that we exemplify here by the specific consideration of the Dynamic Source Routing (DSR) protocol. DSR is for routing mobile ad hoc networks.

DSR can be attacked by any combination of targeting:

a. availability
b. integrity
c. confidentiality
d. nonrepudiation
e. authentication
f. access control

Buchegger and Le Boudec (2002) found that these may:

a. obtain incorrect forwarding (works until upper layers find out);
b. obtain bogus routing information or traffic control;
c. salvage a route that is not broken (if the salvage bit is not set, it will look like the source is still the original one);
d. choose a very short reply time, so the route will be prioritized and stay in the cache longer;
e. set good metrics of bogus routes for priority and remaining time in the cache;
f. manipulate flow metrics for the same reason;
g. not send error messages in order to prevent other nodes from looking for alternative routes;
h. use bogus routes to attract traffic to intercept packets and gather information;
i. deny service attack caused by overload by sending route updates at short intervals; and
j. use promisuous mode to listen in on traffic destined for another node.

With the exception of the last threat, promiscuous listening, all of the threats listed correspond to attacks that the monitoring component in each node can detect either at once or at least when they happen repeatedly.

Free-Riders and Accountability (Turcan & Graham, 2001)

a. Noncooperation
   - Forwarding: There is a trade-off between cooperation and resource consumption. For example, in an IP-based ad hoc network, forwarding packets consumes resources. Since ad hoc networks are generally not constrained by an organization that can enforce cooperation, this generates a need to develop incentives to cooperate in the forwarding of messages (packets).
   - Security Risk: Nodes may boycott communications. This is equivalent to a denial of service.

b. Malicious Intent
   - Routing: Routes should be advertised and set up adhering to a protocol that can reflect the structure of the network topology.
   - Security Risk: Nodes may forward messages to collusion partners for analysis, disclosure, or money. This is not only a denial of the expected service; it is also an invasion of privacy with malicious intent.

   Noncooperation and malicious intent should be sanctioned. They must be held accountable in a manner that is to their disadvantage.

c. Prevention, Detection, and Reaction: If we could make a perfect prevention mechanism we would have a utopian level of security. All threats to a system are in terms of vulnerabilities and this includes the possibility of bypassing the in-place prevention mechanisms. Given this reality, developing methods for detection and response is essential (Schneier, 2000).

Security Threats from P2P (The Environments They Ride On)

1. Steganography: Using a freeware utility called Wrapster, a P2P wrapping tool, a user can disguise a .zip file containing source code as an MP3. An accomplice could then download the "music" file from the user's PC. Millions of dollars worth of proprietary information can be stolen with the appearance of a relatively common abuse of company resources.
2. Encryption Cracking: Using P2P architecture and over 100,000 participants (using only idle CPU time), Distributed.net was able to test 245 billion keys per second to break the 56-bit DES encryption algorithm in 22 hours and 15 minutes. At the time, the 56-bit DES encryption algorithm was the strongest that the U.S. government allowed for export.
3. Bandwidth Clogging: The most visible problem with P2P file sharing programs concerns file sharing. This traffic clogs institution networks to the detriment of ordinary business related traffic. This affects response times for internal users as well as e-business customers. Many organizations have taken to using Internet links to create virtual private networks (VPNs) between their disparate offices or on-the-road users. If legitimate traffic has to compete with nonbusiness file sharing traffic, VPN performance will suffer.
4. Bugs: P2P file sharing applications require software to be installed on each participating leaf-node, exposing the network to a number of risks. A badly written application may cause the system to crash or conflict with business

applications. Security flaws in the P2P application may provide attackers with ways to crash computers or access confidential information.

5. Trojans, Viruses, Sabotage, Theft: An attacker could also convince a naïve user to download and install a booby-trapped P2P application that does damage or allows obtaining more information than they should have (backdoor access). A user of P2P software can misconfigure to expose confidential information for gain or for revenge. P2P file sharing applications can also cause a loss of control over what data is shared outside of the organization. (Note that these risks concern P2P applications that admit these problems to the network it rides on.)
6. Backdoor Access: Combining P2P applications with a VPN, security problems compound. If a user starts Gnutella and then clicks into the corporate VPN to check e-mail, a determined attacker could use this backdoor to gain access to the corporate LAN. Many companies delay rollouts of VPNs in spite of the potential cost savings.
7. Nonencrypted Instant Messaging (IM): IM leaf clients also pose an information leakage threat to the organization. All the messages sent back and forth in plain text across the network or Internet can be captured and read using a simple network-monitoring program.

Security Threats to P2P (The Large-Scale Issues)

- Interoperability: A central security issue-set in P2P networks is associated with the introduction of different platforms, different systems, different applications, working together in a given infrastructure. These differences open the set of security issues we associate with interoperability. The more differences we have to cope with, the more compounded are the security problems. Though wireless connectivity can lower interoperability problems, we as yet have no way of eliminating promiscuous listening.
- Private Business on a Public Network: Using P2P to conduct private business on a public network is clearly an exposure to security risks. The risks entailed by use of a public network must be addressed in order to avoid the liability this use entails.
- Variable Membership: With people “permanently” leaving the network, joining the network, there must be a feasible method to add or delete users without increasing vulnerability. The system is most vulnerable to users and former users who know the ins and outs of the system.
- General Security: P2P shares many security problems and solutions with networks and distributed systems.

Developers of P2P applications and security software need to meet on common ground to develop ways to use these products safely in a public environment.

Policy Modes of Security

a. Two General Modes that underpin organization policy:
   1. That which is not expressly prohibited is permitted.
   2. That which is not expressly permitted is prohibited.

These rules are dual to each other. The second rule is the most common rule-based policy put on firewalls. The first has more flexibility, but it admits greater security risk. The principles of Information Risk Management (IRM) can help define policy.

b. Information Risk Management (IRM): The processes of identifying, analyzing, and assessing, mitigating, or transferring risk is generally characterized as risk management. We want to secure information effectively from the following (Schneier, 2001):
   1. Threats
   2. Vulnerabilities: The sheer complexity of modern software and networks means that many vulnerabilities are inevitable. They are in every major software package.
   3. Breaches

These efforts are in a never-ending catch-up mode.

The following four questions are at the core of IRM:

1. What could happen (threat event)?
2. If it happened, how bad could it be (threat impact)?
3. How often could it happen (threat frequency)?
4. How certain are the answers to the first three questions (uncertainty)?

When these questions are answered, we can ask:

1. What can be done (risk mitigation)?
2. How much will it cost (annualized)?
3. Is it cost effective (cost/benefit analysis)?

c. Uncertainty Is the Central Issue of Risk. Does the Gain Warrant the Risk? These determinations of risk aid in defining and prioritizing security levels. These in turn help define policy and help the security professional develop layers of security (Schneier, 2000).

d. Routing Policy: Permits the use of different paths to minimize impact of attacks on data. New requirements have made the representation of routing policies more difficult (Security, 2003).
    1. There is a need to represent the routing policies of many organizations.
    2. CIDR (Classless Inter-Domain Routing, RFC 1519) and overlapping prefixes and the increasing complexity of routing policies and the needs of aggregation have introduced new requirements.
    3. There is a need to assure integrity of the data and delegate authority for the data representing specifically allocated resources to multiple persons or organizations.
    4. There is a need to assure integrity of the data and distribute the storage of data subsets to multiple repositories.

e. Firewalls: A firewall can block all traffic to and from a network. For outside communication, it is necessary to poke holes in the firewall. The firewall can direct incoming traffic to a DMZ, which in turn sorts out for delivery according to security policy. Peer-to-peer can penetrate firewalls (Hacking, 2003).

To date, with respect to enterprises, to protect the networks that P2P rides on, the controls that can be put in place are vigilant monitoring with clearly defined security policies.

Environmental Security

Many technology people who work on security issues tend to neglect security issues that lie outside the technology domain. In this sense, they treat security issues as if the users were functioning in a military installation that already has a physical security system in place.

When technology people focus on only the technology security without concern for the system environment, it reveals that security was not treated in a holistic way. This may deliver the unspoken and false message to users that the system is secure. Furthermore, it shows that the technology security was developed independently of the environment security and the gains that might come from a coordinated effort are lost (Schneier, 2000).

For mobile ad hoc networks, as utilized in Project AmIGo, these security issues take a new turn. They must include both the permanent position stations and the mobile stations. Thus, the security issues expand in their scope.

Readers interested in taxonomy precision for the distinction between pure and hybrid forms of P2P are referred to the Proceedings of P2P2001, especially the foreword. This and the Proceedings for P2P2002 have interesting applications that run on P2P overlay networks. Protocols and their routines for establishing trust define one of the more recent trends in P2P research and a number of articles on this aspect of security appear in the Proceedings for P2P2003.

Conclusion

These are some of the main issues in the analysis and design of a secure ad hoc mobile nomadic network. There is a justification for using P2P technology and a number of applications suggested for ad hoc mobile devices. The security threats have been outlined and in effect form a menu of further research to determine their specific role in the functions of these ad hoc networks. The focus here has been on threat identification. The next step is toward developing the security specifications in a design, such as required for the AmIGo Project. The AmIGo Project is an on-going collaboration between companies (Ericsson, Prevas, Nexus), research institutes (Santa Anna, SICS), and the Linköping municipality (Mjärdevi Business Incubator) with the goal to create a regional innovation system in the areas of peer-to-peer communication and ambient intelligence.

References

Baran, P. (1964, August). *On distributed communications*. Memorandum, RM-3420-PR. The Rand Corporation.

Barkai, D. (2001). *Peer-to-peer computing: Technologies for sharing and collaborating on the Net*. Hillsboro, OR: Intel Press.

Big LEO (2003). Retrieved 2003 from www.ee.surrey.ac.uk/Personal/L.Wood/constellations/tables/tables.html

Bluetooth (2003). Retrieved 2003 from www.bluetooth.com

Buchegger, S., & Le Boudec, J.-Y. (2002). Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc network. *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing*.

CSMA/AMPS Cellular (2003). Retrieved 2003 from www.computer.org/proceedings/hicss/1435/volume9/14350299abs.htm

Gollmann, D. (1999). *Computer Security*. Wiley & Sons.

Graham, R.L., & Shahmehri, N. (Eds.) (2001). *First international conference on peer-to-peer computing*. IEEE Computer Society Proceedings.

Graham, R.L. & Shahmehri, N. (Eds.) (2002). *Second international conference on peer-to-peer computing*. IEEE Computer Society Proceedings.

GSM-PCS (2003). Retrieved 2003 from www.cellular.co.za/gsm overviewpage.htm

Hacking (2003). Retrieved 2003 from www.interhack.net/pubs/fwfaq/

History (2003). Retrieved 2003 from www.isoc.org/internet/history/brief.shtml#JCRL62

Li, S. (2001). *JXTA: Peer-to-peer computing with Java*. Wrox Press.

Little LEO (2003). Retrieved 2003 from www.itu.int/newsarchive/press/WRC97/little-leo.html

Metricom (2003). Retrieved 2003 from www.metricom-corp.com

Moore, D., & Hebeler, J. (2002). *Peer-to-peer: Building secure, scalable, and manageable networks*. Osborne.

Oram, A. (2001). *Peer-to-peer: Harnessing the benefits of a disruptive technology*. Sebastopol, CA: O'Reilly.

Roberts, L.G. (2003). Retrieved 2003 from www.ziplink.net/~lroberts/InternetChronology.html

Schneier, B. (2000). *Secrets and lies: Digital security in a networked world*. Wiley & Sons.

Schneier, B. (2001, November 15). Crypto-gram (an e-mail newsletter).

Security (2003). Retrieved 2003 from www.fictitious.org/internet-drafts/rps-auth/rps-auth.pdf

Shahmehri, N., Graham R.L., & Caronni, G. (Eds.) (2003). *Third international conference on peer-to-peer computing*. IEEE Computer Society Proceedings.

SkyTel 2-Way (2003). Retrieved 2003 from www.skytel.com/ www.esselshyam.net/essel/wwwroot/SKYTEL.htm

TDMA-PCS (2003). Retrieved 2003 from www.iec.org/online/tutorials/pcs/ topic01.html

Toh, C.-K. (2002). *Ad hoc mobile wireless networks*. Prentice Hall.

Turcan, E., & Graham, R.L. (2001). Getting the most from accountability in peer-to-peer. *IEEE Proceedings First International Conference on Peer-to-Peer Computing*.

Wireless LANs (2003). Retrieved 2003 from www.computer.org/students/ looking/summer97/ieee802.htm

Chapter VI

Potential Security Issues in a Peer-to-Peer Network from a Database Perspective

Sridhar Asvathanarayanan
Quinnipiac University, USA

Abstract

Computing strategies have constantly undergone changes, from being completely centralized to client-servers and now to peer-to-peer networks. Databases on peer-to-peer networks offer significant advantages in terms of providing autonomy to data owners, to store and manage the data that they work with and, at the same time, allow access to others. The issue of database security becomes a lot more complicated and the vulnerabilities associated with databases are far more pronounced when considering databases on a peer-to-peer network. Issues associated with database security in a peer-to-peer environment could be due to file sharing, distributed denial of service, and so forth, and trust plays a vital role in ensuring security. The components of trust in terms of authentication, authorization, and encryption offer methods to ensure security.

Introduction

Over the last few years, the world has witnessed the explosion of Internet technology and the rapid growth in the use of peer-to-peer-based applications. The open nature of peer-to-peer networks and the decentralized administration and management of resources make it flexible for servents to operate in complete autonomy, thereby allowing them to freely participate or withdraw from the network without disclosing their true identity. While this can be considered as one of the salient features of a peer-to-peer network, the same can also be viewed as an inherent vulnerability built into these networks as they open up issues related to servent trust and security. The threat to database security, due to inherent vulnerabilities in the product and network, is further amplified when considering database implementations on a peer-to-peer network. While it is essential to discuss the security issues pertaining to peer-to-peer networks in general, it is equally vital to discuss the security issues pertaining to databases in a peer-to-peer network. This paper focuses specifically on database-related security issues in a peer-to-peer environment. The examples discussed are centered on Windows and UNIX environments, but the concepts can be applied to other environments as well.

There has been a growing trend towards using peer-to-peer networks for serious business purposes and for enterprise computing, and hence the need to analyze these security issues receives greater importance. The concept of enterprise peer-to-peer technology is evolving and, over time, it is predicted by observers that distributed data spread across peer-to-peer networks and stored on desktops and other computing devices in various locations where an enterprise operates are likely to replace centralized databases. There are also predictions on the rise of companies thinking in terms of corporate infrastructures that share the characteristics of peer-to-peer and client-server networks (Zeiger, 2001). It is not uncommon for companies that do not have a strong information security policy to have databases residing on departmental servers spread across the organization in various departments rather than being managed centrally by a data center. Most medium-sized companies find the decentralized approach more flexible as each department can be made responsible for its own operational applications, data, and security. The departments, too, enjoy this autonomy. The concept of enterprise peer-to-peer networks are built around this basic premise and the databases being stored and accessed in such peer-to-peer networks are subject to greater security concerns. Database systems, no matter how carefully designed and implemented, are constantly being exposed to security threats in various forms, such as denial of service and worm attacks. Peer-to-peer networks in general are prone to worm and denial of service attacks and with some of the existing vulnerabilities associated with databases, these networks, when they host corporate databases are likely to be an easy target for hackers. Due to the frequent discovery of new vulnerabilities associated with either the networks or with the security architecture of the database management systems, the whole activity of security administration and data management is turning to be a challenge rather than a routine function.

As the placement of data transitions from centralized servers to a distributed peer-to-peer network, the challenge of securing data is only likely to get even tougher. The administration and control of security was centralized when mainframe systems were used to store and manage all corporate data for an enterprise. All data needs of the organization were satisfied by the centrally stored database and there was neither the need to connect outside the corporate network to access data nor the necessity for interaction between host nodes in a distributed network to exchange data. With the emergence of client-server computing, the approach went a little more decentralized allowing users to connect to any data sources available on the corporate network (Zeiger, 2001). As the business processes got more demanding and the need to access and process data outside of the company started becoming more critical to the success of the enterprise, network access started to reach beyond the organization's domain, thus exposing the corporate network to higher risks of security. With the slow but certain evolution of enterprise peer-to-peer technology, organizations have started looking more outward for their data needs and eventually there is a compelling need for security administrators to better understand the associated issues with data security.

This paper discusses some of the data security issues associated with enterprise peer-to-peer networks and some possible measures that can be taken to help reduce them. Distributed peer-to-peer systems are being discussed to give a flavor to distributed data management. The following sections analyze the data security issues involved in distributed networks, and some of the inherent vulnerabilities in peer-to-peer networks that have a bearing on data security will be discussed. Database security issues will be analyzed in general and from the perspective of peer-to-peer networks along with some of the recommended methods in general to provide data security.

Threats to Data Security in a Peer-to-Peer Network

Threats to data security in a peer-to-peer network can be categorized as (a) threats that originate from the inherent vulnerabilities associated with files and databases hosted in centralized systems but amplified in a peer-to-peer environment, and (b) threats that are specific to peer-to-peer networks. Databases hosted on a peer-to-peer network are constantly exposed to threats such as being stolen, introducing high packet traffic on the network, denial of service attacks, being infected by worms, gaining unauthorized backdoor access, and so forth. Some of the vulnerabilities associated with peer-to-peer networks that have a significant impact on database security are discussed.

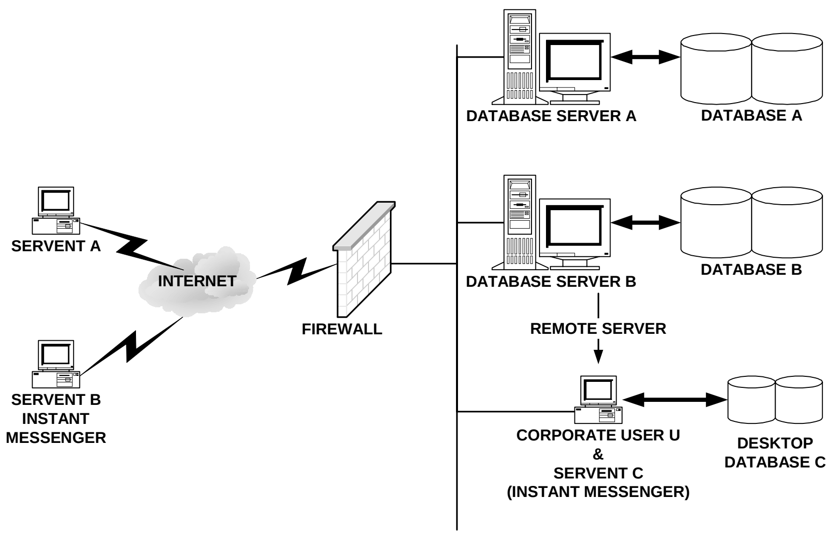
The Effect of File Sharing on Database Security

The majority of relational database management systems store data ultimately in files that can be tampered with when a hacker breaks into the server that hosts the database. Most peer-to-peer applications such as Kazaa and Gnutella give direct access to files that are stored on a node's hard disk. With little effort, it is possible to identify the operating system that runs on the computer and use techniques to break into the shared directories and other resources. There are utilities to hack into the system directories and read the files that store authentication information and decipher passwords. One example is the *l0phtcrack* utility that can run multiple passes on the .SAM file (security file containing ids and passwords) of Windows 2000 and decipher passwords. lophtcrack is a password auditing tool. It can compute Windows NT/2000 user passwords from the cryptographic hashes that are stored by the NT operating system. Using such utility, if the administrator's password is obtained, then the host has been compromised. The impact that this alone can have on database security is enormous: the database service can be stopped or disabled—in effect, taking the database offline and enabling it to be deleted or copied—invalid data could also be introduced into the database causing the database to lose its integrity.

A participant in a peer-to-peer network could be a valid user in a corporate network but the peer-to-peer application that runs on the desktop of this participant could allow other servents from outside the corporate network to possibly connect to this servent. Corporate firewalls are usually set up to check for such intrusions, but in most companies, applications that allow real-time chatting, such as instant messengers, are not blocked from usage. The latest versions of these programs come with functionalities that include file sharing and voice messaging. All risks associated with file sharing are applicable to the use of such instant messengers as well. One of the greatest risks associated with databases in a peer-to-peer network has to do with file sharing.

As shown in Figure 1, Servent B and Servent C are engaged in a conversation. Servent C is also part of a corporate network and a desktop version of database server runs on its computer. Database server B is being accessed by Servent C through a remote database server setup. Since Servent C is a participant in a

Figure 1. Peer-to-peer network servent in a corporate network



peer-to-peer network, there is a high probability of having open shares defined on this node. A few possible ways that the security of the corporate databases A, B, and C can be compromised in this setup are as follows:

a) Servent B can access open shares of folders in Servent C, access the authentication file, run the decryption utility to decipher passwords, and obtain administrative access to Servent C's host computer. Servent B can even install administrative tools on Servent C's computer and take complete control of database C. This would mean unauthorized activities such as deleting database files, stopping the database engine, copying the database file, and so forth.

b) Through the remote database server connection, Servent B can access database B from database server C, view the contents, and depending on the rights that have been assigned to User U, can even update or delete records stored in database B.

c) If User U is part of the domain administrators group in the network, Servent C can log in as User U into database C and run operating system commands within the database server. Most database servers offer operating system command shells that let users execute operating system commands. Breaking into a database server has the potential to cause damage not only to the database server but to the entire network.

d) Servent B can create new user IDs in databases A, B, and C with administrative rights and allow remote access to other servents such as Servent A.

e) Utilities such as *tcpdump* can be installed and run by intruders and tcp packets can be analyzed to obtain passwords and other critical data stored in corporate databases.

f) Files that store server/database alias information such as sql.ini, interfaces, tnsnames.ora, and so forth. could be altered, resulting in client nodes losing access to the databases.

Unlike most of the enterprise databases such as Oracle, DB2, or SQL Server, another class of databases such as Microsoft Access or Paradox are more vulnerable to theft simply because of the fact that there are no services that manage the database files that constitute these Access database files. Since the operating system does not maintain a lock on these files, it is quite straightforward to copy or manipulate Access files using peer-to-peer applications such as *BadBlue*, which has the capability to transcode live Access databases over the Web (BadBlue, 2001). While it can be considered as an outstanding feature allowing users to obtain access to real-time data, the security hole that it creates cannot be overlooked. A peer-to-peer participant that has access to shares in another node can install programs on that node and transform it to a search-enabled Web server. Once installed and configured, it lets users view shared database data, spreadsheets, and documents live.

Peer-to-peer networking works around corporate database security by providing decentralized security administration and shared file storage, providing opportunities to circumvent critical perimeter-level defenses such as firewalls and NAT devices. The ability of users to install and configure peer-to-peer clients surpasses all the server-based security policies implemented by network and database administrators.

Extracting Sensitive Data Through ODBC Messages

A sizable number of applications hosted on peer-to-peer networks use .asp pages served by IIS along with backend databases such as SQL server or Oracle (Trivedi, 2003). These applications connect to the database using ODBC. Using vulnerabilities exposed by weaknesses in this technology, it is possible to derive very valuable information pertaining to database security directly from the error messages returned by ODBC drivers.

One such method takes advantage of .asp Web pages along with the "get" method or the "post" method while passing values to the next page and so on. The difference between the two is that the “get” method shows up the values being passed on the URL while the “post” method does not.

A series of queries typed in the URL as shown below can display different error messages revealing additional information about the underlying database.

Often, is possible to see something on the URL that resembles the following example:

http://www.p2psecurity.com/openpage.asp?cid=36

This means that a value of 36 is being passed to a hidden datatype called *cid*. In order to hack into the database, there is no need to spend efforts to identify what “cid” refers to.

Using the information returned from the previous query, the hacker can now make use of the UNION operator to return additional information as shown in this query. So the modified URL would look as follows:

http://www.p2psecurity.com/openpage.asp?cid=36 union select top 1 table_name from information_schema.tables

The result of this query is the first table name from

information_schema.tables

table. But the result obtained is a table name, which is a string (nvarchar) uniting it with 36 (integer) by UNION. So the following error message is displayed:

Microsoft OLE DB Provider for ODBC Drivers error ‘80040e07’ [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value ‘logintable’ to a column of data type int. /openpage.asp, line

The error message indicates that the first table is “logintable” and it is likely to have login information with passwords. The attacker can go to the next step and obtain column information from that table by running a query such as *http://www.p2psecurity.com/openpage.asp?id=36 union select top 1 column_name from information_schema.columns where table_ name='logintable'*

Extending the same technique, it is possible to identify all sensitive columns from the table called *logintable* and also run queries directly by submitting them as part of the URL and even obtain password information.

The above is a serious security concern for databases hosted on peer-to-peer networks. This was simply an example of how URLs can be altered. By combining intuition and imagination, more serious hacks are possible, resulting in greater damages.

Denial-of-Service Attacks

One of the popular methods of denial-of-service (DoS) attacks on a database server in a peer-to-peer network is by creating a "UDP packet storm" on the server. The effect is an attack on one host causes that host to perform poorly. An attack between two hosts can cause extreme network congestion in addition to adversely affecting host performance. When a connection is established between two UDP services, each of which produces output, DoS can be achieved by ensuring that these two services produce a very high number of packets. This causes packet flooding leading to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. When this congestion is created against two target servers hosting databases on a peer-to-peer network, it creates an infinite nonstopping loop between the two systems, making them useless for any real data exchange from their database servers.

Distributed denial-of-service (DDoS) attack is another form of threat to database security in a peer-to-peer network. These attacks involve breaking into thousands of computers in a peer-to-peer network by following a few coordinated steps, taking control over those computers by installing "root-kit" programs that take over system administrator privileges to keep other administrative users from being able to find the intruder. DDoS attacks are primarily targeted at shared computing resources such as database servers, Web servers, network bandwidth, router processing capability, and so forth.

The compromised database servers in a peer-to-peer network can be controlled by the intruder by sending commands to the installed Trojan program on the servers to carry out the desired damage on all the servers. The vulnerability of the database instance in a database server is largely determined by the authentication schemes used to originally gain access to the database. Once a database server in a peer-to-peer network is hacked, the hacking program first looks for the authentication scheme used by the database engine. The operating system has already been compromised through an administrative login and if a database server uses a domain-based authentication, then the same administra-

Figure 2a. Corporate network behaving like a P2P network

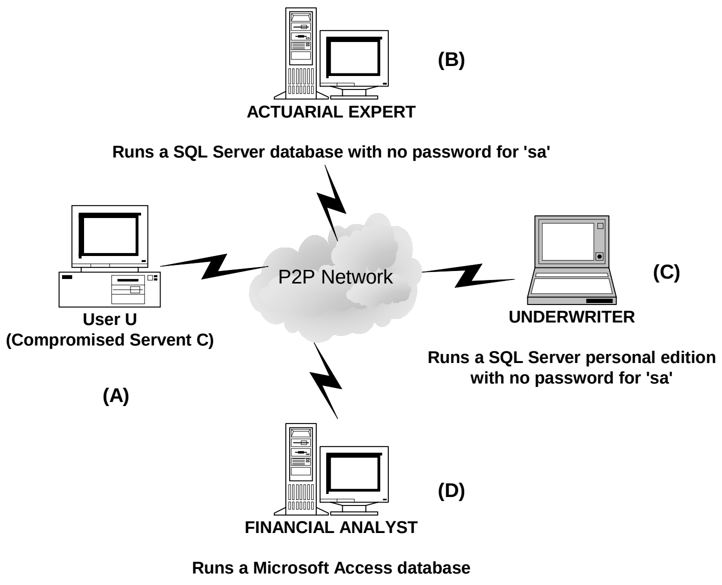
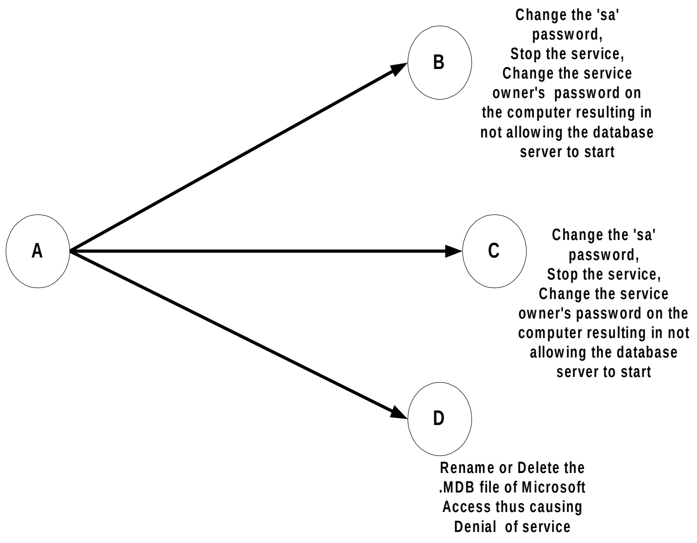

(B)
ACTUARIAL EXPERT
Runs a SQL Server database with no password for 'sa'
P2P Network
User U
(Compromised Servent C)
(A)
(C)
UNDERWRITER
Runs a SQL Server personal edition with no password for 'sa'
(D)
FINANCIAL ANALYST
Runs a Microsoft Access database

Figure 2b. Attacks on the database servers


B
Change the 'sa' password, Stop the service, Change the service owner's password on the computer resulting in not allowing the database server to start
A
C
Change the 'sa' password, Stop the service, Change the service owner's password on the computer resulting in not allowing the database server to start
D
Rename or Delete the .MDB file of Microsoft Access thus causing Denial of service

tive ID can provide access to the database server as well. Having gained access to the database server, it is possible to propagate the attack to other database servers (using domain authentication) in the peer-to-peer network.

Corporate-wide denial-of-service attacks on database servers can also be initiated through an innocent peer-to-peer servent within the corporate network as shown in Figure 1. If Servent C (User U) is compromised, then the attacking programs installed on that computer can be controlled by the attacker to perform port scans to look for specific designated ports on which database servers run by default (1433 for SQL Server, 1521 for Oracle, as examples in a Windows environment). In most corporate networks, a peer-to-peer setup does exist as most power users have their own copy of the database server running on their desktops and executives share data between themselves.

As an example, let us consider the workforce of an insurance company—highly diversified employees performing a variety of vital roles such as that of underwriter, actuary, financial analyst, and claims adjusters. Each has specific objectives, requirements and software applications running on his/her desktops that are tailored to his/her specific job. While these professionals are domain experts in their own fields, they are not database administrators or security experts. The majority of these applications use some form of a database as their backend are generally centrally controlled, and with the availability of desktop versions of databases, it is very simple to obtain a copy of a database server, install it and have it running in a few hours. Microsoft SQL Server allows the installation of a server with no password for the system administrator (SA) account. It warns against leaving the password blank but ultimately does allow it.

As shown in Figure 2a, if there are database servers in the network that store critical statistics, pricing, and other financial data along with a nonprotected SA account, the compromised computer (Servent C) would find these as its first victims to attack.

A method that is adopted to cause denial-of-service attacks on database servers in a peer-to-peer network is by exploiting the existing vulnerability in buffer overruns in SQL Server and causing it to execute arbitrary code (Hacking MSSQL, 2003). Worms are also built around this vulnerability. Programs can be written to use sockets to, and with the help of port scanning utility such as *NetCat* and file transfer utility such as *TFTP*, it is possible to learn which users, shares, services, processes, and files are associated with the target computer and plan the attack from there.

Figure 2b shows how a database server on a peer-to-peer environment gets attacked and how it propagates the attack. Computer A can attack B and C on the network by opening up listening sessions with B and C and by running a custom-built program to open up a remote shell with B and C. Tools such as

DumpWin can be uploaded from the attacking computer onto the attacked computer through *TFTP*. Using *DumpWin* on the target computer, critical information on users, shares, and so forth, can be dumped to a file and TFTP'd back to the attacking computer. These would give the basic information to carry out the denial-of-service attack on the database server. With respect to computer D, the simplest way to carry out a DoS attack is to rename or delete the Access database.

Role of Trust in Ensuring Data Security in a Peer-to-Peer Network

One of the significant advantages that peer-to-peer networks offer over centralized networks is the ability that it provides its participants to access the multitude of databases and files stored on user laptops, desktops, servers, handheld devices, and so forth, in ways that would be too difficult and cumbersome to implement with legacy systems. When discussing data/database security in a peer-to-peer environment, it is worth considering a security model that encompasses the following components: identification, authorization, access control, confidentiality, nonrepudiation, and availability (Georgiev & Georgiev, 2001). Trust is certainly an issue in every critical peer-to-peer application. In this context, trust can be viewed from two points of view—the level of trust that determines the confidence level of each of the participants over others when they indulge in accessing each other's data, and the degree of trustworthiness of each component in a peer-to-peer network. The latter plays an important role in determining the overall security. The built-in security features of the network components, database engine, and infrastructure strengthen this model. Where there is absence of built-in features, it becomes necessary to develop methods from bottom-up to ensure security to data in a peer-to-peer network. The discussion on trust here is an extension to the model discussed by Georgiev and Georgiev (2001), and it analyzes the impact on data security in a peer-to-peer network through the evaluation of the trustworthiness of the partners, communication media, intermediate systems, clients, client identity, server identity, and administration. One important aspect of trust with respect to database security in a peer-to-peer network is that while it is critical to ensure peer authentication, it is also important to ensure management of the database.

Untrusted Components Involved in the Network

Database security can be compromised in a peer-to-peer network due to transmission of worms or viruses from infected desktops that participate in the network. Personal desktops, laptops, and workstations are inherently untrustworthy due to the fact that they can be connected to any network and they can be running hidden utilities that might access the network to download harmful viruses and worms. In general, data security is highly compromised due to untrusted communication media. Wiretapping, which is used to eavesdrop on communication media such as twisted pair, coaxial cables, and satellite links, could result in theft of valuable data as they are transmitted. Data from a servent to a node in the peer-to-peer network could pass through numerous gateways and routers; making multiple hops, there is no control over the trustworthiness of the intermediate nodes. Packets can be opened and sensitive data can get stolen on their way. A request for authentication that carries important user/password information might have to cross multiple domains and the trustworthiness of the intermediate components and communication media play a vital role in ensuring security of such information.

Establishing Trust to Secure Data in a Peer-to-Peer Environment

The elements involved in establishing trust and securing data in a peer-to-peer computing environment are the three basic ones that are applicable in a centralized environment (Sunsted, n.d.). They are authentication, authorization, and encryption.

a) Authentication: In general terms, it is the process of evaluating the credentials of an entity to establish the fact that the entity is the one that declares itself to be. In the context of databases in a peer-to-peer network, this is a two-pronged process—peers authenticating themselves to other peers over the network, and the users of a database server hosted in the peer-to-peer network authenticating themselves to the database server. Authenticating should be carried out with great care—when using HTTP-based authentication, the username and password are transmitted unencrypted across the Web, which defeats the purpose of authentication. SSL (Secure Socket Layer) is recommended as it encrypts the username and password before transmission. Like passwords, certificates or tokens can be used for authentication. It is a good practice to set up authentication based on network identity—based on IP address or subnet. Wherever a domain-based trusted authentication is possible, it is recommended to use such authentication.

b) Authorization: The process of assigning an authenticated entity in a peer-to-peer network the permission to do certain actions or perform certain tasks on the database server. In short, authorization is the set of rules that govern what resources a user can access on the database server and what the users can do with those resources. It is recommended to create views on the database tables and provide read-only access only to selected views with selected columns. This is a way to protect data in tables from unauthorized manipulators.

c) Encryption: In a peer-to-peer network, encryption plays a vital role. To ensure that data exchanged between two peers are not tapped during communication, encryption can be used to protect the information. Encryption ensures protection of data even in an insecure peer-to-peer network. This can be very effective in preventing data thefts. Encryption ensures that data is not in a readable form and hence an unauthorized peer will find no advantage in possessing the data unless the person obtains the key to decrypt it. Further, by encrypting the data, even the DBMS is prevented from knowing the value of the data and hence malicious Trojan horse codes that traverse through the peer-to-peer network cannot pass spurious values to corrupt the data. Another problem that encryption can address is that an attacker who does not know to encrypt the data will end up entering records that will not appear legitimate when an authorized user decrypts it (Davida, Wells, & Kam, 1981). The level of security that can be attained through encryption is determined by the robustness of the cryptographic algorithm being used. The techniques that can work well on a peer-to-peer network to encrypt databases will be based on either symmetric/secret key or asymmetric/public key cryptography or a combination of the two.

Conclusion

The data security issues discussed earlier in this paper can be addressed to a large extent by enforcing strong passwords, access control through network devices, and proper and timely application of patches supplied by vendors to combat the vulnerabilities. However, the underlying foundation on which peer-to-peer networks are built is, trust—trust in the users that interact with, trust in the network components, trust in the hardware, and trust in the database software, operating systems, and patches.

An inhibiting factor for the growth of peer-to-peer networks is the issue of data security. With the proper deployment of applications, network infrastructure, and a strong security policy in place, it is possible to effectively utilize the power of peer-to-peer technology and its outward approach to bring data shared across the enterprise. Enterprise-wide data design will have to be reshaped to take advantage of the peer-to-peer networks (Zeiger, 2001). With these networks, not only desktops but also PDAs, cell phones, and other computing devices can be linked. With the growing popularity of peer-to-peer collaboration platforms, the rules of data distribution and data security have to undergo changes.

References

BadBlue adds P2P database sharing. (2001, July). Retrieved from http://www.infoanarchy.org

Davida , G.I., Wells, D.L., & Kam, J.B. (1981, June). A database encryption system with subkeys. *ACM Transactions on Database Systems.*

Georgiev, I.K., & Georgiev, I.I. (2001). A security model for distributed computing. *Consortium for Computing in Small Colleges.*

Hacking MSSQL and Beyond. (2003, March). Retrieved from http://www.ebcvg.com

Sundsted, T. (2001, July). The practice of peer-to-peer computing: Trust and security in peer-to-peer networks.

Trivedi, C. (2003, April). Hacking database server P1. Retrieved from http://www.ebcvg.com

Zeiger, A. (2001, July). The impact of peer-to-peer on data management issues for developers. Retrieved from PeerToPeerCentral.com

Chapter VII

Security and Trust in P2P Systems

Michael Bursell, Cryptomathic, UK

Abstract

This chapter examines the issue of security in peer-to-peer (P2P) systems from the standpoint of trust. It takes the view that P2P systems present particular challenges in terms of trust over other socio-technical systems, and identifies three key areas of importance: identity; social contexts; punishment and deterrence. It suggests that a better understanding of these areas and the trade-offs associated with them can help in the design, implementation, and running of P2P systems. The chapter combines a discussion of problems and issues in current systems with a review of some of the wider sociological and nonsystems literature which can aid those involved with P2P systems. It concludes with some suggestions for areas where future research may provide fruitful insights.

Trust and Security

"I would trust my brother or my sister with my life, but I wouldn't trust either of them to back-up my hard drive."

Peer-to-peer (P2P) systems require different entities to decide how to interact with others—or whether to interact with them at all: these are security decisions. The system itself will probably be set up to allow particular types of interaction, or to allow particular choices about interaction: these, too, are security decisions. They are in fact decisions about trust. Within many P2P systems, I need to know whether I can "trust" another entity within that system, Alice, and what to do with a statement from yet another entity, Bob, saying that *I* can trust this Alice because *he* does. "Trust" is a word that is used very loosely in English, but a concept that should exercise the thoughts of anyone thinking about security in a computer system, particularly when that system is distributed, and even more so when it is a P2P system. This chapter addresses how trust and security are linked in certain types of P2P systems and provides some ways of understanding how assumptions about trust can help or hinder the security of a system. It takes a sociological, rather than an overly technical view, with the hope that from such a perspective, designers, builders, and users of P2P systems may have a better chance of achieving the kinds and levels of security they need.

Why Does Trust Matter?

One approach to dealing with how to build trust into a system is that discussed by Waldman and Rubin (2001): "We are concerned less about conceptions of trustworthiness than we are about designing systems that rely as little as possible on trust. Ultimately, we would like to design systems that do not require anyone to trust any aspect of the system ... In this context the ideal trusted system is one that everyone has confidence in because they do not have to trust it. Where that is impossible we use techniques such as reputation building and risk reduction to build trust" (pp. 243–244). For them, trust is an issue to be avoided if possible, and built in if absolutely required. This chapter argues that trust is a key component of P2P systems, whether implicit or explicit, and that understanding the requirements, costs, and trade-offs around it is vital to such systems' design, implementation, and running.

It is worth stating at the outset that this chapter attempts to examine a broad superset of P2P systems and that for some subsets some or many of the lessons addressed may not be relevant. The "perfect" P2P system could be defined as a system with a perfectly flat hierarchy, full communication between all entities, and with each entity allowing other entities to use local resources (storage or processes) according to some agreement or contract, whilst simultaneously using remote resources provided by other entities. While this kind of system may never exist, the aim of this chapter is to look at the kinds of security questions that arise in such a world: other systems can hopefully be examined as subsets or refinements of such a "perfect" P2P system.

One of the great difficulties in thinking about security for P2P systems is that the model of security that must be espoused is different from that of most other types of system. Most other systems have a centralised entity "providing security," though that centralised provision may well be delegated. Without a centralised entity "providing security," requirements and assumptions about security in the system need to be built not only into the fabric of the system (its infrastructure) but also into the models of interaction between entities—this is one of the challenges facing us in the P2P world. In most other systems, there is an assumption that attackers, in the sense of entities who perform "intentional and unwarranted actions" (Schneier, 2003), are most likely to come from without, or at least that internal attackers can be profiled and controlled. In many P2P systems the danger is reversed in that attackers can come from within the system just as easily as from without. The challenge of mitigating the actions of attackers must be tackled at the design stage of system, as it if for the system itself, rather than a godlike administrator (or her minions), to deal with such attacks. If a new way of interacting between entities in the system arises that has not previously been considered, there can be no magic fix that can be applied. A good example of this is the story (later discredited) that the Recording Industry Association of America (RIAA) was planning to "infect MP3 files in order to audit and eventually disable file swapping" (Orlowski, 2003). The P2P systems that were distributing MP3s had no model to deal with malicious insiders distributing malware content: the (implicit) assumption was that all content could be "trusted."

What is Trust?

So what do we mean by "trust"? When thinking about security in a system, various entities need to "trust" others to varying degrees. We can say that we have *trusted hardware*, or that we *trust* administrators to exercise particular *roles* in a system, but without a clearer definition of trust, it is difficult to know what these statements mean. What is meant by "trust" in different contexts is a semantic minefield, and definitions of the term abound, but Sztompka (1999) has a useful general definition: "Trust is a bet about the future contingent actions of others" (p. 25). Another way to think about this is that trust is a way of dealing with risk in the real world. This is not, however, a full enough definition, and trust is taken, for the purposes of this chapter, to mean that *one entity has a particular level of expectation that another entity will perform (or not perform) a particular action in a particular context—typically the consumption or provision of resources of some type*.

Although this is far from a full definition, it is already helpful because it immediately allows us to look at trust within the sphere of security. Capabilities-based security involves the granting of particular rights to particular entities: entity A might base the decision on whether to grant entity B a particular capability dependent on whether A trusts B to use that capability in a way that is consistent with A's goals for the functioning of the system. Trust is a concept that we are used to dealing with if we think about it in these terms. Already, however, we are moving into views of systems that may make P2P adherents uneasy and have an ethical dimension: in a nonhierarchical P2P system, what "right" has A to restrict B's capabilities based on A's goals for how the system should function? The issue of authority is a key one, and we will return to it later in this discussion.

Dealing with Failure

Making decisions about how much risk to accept in a system—and how to deal with it—is not a new one, of course. Within the discipline of computer science and electronic engineering, a great deal of thought has gone into this area in order to meet the requirements of fault-tolerant systems (Fischer, 1987). One area in which the study of fault-tolerant systems can help in the design of P2P systems is in the understanding of different types of failure. Within fault-tolerant systems, there are two different types of faults that are of interest to us: failstop faults and Byzantine faults. In many ways these two faults represent extremes of a spectrum: a failstop fault occurs when a faulty process ceases operation and other processes are notified of the fault—a kind of failure rare in real-world systems, where processes tend to fail and other processes are left to pick up the pieces as best they can. At the other end of the scale is a Byzantine fault, where a faulty process may continue to operate but can send arbitrary messages.

The Byzantine fault condition is named after a model of a distributed system where "generals" are "bribed" to disrupt the operation of a system—a situation similar to some types of attack from internal entities in a P2P system, where a number of entities might collaborate to provide false information. Lamport, Shostack, and Peace (1982) and Gray (1987) discuss how to solve this issue and the cost of deciding on the truthfulness of a particular communication.

At first glance, it might appear that the resolution of this problem offers us a solution to the question of how to trust entities, but unluckily, the world of P2P systems is significantly more complicated. These are solutions aimed at systems involving fairly simple interactions. Many communications and interactions between entities in a P2P system are indeed more complicated, but even so, might it not be that the type of communication with which we are interested could be modeled quite closely on the Byzantine generals problem? If we have enough other entities in the system saying “C is trustworthy,” can we not make a decision on whether to trust C based on this information, using the lessons from fault-tolerant computing? The short answer is no, and largely for the reason that these systems are not looking at the *future* actions of others, and because they do not deal with the social aspects of trust. This issue is the subject of the next section.

Trust as a Social Phenomenon

Although we may need to design a P2P system with resilience to deal with failure, it is the social problems of trust that are relevant within this discussion. This means that the longer answer is more interesting than the short no. The rest of this chapter will attempt to explain why this is the case, and some possible ways to improve the situation. Although the world of distributed systems has provided many different technologies to help secure systems (see Anderson [2001] for a discussion of many of these) one of the main distinguishing features of P2P systems is that they occur within social contexts. More accurately, many P2P systems involve interactions that model social interactions, in that they can occur for different reasons between entities that have varying levels of knowledge of each other and each other’s history. They also have varying degrees of ability to enforce the expected behaviour of the other entities. The following sections will examine these three areas: identity, social contexts, and punishment and deterrence.

It should come as little surprise that security is a social issue. Recent work on security (Anderson, 2001; Schneier, 2000; Schneier, 2003) has stressed the importance of understanding the social contexts of those interacting with computer systems. In many systems, modeling of the actors who may be involved with the system from a security perspective, or of the interactions involved, may be a fairly tractable problem. The discipline of vulnerability analysis and risk assessment employs a variety of relevant methodologies (Pfleger, 1997). For many P2P systems, however, where one of the *reasons* for the existence of the system may be the opportunities for unbounded interactions, the problem is much less tractable. This does not mean that security is impossible within P2P systems—just that it may need to be approached from a different direction.

Identity

Identity, like the issue of rights over others, is a question with an ethical dimension for P2P systems, for the simple reason that many such systems have grown from areas of study or interaction where it is either undesirable to know the identity of other entities or where such knowledge is unnecessary. Examples of the former are systems where plausible deniability is desirable (e.g., Dingledine, Freedman, & Molnar, 2000) or anonymity is paramount (networks of mixmaster remailers, for example: Cottrel, 1996). In such cases, systems are designed to protect privacy—or, more accurately, to protect and disguise the identity of entities on the system. Many file-sharing P2P systems (e.g., the emule project, n.d.) operate with "loose" identity: entities have an identity to provide the opportunity for contact, and not for any other reason. There are even occasions (Orlowski, 2003) when plausible deniability, though not necessarily a design feature, can be a useful by-product of such systems. Although there are indubitably ethical questions around issues of identity and privacy—and the P2P community has taken a strong stance on many issues of civil liberties—there are times when it is very important to be able to identify entities in a system with a high degree of certainty. There are many situations where P2P systems offer great opportunities for collaboration and it is in areas where privacy is important that much of the initial development of these systems has taken place. There are, however, situations in which varying levels of certainty about identity are important: in scientific research, policing, or national security, for instance.

For the purposes of this discussion of a broader set of P2P systems, however, and within the context of "trust," identity is an important part of the security story: privacy-protecting systems are better seen as examples where the need for strong identity has been examined and removed—and, in some cases, steps taken to protect and disguise identity. The reason that identity is important is simple: if you wish to trust entity A, you need to be able to identify it. It may be that you are taking a recommendation from another entity, B, or that you are basing your decision on previous behaviour (more about both of these cases below) but if you cannot identify A, you cannot make a decision about whether to trust it or not.

This is the first instance in the discussion so far that the question of trade-offs comes in. As discussed at length by Schneier (2003), security always involves trade-offs. The trade-off involved in this case is how much time and effort you are willing to take to be certain of A's identity against the damage to you if you get it wrong. There are steps you can take to mitigate against the damage, and deterrence and punishment play their part (see below) but without making a decision on this trade-off, you cannot move forward on a decision to trust an entity or not.

Identification

Public keys and digital cryptography have offered some great opportunities to allow communications with other entities to be identified as coming from that entity, and that entity alone, and this goes some way toward helping us to provide "strong" identification. Being able to sign messages cryptographically means that we can add a time dimension to identity: given an initial identification, and assuming that we are willing to accept the risk that the signing key may be compromised, we can continue to associate communications with the entity we identified at the start of the process. There are, however two major assumptions here: that we are willing to accept the risk of compromise and that we are able to make an initial identification. Making an initial identification was easy (Dunbar, 1996) in pre-urban societies where everybody knew everybody else, or where degrees of separation were almost always very low. The growth of banking systems was the response in many Western societies to the need for trusted third parties, and the rise of the corporation, with a hierarchy of authority, also provided a framework for identifying individuals.

Identification services for public key systems tend to be provided by Public Key Infrastructures (PKIs). PKIs have proved very successful within a restricted range of situations, and are often pointed to as the "great hope" of distributed systems, but there are two main problems with PKIs within the P2P world: first, they have to be run by somebody, and second, they work best within a hierarchy. This is because a PKI is about a single, trusted entity—or, sometimes, a small set of trusted entities—signing the keys of other entities, and a chain of trust flowing down from that premise. It is within hierarchical systems, such as key signing for banks, that PKIs have had their greatest success, but the archetypal P2P system is, by definition, nonhierarchical. Even worse than the problem of having a hierarchy within a P2P system is the question of how to proceed if you decide *not* to have such a hierarchy. In order for a PKI to work in this sort of situation, you need to have a trusted third party. It may be, in a restricted subset of P2P systems, that this situation may be acceptable to the designer of the system, the members, and all future members, but this is, indeed, a restricted subset.

Recommendation Systems

Other identification systems exist, of course, and are more or less appropriate for P2P systems. It might be more accurate to refer to some of them as recommendation systems—the more successful ones might be characterised as those that recommend an entity as "fit for purpose": this is a phrase to which we will return in the section on social contexts. A simple type of recommendation system might be a brokering service, where entities as "fit" for a particular purpose, such as offering hosted services (e.g., Scoot, 2003). This is a very simple case and, unless there is some checking of the claim, such systems can offer us little of interest in this discussion.

A classic example of a tried and trusted recommendation system is the PGP system (Yttebord, 2001), which has much to recommend. Although designed and rolled out before the broad advent of P2P systems, it is intended to work in systems that are peer-to-peer: where no single authority for identification exists, and you have to trust friends and colleagues to provide trust levels for those with whom you interact, allowing the creation of "chains of trust." Such chains of trust are examples of "transitive trust." Schneier (2003) states that "[t]rust should ... be granted directly. If you trust Alice, and Alice trusts Bob, that does not automatically mean that you trust Bob. You might not even know Bob. In systems where you automatically trust Bob, trust is said to be *transitive*. And when trust must be transitive, it creates brittleness" (p. 141). Unfortunately, in most P2P systems, transitive trust is a key component, even if it is qualified—that is, when entities decide to trust others to a certain extent. Without transitive trust, systems are restricted only to a small group of acquaintances, each of whom knows all of the other members. There are two basic approaches to extending the possible size of systems: to allow chains of trust to stretch from one person to another, such as the PGP system, and to have a central repository of transitive trust "reputation" built into the system.

Systems that take the latter approach can be characterised as using "reputation servers," and can be seen as an improvement on the simple "brokering" model. Some good examples of these types of services are news rating, online gaming, and auction services. News rating services such as Slashdot (CmdrTaco, 1999) and kuro5hin (Ainsworth, 2003) allow members to comment on news stories, and for other members to rate the quality of those comments, some with sophisticated checks and balances such as Slashdot's meta-moderation service. In online gaming (e.g., Everquest FAQ, 2003), gamers will typically have a "player," and as they progress through the game, they gain status through various tasks and interactions. Though only loosely a "recommendation system," such online gaming systems associate entities with actions over time, and other entities interacting with them may learn to "trust" them to behave in particular ways when they are in particular situations.

Reputation services around auction services, particularly eBay (eBay, 2003), have attracted a significant amount of academic attention. One of the great attractions of eBay for academics is the amount of data that has been made available for study, and the speed at which new data is added about the operation of what is an extremely popular service. Game theory—best known for its application in "The Prisoner's Dilemma"—is a useful tool to apply to many areas of conflict (for an excellent critical introduction to this field, see Heap and Varoufakis [1995]). Friedman and Resnick (1999) apply game theory to the perennially difficult problem of how to maintain an effective reputation system in an environment where pseudonyms are cheap: in other words, where there is little or no strong identity. They suggest three ways of dealing with this issue: distrust newcomers, increase the cost of name changes, or have an external third party who is trusted to provide "one-off" name changes over a particular "game period." All of these suggestions require trade-offs. The first suggestion requires a change to the interaction rules of the system, and might even be codifiable in certain systems, but reduces the overall possible efficiency of the system. The second could also be codifiable in certain situations, and it may be that the trade-off of cost—or other resources—might be an eminently acceptable one. The third raises again the issue of how to involve external entities and when this might be acceptable. In fact, we have come back full circle—one of the key criticisms of reputation servers is that they rely on a centralised server, even if it is "part of the system," and there are many P2P systems in which reliance on such a centralised server is inappropriate. Such a server-based system certainly does not meet the criteria required for a "perfect" P2P system. Distributing the server's functions and providing failover servers also provoke all the problems with reliable naming services that distributed systems research has been examining over the past few decades (e.g., Stewart [2003] on DNS cache poisoning). Even such a distribution, unless it is to all members of the system, may be felt inappropriate for some P2P systems.

Multiple Identities

Another "cost" of cheap pseudonyms that is mentioned by Friedman and Resnick (1999) is that it is easy for entities in the system to have multiple "identities." There are times when there is little cost associated with trust when entities may have multiple identities—it may be entirely acceptable for a user to have multiple personae in an online gaming context, for instance, but there are, equally, situations in which the use of multiple personalities can be deeply undermining to a system. An example is where majority voting is implemented within a system: if it is cheap for entities to construct new pseudonyms within the system, there is nothing to stop a small number of entities—or even a single one—from influencing the voting process. Reputation systems are themselves very susceptible to these types of attacks: in order to boost your reputation, why not create multiple pseudonyms, all of which give your main pseudonym positive reputation marks? All in all, Friedman and Resnick's advice to take your time before trusting an entity is well judged.

There are times, however, particularly when considering security issues, when allowing multiple identities can be a good thing. The use of roles to identify what capabilities an entity should be granted is typically a useful approach. It might be argued that multiple roles should be granted to a single pseudonym. Not only can this lead to the mixing of social contexts, but there are also times when a particular person needs to perform different roles: what (competent!) UNIX administrator would log in to a system as the root user for normal day-to-day work such as Web browsing or sending personal e-mail, for instance? There is a trade-off, therefore, at least at the design stage, between allowing multiple identities, the cost of identity creation, tracking and deletion, and the need for different roles within the system.

What this reminds us is that identity only exists within a context, and roles are a way of dealing with how different entities act—or are expected to act—in different situations. Trust, like identity, is a concept that only exists within a context, and given our earlier decision to restrict our discussion to interactions with entities that can perform "intentional and unwarranted actions," all such interactions occur within social contexts. This is true even if entities are programmed, as IRC robots are, as they are created to interact with other entities within a context that has a social foundation. The next section takes on this issue and looks at the social contexts within P2P systems.

Social Contexts

Advogato (Advogato, n.d.) is an online software forum skewed toward free software issues, with a stated aim to act as a "research testbed for work on group trust metrics." It has a trust metric (Levien, 2000) that allows members of the site to rate others which is quite sophisticated, and guards against malicious attacks using a set of trusted "seed members" and graph theory analysis to rate members' reputation. The certification process is described thus: "The ... system recognizes the combination of talent and dedication in three levels: Apprentice, Journeyer, and Master. All these levels are determined by peer certification" (Levien, 2000). A perceived problem has arisen as the site has grown in popularity in that members are being certified not for their "talent and dedication," but for the interest that other members have in the diary and article entries that are posted: many members use Advogato as a blog site. Because the social contexts within which members interact are mixed, confusion has arisen, and members are certifying to different criteria than those originally intended by the site designer. In the words of one member in a discussion of the problem, "The trust metric ain't broke; if anything, the trusters are" (Various, 2003).

A realisation that trust always has a social context allows us to make some sense of the statement at the beginning of the chapter. “I trust both my brother and my sister to act in a way which would safeguard my life in a context where that would be relevant, but neither of them has the expertise in the context of computing to back-up my hard drive.” In fact, it probably makes sense to refine the context more clearly. Although, given any random situation where my brother or sister might have the opportunity to save me, I would trust them to do their best, I would also actively seek their help in specific situations: if I were ill, I would seek my brother’s help (as he is a doctor), and if I were in a diving emergency, I would seek my sister’s help (as she is an extremely competent scuba diver and instructor).

In most situations in the “real” (off-line) world, that trust is expressed as an expectation that someone will perform an act in a particular context. When someone says “I trust you” to somebody else, it is usually clear what they are talking about, but within a P2P system, that you “trust” another entity is a null statement unless it is qualified. Even in non-P2P systems, it becomes clear very quickly that trust is context specific. Do I trust you to upload information to my Palm or other PDA? Do I trust you to download information from my PDA? Do I trust you to download information from my Web site? Do I trust you to upload information from my Web site? Do I trust you to give me information about where to find Newfoundland dogs? Do I trust you to give me information on good places to service my scuba gear (on which my life depends)? All of these questions could be answered by the statement “I trust you,” but they suggest very different social contexts.

Roles

The use of roles within systems is a well-understood technique in security (e.g., Anderson, 2001, pp. 54–58), and although the type of Access Control List (ACL) security that is typically associated with role-based security systems may not be appropriate for many P2P systems, allowing members of a system to have multiple roles may suit the needs of the system well. If particular roles are defined as suiting particular (well-defined) contexts well, this may present some value to the system. There are, of course, also trade-offs. Designing particular roles into a system may unduly reduce its opportunities for future growth—Advogato is valued by many of its members *because* of its blogging function, rather than despite it, for example.

All systems designers must make some decisions trading off future flexibility versus usability. Possibly more wide reaching than this is the danger that to use roles in this manner is a rather top-down approach to social context, and is based more on what interactions entities are likely to have with a system infrastructure, than on the interactions that they will have with each other. It is these latter interactions that define our interest in trust within P2P systems. It may be better to try to define social contexts more closely and then to use other techniques to allow entities to adopt roles that will fit those social contexts. This seems closer to the experience of people in the real world: people adopt the role or persona that is most appropriate for a particular set of interactions. In other words, they adopt the role suited to a particular social context.

Who, then, should define these roles and how they match particular contexts? It may be enough in many systems to define a very limited set of social contexts that are appropriate for the system; in other systems, it may be appropriate to attempt to implement standards for metadata about roles; and in yet other systems to allow emergent behaviour to dictate how roles develop. In some systems, this metadata is realised within a naming convention, allowing other entities to make a judgement—not always borne out by the facts—about the likely behaviour of an individual. The dangers in these types of system of "attacks" from within is attested all too vividly in recent convictions of paedophiles who have masqueraded as children within the social context of a chatroom (BBC Online, 2003).

Although such events are very real concerns, there are systems in which you do not necessarily need to be able to identify a real-world person/corporation, and there are others where, particularly for legal reasons, you may wish to do this. In almost all systems, however, you need to be able to track the identity within the system and over time: this is where public key cryptography in relation to identity is important, as outlined in the previous section. If you cannot track identity, you will have to hold a single—typically low—level of trust for each user.

One way to overcome the problem of allowing multiple roles for a single identity is to allow one identity to have multiple roles, and to present different roles within different social contexts. This may well be a plausible approach in some systems, and the creation of an infrastructure to support such a scheme would be an interesting research project in itself. Why, however, is there a need for such an infrastructure in the first place? In the real world, as already mentioned, we are very good at working out not only what social context we are interacting in, but also the role that another person is fulfilling—as well as how we should adjust our role or the persona we present for this social context. A large part of the problem is the lack of tacit communication within most online contexts. Davies (2003) points out that "off-line networks"—or sets of interactions—are better for tacit communications, and online networks better for codified communications. Schelling (1960), in one of the seminal works on game theory, also highlights the importance of tacit communication outside basic "Prisoner's Dilemma"-type interaction. But what if we wish to make use of tacit communication within a P2P system? There are various attempts to allow more tacit—

or at least more nonverbal—communications within online contexts, from "smileys" to video conferencing, and this, too, is a fertile area for future research, particularly in its application to the P2P sphere.

Transferral of Trust

The "chains of trust" typified by the PGP (originally Pretty Good Privacy) system, and the role of recommendation and reputation has already been briefly discussed above, but the wider issue of transferral of trust is important in any discussion of social contexts. Sztompka (1999, p. 75) notes that "transfers of reputations, and consequently of trust, from one domain to another are based on an implicit theory of human personality," and then cites Chong's (1992, p. 695) presentation of this theory, that "people have consistent personalities and traits and that their behavior is driven by them. Either they are a certain way (e.g., honest, fair, selfish, etc.) or they are not." Even if this *is* true across social contexts that does not necessarily mean that an entity is "fit for purpose." It was suggested above that successful recommendation services are those that allow an entity to recommend another in exactly this way: as "fit for purpose." Now that we have a better understanding of social contexts, we can see that this entails the acceptance of a shared social context. In fact, the act of accepting a recommendation is in itself an act of acceptance of a shared social context between the three entities in the interaction: the requester, the recommender, and the recommendee. When this shared context is implicit or assumed, problems can occur, and this is one of the dangers of "chains of trust" as exemplified by the PGP system. Within the PGP system, it is not clear exactly what one is saying when one signs another's public key—is it that you trust that they are who they say they are, that they have a continuous history as that individual, or just that they are the only person who will have access to the associated private key? If there is a subtle shift in context along the chain of trust, to what extent should I trust a person three links down the chain? Five links down the chain? Fifty links down the chain?

In fact, even if I do believe that the social context is well defined, it may well be that I am unhappy about trusting people too far down the chain. With a model of transferral of trust, we have to assume that each link is "strong," and that we can start with a known entity that is "close" to us (though see Abdul-Rahman and Hailes [1997] for an alternate view). How exactly "closeness" is defined is a perplexing problem, if only because it may vary widely between different social contexts. My knowledge of one person's fitness for purpose may be based on a single action observed at close hand—how well they drive in a difficult situation—on a set of examples of their work—programs they have written over a period of time—or on a long-term knowledge of their competence.

Even if the closeness between each link in the chain is equally high as the first measure, people are bad at realising that a number of unlikely but conjunctive events (such as the chance of a broken chain of trust where all the links in the chain enjoy a high probability of trustworthiness) are likely to yield a bad outcome (Tversky & Kahneman, 1982). This is another reason why we should be very wary of long chains of trust. Chains of trust are, for many models of interaction, not a very strong model—maybe reputation servers, where all recommenders are only one step or "link" away from the recommendee, can provide an improvement?

The problem with reputation servers is that with them we are working against the opposite problem. Rather than trusting that each link in the chain is strong, and starting with a link "close" to us, we typically need to assume that a large number of close links, even if they are not strong, can lead us to believe that a particular entity is "fit for purpose" within the social context that interests us. But many of these entities are *not* close to us, either in terms of trust, or—it may be—in terms of agreement about social context. As far as "closeness" to the recommender goes, this is a particular problem where cheap pseudonyms are available, and individuals can adopt multiple personae to recommend themselves—we would clearly be unwise to put too much faith in such a system. There is also a danger that individuals, or sets of individuals, can "flood" a system and cross-recommend each other, whether maliciously or due to lack of understanding as to appropriate measures: "My friend Simon's a good guy, and he's recommended me, so of course I'll recommend him back."

Time as Context

The implicit assumption about how close another recommender was to a requester of trust was that such a "closeness" was in terms of knowledge of the other and their fitness of purpose. Time is in itself a context, even if it fits only rather loosely into the "social context" category. Chijiiwa (n.d.) points out the relevance of timeliness to reputation, and there is a natural tendency for trust to decay if there is a lack of interaction with the entity you are being asked to recommend. This seems to run counter to the suggestion of Friedman and Resnick (1999) that we should wait some time before trusting "new" entities. However, once considered within the framework offered by Engquist and Leimar (1993), who argue that highly mobile organisms need time to learn about each other and to exchange "gossip," we begin to see such interactions within the social contexts that we have been examining. Of course, Friedman and Resnick have examined the issue of reputation from a strongly game theoretic viewpoint, and such a viewpoint tends to discount communication channels, though there is a wide and long literature of experiments with adding varying amounts of communication between different entities that may provide fertile ground for systems designers (e.g., Deutsch & Krauss, 1960; Wichman, 1970). All such communications take place within a time context as well, however, and there are often good reasons both for waiting for a while to get to know someone before trusting him/her and for letting the level of trust decay over time: both are important within a P2P system setting and neither is typically addressed in currently deployed systems.

Game theory may be less helpful within systems where there are multiple interactions, within social contexts and over varying amounts of time. Another useful discipline is the modeling of market economics—Coleman's theories on individualism within markets are often quoted (e.g., Coleman, 1986)—which can offer a number of lessons for P2P systems designers. Two examples are Hayek (1944), who offers a seminal view of benefits that liberal policies in the marketplace offer to individualism, and Marsden and Laumann (1977), who provide a good example of the use of mathematical modeling to study power bases and sources of influence in community elites. The differences between "perfect" markets and P2P systems should always be borne in mind. Although similarities such as actions over long and unbounded periods of time and interactions between distributed actors are useful, certain assumptions such as the instance availability of "perfect information," which are sometimes made in such models, are less likely to be reflected in P2P systems.

Some social contexts are themselves time related. In the earlier example about trusting my brother if I were ill and my sister if I were in a diving emergency, the timeliness of interaction is likely to be different in the two cases, as is the length of interaction. For most illnesses, action within hours, days, or even weeks is appropriate, as is the period of care. If action from my sister for a diving emergency were to take place on similar timescales, I might have occasion to regret not having placed my trust elsewhere!

Since we have strayed beyond the obvious confines of "security" in this discussion of social contexts, it is worth touching on two very different time-related trust issues that present real dangers within P2P systems. The first is the danger of entities taking a "long view" of the system, and building up a reputation for a single, big "sting." An example might be of a seller on eBay who gains an excellent reputation by selling items of increasing value before offering one last high-value item that he/she has no intention of supplying. There is very little that any design can do about such an attack, and an analysis of reputation would suggest that this user *should* be trusted, though risk analysis and an awareness of the cost of resources that could be lost may help. This is an area that game theory has examined from many angles, and Heap and Varoufakis (1995) offer a good critical introduction to this area and others.

Another area where time plays an obvious part is in the issue of public–private key pairs. Although most commercial certificates are time limited, many private individuals do not time limit their keys or associated certificates—and individuals are neither willing nor able to protect their private keys to the same extent as do commercial concerns. It is certainly sensible to trust keys less over time unless—and in many cases even if—there is a revocation infrastructure that can be leveraged from within the system.

Punishment and Deterrence

It appears that one of the problems from which eBay's reputation system suffers is that there is not enough negative comment or reputation distributed around the system (Resnick, 2003). In fact, no system involving security can operate for long without a credible system of deterrence and punishment. As before, this is an area that has been widely explored by game theory, but yet again, game theory can only offer us so much: it is most concerned about interactions between individuals, whereas in P2P systems, there are opportunities for far more complex types of interactions between many entities in multiple contexts. These types of complex interactions are like those modeled in economic models, and in markets the "laws" of supply and demand coupled with price fluctuations, provide deterrents and punishments for those who are perceived as "wrong-doers." However all but the most stringent proponents of free markets advocate at least some checks and balances, such as antimonopoly laws and regulations against price fixing by cartels. What options exist for P2P systems?

One way to deal with the problem is through the use of reputations, as already discussed. One issue with this approach is that reputation-based systems, when they are used for punishment, are open to the possible danger of movement into a dissonant social context. If reputation systems are reflections of fitness for purpose, then causing a reputation to be diminished may not reflect a diminution in fitness. Reputation becomes a currency in itself, and entities may go to excessive lengths to keep their reputation high—a process that may be reflected in Resnick's points above. Slashdot's "karma" system has historically suffered in this regard (CmdrTaco, 1999).

How, then, can punishment—and the expectation of punishment, which can provide deterrence—be implemented within a system? It seems that there must be a system of punishment that is appropriate for the social context within which the relevant interactions are occurring. There must also be a way of encouraging entities to dish out punishment to each other within this context—though this can be difficult to ensure, as we have seen—or there must be some type of justice system to which those wronged and those accused can turn.

Externally based systems lead to requirements for the types of trade-off that have already been discussed in the section on identity above. Given that punishment is likely to involve withdrawal of resources of some type, entities may wish to have recourse to legal structures. However, where a system is distributed, there may be very real questions about jurisdiction. Trade-offs between "liberty" and opportunities for enforceability of contracts for resources must be made.

Internal systems of punishment will either need to be designed in the system or emerge as properties of the wider system. Exactly how such systems can arise, and how they gain their authority, has been a subject of debate for several centuries, as evidenced by Hobbes' *Leviathan* (Hobbes, 1651). The rise in research into communities of practice, which can include an element of self-policing behaviour, may offer a basis for research in this area (Wenger, 1998). Whether a "legal system" is run by a self-imposed elite, is specified by the designers or "owners" of a system, is elected, or comes about by another means such as through a community of practice is a fertile ground for further research.

Trust and Security: Summary

This chapter has argued that P2P systems must be regarded from a social point of view, and mentioned some of the major issues that are likely to require thought during their design, implementation, and running. Although it has become clear that there are some major decisions and trade-offs that need to be considered for P2P systems, it should be stressed that the news is not all bad. The first reason for hope is that there is never likely to be a single system where all of the issues that have been raised will cause problems. The major aim of this chapter has been to raise awareness among designers, implementers, and users of P2P systems of the need to examine what trade-offs need to be made. Some decisions will be easy, some will be hard, but it is always better, when considering issues related to security, to know what trade-offs are being made *explicitly*, before they become implicit properties of a system.

The second reason for hope is that there is a wealth of evidence to suggest that socio-technical systems tend to exhibit significantly more resilience than purely technical systems. This can help to mitigate the complexity added to the field by the realisation that security in P2P systems involves a social dimension. There is also a large corpus of research into social structures, and how to encourage trust and resilience to grow within them (e.g., Brown, 2000; Fukuyama, 1995; Axelrod, 1990; Dasgupta, 1988; Good, 1988). The concept of "social capital" has gained considerable currency with the field of sociology, and is one that can be of great benefit to understanding the possibilities and constraints of a P2P system.

One last major question remains to be posed: "Who owns the system?" The simple answer, "The entities who comprise it," is typically not good enough, unless you believe in a truly fluid, anarchic system. If some structure and continuity is to be embodied within a P2P system, the question of authority, which has been touched on throughout this chapter, must be addressed. There are issues of legal authority, authority of identity provision, and of reputation—and almost any aspect of a system may feel the effect of a weak conception of authority. Should authority reside inside the system or outside it? Should it be emergent, imposed, or guided? Should it reside in a few entities or the entire "commonwealth" of the system? These are questions that we can expect to see arising more and more frequently as P2P systems are designed, implemented, and deployed.

Conclusion

Although security is a broad topic of discussion, this chapter has suggested that it is useful to examine the issue of security in peer-to-peer (P2P) systems from the standpoint of trust. It has taken the view that P2P systems present particular challenges in terms of trust over other socio-technical systems, and identified three key areas of importance: identity, social contexts, and punishment and deterrence. A better understanding of these areas and the trade-offs associated with them can help in the design, implementation, and running of P2P systems, and some of these trade-offs have been discussed. There is a growing corpus of research in P2P systems, and the chapter has looked at some of the issues that this research addresses, and how the wider sociological and nonsystems literature can help those involved with the building and maintenance of P2P systems. The chapter has concluded with some suggestions of areas where future research may provide fruitful insights, and one major question that needs to be addressed for a better long-term understanding of the needs of peer-to-peer systems: "Who owns the system?"

References

Abdul-Rahman, A., & Hailes, S. (1997). A distributed trust model. Retrieved April 4, 2003, from http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/hicss33.pdf

Advogato (n.d.). Retrieved October 15, 2003, from http://www.advogato.org/

Ainsworth, C. (2003). kuro5hin.org || Frequently asked questions. Retrieved October 30, 2003, from http://www.kuro5hin.org/special/faq

Anderson, R. (2001). *Security engineering: A guide to building dependable distributed systems*. New York: Wiley.

Axelrod, R. (1990). *The evolution of co-operation*. London: Penguin Group.

BBC Online (2003). Paedophile's sentence increased. Retrieved October 15, 2003, from http://news.bbc.co.uk/1/hi/england/cambridgeshire/3193246.stm

Brown, R. (2000). *Group processes: Dynamics within and between groups* (2nd ed.). Oxford: Blackwell Publishers.

Chijiiwa, Ryo (n.d.). Reputation economy and the Internet. Retrieved April 4, 2003, from http://ryo.iloha.net/?section=writing&page=reputation_econ

Chong, D. (1992). Reputation and cooperative behavior. *Social Science Information, 31*(4), 683–709.

CmdrTaco (1999). Slashdot moderation. Retrieved October 5, 2003, from http://slashdot.org/moderation.shtml

Coleman, J.S. (1986). *Individual interests and collective action: Selected essays*. Cambridge, UK: Cambridge University Press.

Cottrel, L. (1996). Frequently asked question about Mixmaster Remailers. Retrieved October 30, 2003, from http://www.obscura.com/~loki/remailer/mixmaster-faq.html

Dasgupta, P. (1988). Trust as a commodity. In D. Gambetta (Ed.), *Trust: Making and breaking cooperative relations* (pp. 49–72). Oxford: Basil Blackwell.

Davies, W. (2003). You don't know me, but ... : Social capital & social software. Retrieved September 27, 2003, from http://www.theworkfoundation.com/pdf/1843730103.pdf

Deutsch, M., & Krauss, R.M. (1960). The effect of threat on interpersonal bargaining. *Journal of Conflict Resolution*, *6*, 52–76.

Dingledine, R., Freedman, M., & Molnar, D. (2000). Accountability. In A. Oram (Ed.), *Peer-to-peer: Harnessing the power of disruptive technologies* (pp. 271–340). Sebastopol, CA: O'Reilly.

Dunbar, R. (1996). *Grooming, gossip and the evolution of language*. London: Faber and Faber.

eBay (2003). About eBay: eBay Community. Retrieved October 30, 2003, from http://pages.ebay.com/community/aboutebay/community/index.html

The emule project (n.d.). Retrieved October 13, 2003, from http://www.music-download-world.com/

Engquist, M., & Leimar, O. (1993). The evolution of cooperation in mobile organisms. *Animal Behaviour 45*, 747–757.

Everquest FAQ (2003). Retrieved October 30, 2003, from http://eqlive.station.sony.com/library/faqs/faq_eqlive.jsp

Fischer, M.J. (1987). A theoretician's view of fault tolerant computing. In B. Simons & A. Spector (Eds.), *Fault Tolerant Distributed Computing* (pp. 1–9). New York: Springer-Verlag.

Friedman, E.J., & Resnick, P. (1999). The social cost of cheap pseudonyms. Retrieved September 27, 2003, from http://www.orie.cornell.edu/~friedman/pfiles/anon.pdf

Fukuyama, F. (1995). *Trust: The social virtues and the creation of prosperity*. London: Hamish Hamilton.

Good, D. (1988). Individuals, interpersonal relations, trust. In D. Gambetta (Ed.), *Trust: Making and breaking cooperative relations* (pp. 31–48). Oxford: Basil Blackwell.

Gray, J. (1987). A comparison of the Byzantine agreement problem and the transaction commit problem. In B. Simons & A. Spector (Eds.), *Fault tolerant distributed computing* (pp. 10–17). New York: Springer-Verlag.

Hayek, F.A. (1944). *The road to serfdom*. Chicago: University of Chicago Press.

Heap, S.P.H., & Varoufakis, Y. (1995). *Game theory: A critical introduction.* London: Routledge.

Hobbes, T. (1651). *Leviathan.* R. Tuck (Ed.) (1996). Cambridge, UK: Cambridge University Press.

Lamport, L., Shostack, R., & Peace, M. (1982). The Byzantine generals' problem. *Communications of the ACM, 16*(10), 613–615.

Levien, R. (2000). *Advogato's trust metric*. Retrieved April 13, 2003, from http://www.advogato.org/trust-metric.html

Levien, R. (2000). *Certification.* Retrieved October 15, 2003, from http://www.advogato.org/certs.html

Marsden, P.V., & Laumann, E.O. (1977). Collective action in a community elite: Exchange, influence resources, and issue resolution. In R.J. Liebert & A.W. Imershein (Eds.), *Power, paradigms and community research.* London: Sage.

Orlowski, A. (2003). Is the RIAA "hacking you back"? Retrieved October 30, 2003, from http://www.theregister.co.uk/content/6/28842.html

Pfleger, C.P. (1997). *Security in computing* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Resnick, P. (2003). eBay live trip report. Retrieved September 27, 2003, from http://livejournal.com/users/presnick/8121.html

Schelling, T. (1960). *The strategy of conflict*. Cambridge, MA: Harvard University Press.

Schneier, B. (2000). *Secrets and lies: Digital security in a networked world.* New York: John Wiley & Sons.

Schneier, B. (2003). *Beyond fear: Thinking sensibly about security in an uncertain world*. New York: Copernicus Books.

Scoot (2003). About. Retrieved October 30, 2003, from http://www.scoot.co.uk/static/business/about/who_we_are.asp

Stewart, J. (2003). DNS cache poisoning — The next generation. Retrieved October 5, 2003, from http://www.securityfocus.com/guest/17905

Sztompka, P. (1999). *Trust: A sociological theory*. Cambridge, UK: Cambridge University Press.

Tversky, A., & Kahneman, D. (1982). Judgment under uncertainty: Heuristics and biases. In D. Kahneman, P. Slovic, & A. Tversky (Eds.), *Judgment under uncertainty: Heuristics and biases* (pp. 3–20). Cambridge, UK: Cambridge University Press.

Various (2003). "Certifier nullification and the Advogato trust metric," discussion thread on Advogato. Retrieved October 15, 2003, from http://www.advogato.org/article/609.html

Waldman, M., & Rubin, A. (2001). Trust. In A. Oram (Ed.), *Peer-to-peer: Harnessing the power of disruptive technologies* (pp. 242–270). Sebastopol, CA: O'Reilly.

Wenger, E. (1998). *Communities of practice: Learning, meaning and identity*. Cambridge, UK: Cambridge University Press.

Wichman, H. (1970). Effects of isolation and communication on cooperation in a two person game. *Journal of Personality and Social Psychology 16*, 114–120.

Yttebord, S.S. (2001). Frequently asked questions about PGPi. Retrieved October 30, 2003, from http://www.pgpi.org/doc/faq/pgpi/en/

Chapter VIII

Peer-to-Peer Technology and the Copyright Crossroads

Stacey L. Dogan
Northeastern University School of Law, USA

Abstract

The introduction of peer-to-peer technology has posed enormous challenges for traditional copyright law. Whereas historically copyright holders could preserve their core economic markets by pursuing a fairly limited set of commercial actors, peer-to-peer technologies decentralize content distribution and make it possible for anyone with a computer to disseminate infringing content around the world. In the struggle to find a solution to the peer-to-peer crisis, courts and policymakers have considered a number of alternatives, from abandonment of copyright law to a wholesale restructuring of its entitlement structure. This chapter explains why peer-to-peer technology presents such a challenge for copyright, and explores some of the pending proposals to solve the current dilemma.

Introduction

Before 1998, copyright law rarely made headline news, and few nonlawyers thought much about its existence. Today, copyright issues turn up on the front pages of newspapers and play a prominent role in public debate. As teenagers defend against infringement suits, courts shut down peer-to-peer networks, and social movements form in opposition to strong copyright law, it becomes increasingly clear that United States copyright policy is at a crossroads. Peer-to-peer technologies have forced an unprecedented reexamination of the purpose and meaning of copyright law in this country, with powerful voices engaged on all sides of the debate. Even traditional copyright allies—the software, music, and movie industries—find themselves in conflict over how the law should respond to the peer-to-peer crisis. And a number of serious legal scholars are advocating the outright abolition of digital copyright.

Why has peer-to-peer technology wreaked such havoc for copyright law? And what, if anything, should judges and policymakers do about it? This chapter considers these questions. It has three primary goals: (1) to explain copyright's economic objectives and why file sharing challenges them; (2) to explore the legal arguments for and against using copyright law to prohibit the distribution, operation, and/or use of peer-to-peer software; and (3) to introduce some of the alternatives that are currently being considered to address the peer-to-peer copyright crisis.

The Economic Foundations of United States Copyright Law

Copyright in the United States exists to induce people to make creative works. The law rests on an assumption that, without legal protection against copying, authors and publishers would under-invest in expressive works such as books, movies, music, and software. By granting an exclusive right to copy and distribute, the law insulates copyright holders from competition by low-cost copiers who did not incur the expense of creation and production, and thus offers an incentive to others to invest in these activities.

In economic terms, copyright law is justified by the "public goods" nature of creative expression. A public good is one that is both nonrivalrous (multiple parties can possess it simultaneously without interfering with one another's possession) and nonexcludable (once released to the public, its creator cannot physically exclude others from accessing and using the work). Economic theory predicts that, without some legal means to limit the use of such goods, the public would attempt to free ride by taking the good without contributing to its production costs. Creators, unable to capture the value generated by their works, would fail to produce works whose net benefits outweigh their costs, resulting in a classic market failure. Essentially, copyright law gives creators the legal right to demand payment from those that use their works in certain ways, which facilitates markets for such works and, hopefully, provides an engine for creative production.[1] At the same time, copyright law expressly allows for fair use of copyrighted works by educators, commentators, and others whose use, on balance, benefits the public and does not threaten copyright holders' core economic markets.[2]

The economic orientation of United States copyright law has played a continuing role in shaping authors' rights in the face of new technologies. As new markets for creative expression have emerged, Congress has generally reserved them to copyright holders, albeit sometimes with a caveat. In the early 20th century, for example, after a Supreme Court ruling that the right to copy included only human-readable forms (*White-Smith Music Publ'g Co., Inc. v. Apollo Co.* [1908]), Congress amended the Copyright Act to give authors rights over mechanical reproductions of their works, in forms such as piano rolls and records. Because a single company dominated the market for piano rolls, however, Congress added a compulsory license that allowed competing manufacturers to sell recordings of music as long as they paid a fixed fee to the music copyright holder. This so-called mechanical license is still in effect, along with more recently enacted compulsory licenses for activities such as cable and satellite broadcasting (17 U.S.C. §§ 111, 119).

Despite these adjustments to the contours of its protection, copyright law through the late twentieth century retained a critical feature: it defined the copyright holder's economic markets—and their exclusive rights—by reference to actual uses of copyrighted works. The Copyright Act of 1976, for example, gave copyright holders the exclusive right to copy, to distribute copies to the public, to perform or display works publicly, and to create adaptations or modifications of the copyrighted expression (17 U.S.C. § 106).[3] Others who benefited indirectly from these activities—such as manufacturers of printing presses, record players, and other technologies that people occasionally used to infringe—were free to sell their products without interference from the copyright holder. Copyright law reached only two narrow subsets of people who did not themselves infringe: those who controlled and profited from direct infringers' behavior and those who actively helped others to infringe. These forms of liability—called vicarious and contributory liability, respectively—were narrowly defined and rarely invoked.

This emphasis on those who copied, sold, modified, or performed works made sense given the way in which creative content was produced, sold, and distributed in the pre-Internet age. Before the late twentieth century, creative content was distributed in physical copies whose creation and dissemination required substantial investment. To capture the primary economic markets for creative works (and thus to preserve their financial incentive to invest in creation and distribution), copyright holders needed only to focus on a core set of visible, commercially motivated actors who profited from exhibition, reproduction, and distribution. In effect, copyright law facilitated the growth of these distribution markets by enabling revenue-generating licensing deals that in turn motivated further investment in creation and distribution. By insisting that those who commercially copied, sold, or performed expressive content obtain a license from the copyright holder (usually at a price), copyright law allowed creators and distributors to capture the value of their investment in creative works, while at the same time allowing copying technologies to develop without interference and leaving individuals free to engage in personal, small-scale uses of works that did not threaten the copyright holders' core markets.[4]

Peer-to-peer technologies challenge this legal and economic framework in two fundamental ways. First, by empowering individual users to distribute content files without any centralized gatekeeper,[5] the technologies debunk the historical assumption that copyright holders could capture their core markets by insisting on licenses from commercial copiers and distributors who actively handled their content. Second, by emphasizing communitarian values such as cooperation and sharing, peer-to-peer networks preserve the appearance that their members are engaging in the kinds of activities that historically fell beyond the reach of copyright, while at the same time promoting behavior whose economic impact is hard to distinguish from the commercial distribution of an earlier age. Unconstrained sharing of books, movies, music, audiovisual works, video games, and software, in other words, threatens to displace legitimate sales of those products in the same way that pirated hard copies did in the past.[6] Yet, because today's sharing occurs between individuals without any personal profit motive, much of the public views it as more akin to personal use than to infringement. As a practical matter, even if we view such sharing as infringement, locating and pursuing its perpetrators is no easy task.

Peer-to-peer technology therefore poses serious challenges to copyright. While recent music lawsuits have received the most attention, the peer-to-peer crisis in copyright law extends well beyond the music industry to encompass virtually all information products that satisfy a low threshold of creativity. Software, musical scores, movies, photographs, video games, novels, dictionaries, textbooks, maps, screenplays, even some databases—all of these products and more are protected by copyright, and if copyright law proves inadequate to curb peer-to-peer distribution, creators and publishers in all of these industries risk a decline in at least one form of revenue on which they have historically depended—the sale of copies of their products. Whether we should celebrate such a development or despair of it is the subject of much debate, and we will consider that question at the end of this chapter. But that debate begs the question of whether existing copyright law has enough teeth to bring peer-to-peer infringement under control. The next section considers that question.

Copyright Rules and Peer-to-Peer Networks

Scholars, policymakers, legal experts, and the public have remarkably diverse viewpoints on the extent to which copyright law applies to peer-to-peer networks. Many people view unauthorized file sharing as private, noncommercial behavior that should fall beyond the reach of copyright. Others view uploading and downloading copyrighted files as infringement but think that peer-to-peer software developers, rather than individuals, should be held accountable (Lichtman & Landes, 2003). A third group, concerned about copyright holder interference with evolving technologies, advocates exactly the opposite result: harsh sanctions against individual file sharers and legal immunity for peer-to-peer network designers (Lemley & Reese, 2004). The courts, so far, have shown little harmony in their own approaches to these questions. A full appreciation of these viewpoints, as well as the recent lawsuits against peer-to-peer networks and those who use them, requires a brief detour into the rules of copyright law.[7]

Copyright Rules

As discussed above, copyright law gives qualifying authors a series of exclusive rights to use their works in certain ways. Only original works are entitled to protection, and United States federal law requires that the work be stored in some “tangible medium of expression” (17 U.S.C. § 102(a)). Because originality demands only a modicum of creativity, and because fixation includes machine-readable forms, most files traded on peer-to-peer networks easily satisfy these thresholds. And while the law excludes “ideas, processes, and methods of operation” (17 U.S.C. § 102(b)) from the subject matter of copyright, protection extends to many expressive aspects of functional works, including most forms of computer code.[8]

Copyrighted works currently receive protection for an extraordinary length of time: the life of the author plus 70 years for individually authored works, and up to 120 years for employee-created works.[9] Nonetheless, many works created in the last century have fallen into the public domain, either because their authors neglected to comply with prior legal requirement such as registration or renewal,[10] or because the work's copyright term expired before the most recent term extensions went into effect.[11] Many classic photographs, early musical recordings, and other valuable works may thus be copied and distributed without infringing copyright.

Among the exclusive rights reserved to the copyright holder, the most pertinent in the peer-to-peer context are the right to copy and the right to distribute. A copyright holder has the exclusive right "to reproduce [her] work in copies or phonorecords," and "to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership, or by rental, lease, or lending" (17 U.S.C. § 106). "Copies" include any fixed embodiment "from which the work can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device" (17 U.S.C. § 101), a definition that easily covers data files transferred over peer-to-peer networks.

The exclusive rights of copyright holders, however, are not absolute. The Copyright Act contains 16 separate provisions limiting copyright owners' ability to prevent use of their content in particular contexts. The broadest exception, and the one most commonly invoked by peer-to-peer advocates, provides that "the fair use of a copyrighted work, … for purposes such as criticism, comment, news reporting, teaching … , scholarship, or research, is not an infringement of copyright" (17 U.S.C. § 107). The Act lists a number of factors that courts should balance in deciding whether a use is fair, including whether the use was commercial, whether the user transformed the expression in some way, whether the copied work was highly creative or more factual, and whether uses like the one at issue are likely to harm potential markets for the copyrighted expression.

Of course, defining copyright holders' exclusive rights achieves only part of the law's objectives; the law must also assign responsibility when those rights are violated. Copyright law does this in two ways. First, it provides that anyone who commits one of the acts reserved to the copyright holder without authorization is an infringer (17 U.S.C. § 501). Second, through a series of judge-made rules, copyright law extends legal responsibility to other parties who did not themselves infringe, but had an intimate relationship either to the infringement or to the party committing it. The doctrine of contributory infringement, for example, contemplates liability against one who, "with knowledge of the infringing activity, induces, causes, or materially contributes to the infringing conduct" of another party (Gershwin Publ'g Corp. v. Columbia Artists Mgmt., Inc., 1971; A&M Records, Inc. v. Napster, Inc., 2001).[12] Vicarious liability is imposed against parties who have a right and ability to control direct infringers' behavior, and who receive a direct financial benefit from the infringement (Shapiro, Bernstein & Co. v. H.L. Green Co., 1963).[13]

Historically, vicarious and contributory liability were imposed mainly against enterprises whose businesses encompassed or actively facilitated infringement, such as dance hall owners and agents for infringing performers.[14] In the late 1900s, however, the character of vicarious and contributory infringement claims began to change. With the growth of personal copying technologies such as photocopiers, VCRs, and cassette tape recorders, copyright holders complained that individuals were committing infringement with these technologies, and that those who made and sold the machines should be held to account. For the first time, copyright holders sought to establish third-party liability based on the sale of products that could be used to infringe.

The Supreme Court first considered such a claim in 1984, in *Sony v. Universal City Studios* (1984). In *Sony*, a group of movie studios sued the manufacturer and distributors of the Betamax video cassette recorder, claiming that people were using the machine to infringe and that the defendants were vicariously and contributorily liable for that infringement. In resolving the case against the studios, the Supreme Court made two legal rulings that have had a profound impact on the development of copyright law in recent decades. First, the Court held that so-called time shifting of off-the-air television programs constituted fair use, given its noncommercial nature and the absence of any proven harm to the market for the copyrighted shows.[15] Second, the Supreme Court held that the existence of this "substantial noninfringing use" insulated Sony from liability under copyright law: "[T]he sale of copying equipment, like the sale of other articles of commerce, does not constitute contributory infringement if the product is widely used for legitimate, unobjectionable purposes. Indeed, it need merely be capable of substantial noninfringing uses" (Sony v. Universal City Studios, 1984, p. 442). The Court reasoned that to allow copyright holders to sue based on the infringing use of such "staple articles of commerce" would give them undue influence over markets "substantially unrelated" to their copyright grant (p. 442).

Peer-to-peer advocates have argued that *Sony* protects both users and developers of peer-to-peer networks. So far, the arguments have met with a mixed reception in the courts. On the fair use question, judges have shown little sympathy for the attempt to equate file sharing with time shifting of television programs, based on differences in the scale of the behavior and in its likely market impact (A&M Records, Inc. v. Napster, Inc., 2001). Nonetheless, even if unauthorized sharing of copyrighted content constitutes infringement, peer-to-peer services undeniably have noninfringing uses—including sharing that some copyright owners authorize and sharing of uncopyrighted content—and courts have struggled with the legal significance of that fact. The following discussion explains why courts have held unauthorized file sharing to be infringement, and explores the different approaches that courts have taken in considering the peer-to-peer services' liability for that behavior.

Uploading and Downloading as Infringement

Although copyright actions against individual file sharers have not yet made their way through the courts,[16] a number of courts have addressed the legality of file sharing behavior in the context of contributory and vicarious liability claims against peer-to-peer providers. The only detailed treatment of the question came in the first case to consider it, *A&M Records v. Napster* (2001).[17] In *Napster*, a group of music copyright holders sued the operators of the Napster peer-to-peer network, claiming that Napster users were committing massive copyright infringement and that Napster itself was legally responsible for their behavior. The trial court agreed with the music plaintiffs and entered an injunction requiring Napster to block the trading of infringing files. On appeal, the Ninth Circuit Court of Appeals affirmed as to liability, although it remanded for entry of a narrower injunction. While the courts were ultimately concerned with Napster's liability—not that of its users—the contributory and vicarious liability claims depended critically on a showing that those who used Napster's service to trade files without authorization were themselves engaged in infringement.[18]

The Ninth Circuit had little trouble concluding, as an initial matter, that Napster users had engaged in unauthorized reproduction and distribution of copyrighted music files. "Napster users who upload file names to the search index for others to copy violate plaintiffs' distribution rights" (p. 1014).[19] "Napster users who download files containing copyrighted music violate plaintiffs' reproduction rights."[20] While equating downloading with copying finds support in the law,[21] it is significant that the court found a violation of the distribution right based on the offering of music files to the public. The court, in other words, suggested that a user violates the distribution right by making music files available to the public, even if no one takes up the offer to download those files from the user's shared directory. While the distinction may not have made a difference in *Napster*—where the evidence showed both uploading and downloading of music files across the network as a whole—it could have consequences in individual suits, when plaintiffs frequently have evidence only that the defendant made files available for sharing.

Having concluded that Napster users were distributing and copying plaintiffs' files, the court then considered Napster's fair use defense. Napster relied heavily on *Sony* to claim that two significant uses of its service were noninfringing: "sampling, where users make temporary copies of a work before purchasing," and "space-shifting, where users access a sound recording through the Napster system that they already own in audio CD format…" (p. 1014).[22] The Ninth Circuit, however, found important differences between *Napster* and *Sony*. For one thing, the court noted that most Napster users were using downloads to substitute music purchases, which the court described as a commercial, rather than noncommercial, use (p. 1015). Even "samplers," the court added, made exploitative use of the material they sampled, given the music industry's heavy control over promotional downloads and the revenue that they generated (p. 1018). Perhaps most significantly, the court noted that the worldwide scale of peer-to-peer distribution made it qualitatively different from the time shifting at issue in *Sony*. Even if individuals intended to use the Napster service primarily for space-shifting purposes, "it is obvious that once a user lists a copy of music he already owns on the Napster system in order to access the music from another location, the song becomes 'available to millions of other individuals,' not just the original CD owner" (p. 1019).

Because the court in *Napster* was addressing fair use in the abstract—that is, without the benefit of any end user's testimony about his/her own purposes and motivations in using the Napster service—the court's analysis painted a broad brush and did not address subtle differences in how people actually use peer-to-peer services. As the end-user cases make their way through the courts, judges will have to evaluate fair use claims by individuals, many of whom may present more particularized evidence of their use of peer-to-peer networks to sample new music or to space-shift music that they own. It is significant that, thus far, the suits against file sharers have involved only uploaders, rather than downloaders of copyrighted songs. While courts considering these cases might take different approaches to the details of the fair use analysis,[23] however, it seems fairly clear that the sheer scope and distributional impact of uploading on peer-to-peer networks makes it different in nature and effect from the time shifting at issue in *Sony*, and that most courts will find unauthorized uploading to constitute infringement.

The fact that some unauthorized file sharing constitutes infringement, however, by no means suggests that all file sharing is unlawful. Peer-to-peer networks have legitimate uses, including the sharing of files whose copyright has expired and the sharing of data files with permission. Many small artists, independent music labels, and open-source software advocates, for example, encourage the sharing of their work on peer-to-peer services. People who use peer-to-peer networks for these purposes are engaged in perfectly legitimate behavior.

Nonetheless, it seems indisputable that the vast majority of file sharing, at least until now, has consisted of unauthorized trading of copyrighted files. In the face of this widespread infringement, copyright holders have sought to hold peer-to-peer services and software providers legally responsible for their users' behavior. These suits—a number of which are pending as this book goes to press—raise many of the same issues as *Sony*. In particular, the courts must resolve whether the existence of noninfringing applications should insulate peer-to-peer providers from liability for users' infringement. This question—and the more general question of what *Sony* means—have so far befuddled the courts and divided the scholarly community.

The results of this struggle will have significant consequences. If peer-to-peer technologies generally are immune from copyright liability, then copyright holders seeking to contain peer-to-peer infringement have only one option under existing copyright law—to pursue end users in direct infringement suits in the hope that they can deter unauthorized behavior. If, on the other hand, peer-to-peer services are found liable, then they will have to shut down or redesign their products to reduce or eliminate infringement. The differences of opinion over peer-to-peer liability reflect, in part, philosophical differences over the underlying goals of copyright law. Many of those who believe that the law should maximize efficiency believe that peer-to-peer providers are in the best position to avoid the costs of rampant copyright infringement on their services, and should be forced to redesign their products to reduce infringing behavior. Others, focused on the *Sony* distinction between infringement and infringement-enabling tools, argue that liability would elevate the interests of copyright holders over those of the public, with long-term costs to the progress of technological innovation. The courts have, sometimes within the same legal opinion, shown sympathy to each of these positions, leading to much confusion over the applicability of copyright law to peer-to-peer services and software providers. The following section will discuss these cases.

Liability of Peer-to-Peer Services and Software Providers

The Ninth Circuit Court of Appeals in *Napster* offered the first judicial input on peer-to-peer service liability. *Napster* involved a fairly centralized file sharing service, which facilitated file trading through central servers and an index of music files available from its users. Any time a user logged onto Napster, the names of songs that he/she wished to share would be uploaded to the Napster server and available in the searchable index, accessible to all other logged-on users. Despite defendants' protestations that they had ambitions of promoting unknown artists, the evidence in the case revealed Napster's raison d'être as the unauthorized trading of copyrighted music files.[24] After deciding that such behavior was indeed infringement, the court had to resolve whether Napster was responsible for that infringement or whether, as Napster claimed, *Sony* protected it from liability.

The Ninth Circuit found Napster liable under theories of contributory and vicarious liability, in a fact-specific and sometimes ambiguous decision that left many open questions about the legal status of later-generation peer-to-peer services. Its treatment of *Sony*, moreover, reflected an odd interpretation with which a number of courts and commentators have already disagreed.

The court first addressed contributory liability, which requires knowledge of and material contribution to, the infringement of another party. Rather than treating *Sony* as a separate inquiry, the Ninth Circuit effectively folded it into the contributory infringement analysis. According to the court, the Supreme Court's sole objective in *Sony* was to avoid imputing knowledge of infringement to parties that sold technologies with both infringing and noninfringing uses (A&M Records, Inc. v. Napster, Inc., 2001, p. 1020). In other words, the Ninth Circuit viewed *Sony* as narrowly addressed to the knowledge element of contributory infringement claims, and designed to prevent courts from inferring that those who sold copying technologies knew specifically that buyers would use them to infringe. Because Napster, unlike *Sony*, had "actual, specific knowledge of direct infringement" on its service, the court found "*Sony's* holding of limited assistance to Napster" (p. 1020). Interestingly, the court suggested that if *Sony* did apply, Napster's noninfringing applications would almost certainly qualify as "substantial non-infringing use" (p. 1021),[25] but its knowledge made a *Sony* defense unavailable.

Commentators have criticized the Ninth Circuit's constricted reading of *Sony*, contending that the focus on knowledge undercuts the Supreme Court's broader goal of preventing copyright holder interference in unrelated markets. Professor Paul Goldstein, for example, has pointed out that "[t]he substantial non-infringing use doctrine serves a purpose entirely separate from the knowledge requirement and, by subordinating it to the knowledge requirement, the Napster court necessarily undermined the object of the doctrine: to ensure that consumers not be required to pay monopoly tribute for unpatented or otherwise unprotected goods and equipment" (Goldstein, 2000, § 6.1.2).

The critics are almost certainly correct to point out that the Supreme Court intended the staple article of commerce doctrine to apply even when knowledge was established—that is, even when the plaintiff had proven both elements of a contributory infringement claim. Nonetheless, the Ninth Circuit, through its odd knowledge-based interpretation, put its finger on an important difference between Napster and the defendants in *Sony*: Napster, unlike Sony, maintained an ongoing relationship with its users that made it at least possible to act upon knowledge of infringement, and to obstruct trading of infringing files before it occurred.[26] When read in context, the Ninth Circuit's willingness to impose liability depended critically upon the pairing of knowledge and capacity to block.[27] And the court's ruling that Napster could block content, in turn, relied upon the Napster architecture, which maintained a centralized index of files that Napster could use to identify and remove infringing files. The court's ruling on contributory infringement, therefore, turned on three critical facts: that Napster received specific notice of infringing files on its service, that it could have blocked access to those files after receiving such notice, and that it failed to do so.

The same facts that persuaded the court of Napster's contributory liability also proved critical in evaluating the vicarious infringement claim. Vicarious liability requires that the defendant receive a direct financial benefit from infringement committed by someone within his/her control.[28] The Ninth Circuit first found that Napster had enjoyed a direct financial benefit because "Napster's future revenue is directly dependent upon 'increases in user base,'" which in turn depend on the quality and quantity of music available on the service (p. 1023). While some commentators have criticized this conclusion because Napster had earned no revenues at the time of the suit, the facts showed that Napster planned to become a profitable enterprise, and that the infringement directly enhanced the value of that enterprise (A&M Records, Inc. v. Napster, Inc., 2000, p. 902, 921–922). This may not fit the traditional, revenue-based model of direct financial benefit, but it certainly qualifies as a financial benefit in economic terms. On the second requirement—that Napster have a right and ability to supervise its users' conduct—the Ninth Circuit turned again to the architecture of the Napster system, and particularly the central index: "Napster … has the ability to locate infringing material listed on its search indices, and the right to terminate users' access to the system. The file name indices, therefore, are within the 'premises' that Napster has the ability to police" (A&M Records, Inc. v. Napster, Inc., 2001, p. 1024). The court found that after receiving notice of infringing files on its system, Napster had a duty to make good-faith attempts to purge those and similarly named files (p. 1024).[29]

Cutting through the legal doctrine, the *Napster* decision suggested a rule of thumb for peer-to-peer services, under which their liability would turn on (1) whether they received specific notice of files traded using their software, and (2) whether the architecture of their system enabled them to weed out infringing files. Not surprisingly, peer-to-peer networks designed after *Napster* adopted a new, decentralized architecture that no longer relied upon central servers or indices to locate and transfer data files.[30] While these decentralized services vary somewhat from one another, they generally avoid centralization by harnessing the power of users' computers and operating as a set of interconnected mini-networks. Because these peer-to-peer software providers' own servers no longer play an ongoing role in searching for music files, they arguably cannot, even after receiving notice, identify and block unauthorized files that are traded using their software. Preliminary results suggest that this strategy just may be working, at least in the Ninth Circuit.

Three years after it decided Napster, the Ninth Circuit considered a suit asserting contributory and vicarious liability claims against a group of decentralized peer-to-peer services, including StreamCast Networks and Grokster (Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 2004).[31] Relying heavily on its earlier emphasis on the centralized features of the Napster service, the court held that the absence of such an index made the *Grokster* defendants more like a Betamax video recorder than like the interactive network at issue in *Napster* (pp. 1163-1164). In the *Grokster* court's view, the question of contributory liability, under *Napster*, is whether the defendants "had 'specific knowledge of infringement at a time at which the contribute[d] to the infringement, and [] fail[ed] to act upon that information'" (p. 1162, quoting Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 2003, p. 1036). Because the decentralized services simply released their software and had little ongoing contact with their users, the court held that they could not stop infringement as it occurred. As a result, these peer-to-peer services were eligible for consideration as staple articles of commerce; because the defendants did a thorough job of documenting actual noninfringing uses,[32] the court had no trouble concluding that *Sony* immunized them from liability. The lack of centralized control also immunized Grokster from vicarious liability, given the absence of any "point of access for filtering or searching for infringing files" (pp. 1164-1166). In the Ninth Circuit, then, peer-to-peer services can escape liability as long as their system design makes it infeasible to monitor users' conduct at any particular moment. The move to a decentralized structure thus effectively insulates peer-to-peer services from liability under copyright law.

While most of the case law so far has come from the Ninth Circuit, the Seventh Circuit Court of Appeals (which sits in the Midwest, as opposed to the West Coast) has offered a somewhat different approach to the question of peer-to-peer liability under copyright law. The defendants in *In re Aimster Copyright Litigation* (2003) offered a peer-to-peer service designed to complement an instant messaging service, such as AOL's Instant Messenger. After downloading the Aimster software, users could make music files available to other Aimster users generally, or to a more limited set of "buddies" that they specified up front (p. 646). To acquire music files, users had a choice. They could enter a search term, after which the Aimster server would search all available files and, after finding a match, would instruct the computer holding the file to transfer it to the requesting user. Alternatively, users could pay a fee and join "Club Aimster," which offered a simplified process for downloading top-40 pop music hits. Unlike Napster, Aimster also offered encryption software, which encrypted all of the file transfers and, Aimster argued, made it impossible for it to have knowledge of which music files were traded on its service (p. 650).

A music industry coalition filed suit against Aimster, and the district court entered a preliminary injunction, finding Aimster likely to succeed on its contributory and vicarious liability claims (p. 646).[33] The Seventh Circuit affirmed in an expansive opinion written by Judge Posner, who took the opportunity to reflect on the meaning of contributory infringement and the import of *Sony*. The court offered a mixed bag of comfort and disappointment to both sides of the peer-to-peer debate.

The court began by addressing *Sony*. Unlike the Ninth Circiut court, Judge Posner interpreted *Sony* as applying even in cases in which a defendant has actual knowledge of infringement using its products. The court "agree[d] with Professor Goldstein that the Ninth Circuit erred in [*Napster*] in suggesting that actual knowledge of specific infringing uses is a sufficient condition for deeming a facilitator a contributory infringer" (p. 649). He also found that *Sony* applied to services, not just products: "If a service facilitates both infringing and noninfringing uses, as in the case of AOL's instant-messaging service, and the detection and prevention of the infringing uses would be highly burdensome, the rule for which the recording industry is contending could result in the shutting down of the service or its annexation by the copyright owners (contrary to the clear import of the *Sony* decision), because the provider might find it impossible to estimate its potential damages liability to the copyright holders and would anyway face the risk of being enjoined" (648–649). By finding *Sony* potentially available to services, regardless of their actual knowledge of infringement, the Seventh Circuit decision appears, at a glance, more generous to defendants than *Napster*.

What he gave with one hand, however, Judge Posner took away with the other. In contrast to the Ninth Circuit which held that *Sony*, where applicable, exempts from liability any product with a *potential* noninfringing use (A&M, Inc. v. Napster, Inc., 2003, p. 1021) the Seventh Circuit suggested that only *actual* noninfringing uses should factor into consideration of a staple article of commerce defense. "It is not enough," held the court, "that a product or service be physically capable, as it were, of a noninfringing use" (In re Aimster Copyright Litigation, 2003, p. 651–653). Even when a service qualifies as mixed use, moreover, Judge Posner indicated that the service provider might, in some circumstances, have to redesign its product to eliminate infringement:

Even when there are noninfringing uses of an Internet file-sharing service … if the infringing uses are substantial then to avoid liability as a contributory infringer the provider of the service must show that it would have been disproportionately costly for him to eliminate or at least reduce substantially the infringing uses. (p. 653)

The court also indicated that a ruling on contributory infringement must take into account "some estimate of the respective magnitude" of infringing and noninfringing uses of such as service (pp. 649–650).

Judge Posner, in other words, did not view *Sony* as a complete haven for services with actual noninfringing applications. In Posner's view, even services with noninfringing uses can face liability if their infringing uses are substantial and could have been avoided without "disproportionate[ ]" cost (p. 653). Posner suggested, without deciding, that the encryption feature of the Aimster service might, on balance, contribute more to facilitating infringement than to any legitimate, noninfringing goals, making it a likely candidate for redesign (p. 653). Given the early stage of the proceedings, however, Judge Posner reached no ultimate conclusions and offered little additional guidance on how courts should go about their task of balancing infringing and noninfringing applications and weighing the cost of redesign against the burden imposed on noninfringing users.[34]

Taken together, these cases leave as many questions as answers about the extent to which copyright law applies to peer-to-peer providers. On the one hand, the courts appear sympathetic to copyright holders' claim that the law should reach those who design services primarily to enable others to infringe. On the other hand, peer-to-peer services undeniably have noninfringing applications, and the courts are struggling over the legal implications of that fact. Does *Sony* require courts to immunize peer-to-peer services from liability in order to protect their noninfringing uses, or did the Supreme Court leave room for a more flexible approach? Both of the appellate courts that have considered this question have found some room for flexibility, though their rationales diverge quite significantly. In the Ninth Circuit's view, *Sony* does not preclude liability against a peer-to-peer service that receives notice of infringement on its network at a time when it *could* block or remove the infringing file. Without specific knowledge and the ability to block, however, the Grokster opinion suggests that *Sony* requires complete immunity for peer-to-peer providers. The Seventh Circuit, on the other hand, would invoke *Sony* even in some cases involving specific knowledge and ability to block, but might sometimes require redesign of technologies whose infringing uses far outweigh their noninfringing applications. As peer-to-peer networks become further decentralized, the Ninth Circuit approach suggests a move away from peer-to-peer liability, while the Seventh Circuit standard leaves open questions over the relative significance of infringing and non-infringing uses, and the burden that redesign would impose on noninfringing uses of peer-to-peer services. The division in the courts of appeals suggests that the Supreme Court will ultimately have to resolve the issue, and give the lower courts greater guidance in cases involving mixed-use technologies in an Internet age.

As a practical matter, a strategy limited to lawsuits against peer-to-peer services will do little to stem the tide of peer-to-peer infringement, at least in the short term. For one thing, whatever legal standards the courts develop, creative technology developers will find some way to skirt them, just as the developers of Gnutella and KazAa designed around the rules that the Ninth Circuit adopted in *Napster*. As the courts grapple with the implications of these new designs, the software will proliferate and infringement will persist. Perhaps more important, as this book makes clear, peer-to-peer technology is developing into more than just an engine for massive copyright infringement. As peer-to-peer moves further away from its Napster-based roots, courts will find it hard to deny the significance of its noninfringing applications. Unless the Supreme Court abandons the staple article of commerce doctrine, it seems likely that many peer-to-peer services will escape liability under copyright law, leaving consumers with the tools to engage in widespread infringement. The solution to the peer-to-peer crisis, then, cannot rely on secondary liability claims alone.

The Road Ahead

Peer-to-peer technology unquestionably challenges the traditional structure of content markets and the efficacy of traditional copyright law. Whether and how the law should respond to that challenge remains the subject of intense debate in political, technological, and academic circles. Indeed, the peer-to-peer issue has forced a reexamination of the underlying rationale of rights-based copyright law, with some arguing that the law has outlived its purpose of motivating creative expression (Ku, 2002). Others believe in the continuing relevance of copyright, but differ over the adjustments required to protect content markets (and therefore to preserve economic incentives) in a peer-to-peer age. In the final section of this chapter, we will consider some of the current proposals for resolving the peer-to-peer copyright crisis. Essentially, the proposals break down into three approximate clusters: those who advocate the abolition and/or replacement of rights-based copyright law; those who back a major fortification of existing exclusive rights; and those who support a wait-and-see approach, given the prospect that the combination of existing legal efforts and emerging market forces may realign themselves into a new equilibrium.

A. A Post-Copyright Paradigm?

When the Internet began to emerge as a worldwide distributional tool in the mid-1990s, many people argued that the new medium would ultimately make copyright law irrelevant. Copyright law, they contended, had long served primarily to subsidize content distribution, not creation, and as global distribution became available for free, creators would opt for it as the best way to reach the broadest audience. To some extent, these visionaries were right: the explosion of creative content on the Internet, voluntarily posted by authors, attests to the fact that many people are motivated more by the desire to be heard than by the traditional, rights-based economic incentives of copyright law. At the same time, free Internet distribution has not displaced traditional publishers as the intermediary of choice for creators of many valuable copyrighted works. Developers of software, movies, music, and books have continued to sell (or to license publishers to sell) copies of their works, and have policed those who distribute them without permission. And while the early Internet presented challenges for copyright holders seeking to protect their rights, a combination of judicial and Congressional decisions initially left them the tools to enforce copyright against Internet distributors.[35]

The introduction of peer-to-peer technology, however, breathed new life into the arguments for a post-copyright paradigm for two reasons. For one thing, peer-to-peer technology enabled a whole new level of the "individual autonomy, self-expression, and creative collaboration for which we celebrate the Internet" (Netanel, 2003; Fisher, 2004, ch. 6). The interactive, iterative, and communitarian nature of peer-to-peer "sharing," in this view, was something to be celebrated and valued, and would be largely lost if copyright law were successfully applied to the new medium. Second, commentators began to emphasize the social costs imposed by exclusive rights-based copyright law. Beyond the higher prices paid by consumers as a result of publishers' exclusive rights,[36] the escalating costs of copyright include the colossal legal and judicial resources required to contain infringement on a network without some central gatekeeper (Netanel, 2003, pp. 24–30). All of these costs could be avoided, commentators argued, if we abandoned a copyright system focused on the enforcement of exclusive rights.

The question remained, of course, whether abandoning copyright would leave sufficient incentive for the *creation*—as opposed to the distribution—of works of authorship. Early proponents of a post-copyright paradigm contended that, even in the absence of any right of remuneration for the distribution of their works, artists would likely continue to create. Professor Ray Ku (2002), for example, pointed out that many artists—particularly musicians—have a financial incentive to record and distribute their works in order to enhance performance revenues; he also pointed to various other forms of payment, including online tipping, merchandising, and product placement advertising, that would bring revenues to creators of valuable expression (pp. 306–311). Even if these forms of compensation preserved incentives for musicians, however, they almost certainly would not address the need for economic incentives in other content markets, including movies, books, and software. In a nod to this reality, Ku suggested that if his alternatives proved inadequate to spur artistic creation, some form of levy system with proceeds distributed to authors would be preferable to the existing copyright regime of exclusive economic rights (pp. 311–315).[37]

Since Ku's 2002 article, two prominent legal scholars have taken up the call for a statutory levy to replace existing copyright law in the peer-to-peer context. In separate proposals, Professors Neil Netanel (2003) and William Fisher (2004) each advocate a tax on technology, with the proceeds to be distributed among copyright holders whose works are traded and copied by noncommercial users. While the two plans differ quite substantially in their details,[38] they share a common philosophy: that society would benefit if we liberated content from profit-minded copyright holders and allowed it to percolate freely through our culture, while at the same time providing a financial reward to content creators based on the relative value of their works.

The levy proposals have received a good deal of attention from the press, policymakers, and academics, and with good reason. A levy offers substantial advantages over the existing rights-based system: with a license to do as they wish with creative expression, consumers can enjoy it, transform it, reinterpret it, and proliferate it without having to answer to copyright holders. The values that copyright was designed to promote—speech, artistic creation, and learning—would arguably benefit from a liberation of creative content. By shifting the balance from copyright enforcement to use with compensation, moreover, the proposals avoid the costs associated with policing and enforcement of copyrights.

At the same time, these schemes come with some serious drawbacks. To operate effectively, each of the levy proposals would require a bulky administrative regime charged with setting rates, monitoring use of content, and distributing proceeds to copyright holders. While such a regime is certainly possible, it would not come cheaply, making the levy proponents' cost-cutting assertions at least debatable. These administrative costs, moreover, would continue as long as the levy remained in effect, in contrast to copyright enforcement costs, which could well decrease if existing legal efforts managed to stanch the tide of file sharing.

Beyond their overall cost, the levy proposals reflect an abandonment of the historical assumption that markets, rather than regulators, do the best job of setting prices and guiding production decisions. The plans abandon this assumption in two ways. First, they seize from content distributors the right to market their products under price structures that they view as optimal to derive value from their works. While this may ultimately prove better for the public (which, after all, will not be hobbled by monopolistic pricing decisions), it could instead reduce distributor revenues in a way that negatively affects incentives in content markets. Second, the plans tamper with technology markets by applying a standardized, across-the-board tax on users of technology products. As an economic matter, such taxes undoubtedly distort purchasing decisions, at least on the margins. Some people who value their broadband service for uses other than streaming or copying entertainment content, for example, may well abandon broadband rather than pay a tax on features that they do not use. Under either of the levy plans, those who do not personally benefit—or who benefit only slightly—from the use of peer-to-peer technologies would effectively subsidize college students and other heavy traders of copyrighted content.

B. Fortifying Copyright Law

At the same time that some commentators argue for the displacement of copyright law, others—primarily copyright holders—seek new and more powerful legal tools to protect their existing entitlements. While levy advocates argue for a complete liberation of creative content, copyright holders press for legislation that makes it harder for people to use their works in unauthorized ways. In 2002, for example, a group of copyright holders lobbied for passage of a bill that would mandate standard security technology in virtually every computer-related product. The Consumer Broadband and Digital Television Promotion Act (2002), proposed by Senator Fritz Hollings, would have required every digital media device distributed in the United States to include standard security technology to protect against the unauthorized use of copyrighted works (§§ 3, 5). Digital media devices include any hardware or software that retrieves, transfers, converts, or reproduces copyrighted content. Supporters of this bill, in other words, sought legal affirmation of copyright holders' near-absolute right to determine the usage of their protected works.

There are several problems with the Hollings proposal and others like it. First, on a purely pragmatic level, such standardized digital rights management technology would arguably have little success in stemming the transfer of copyrighted content. Experts agree that "no technological barrier can ultimately prevail over determined hackers who have physical access to the encrypted items, including … mass-marketed CDs and DVDs, personal computers, consumer electronic devices, and software embedded in those items" (Netanel, 2003, pp. 9–10). All it takes is one unencrypted file somewhere to appear on a file sharing network, and the expense of creating and enforcing a technological standard will be proven a waste.

Beyond these practicalities, a federally mandated security standard would raise serious legal and economic concerns. Providing airtight copyrights would arguably interfere with the public's right to use copyrighted works in some unauthorized but legitimate ways. As an economic matter, imposing a single technological standard upon vast sectors of the national economy would burden those sectors economically and technologically. Competition over the "best" encryption standard would decline, and innovators in the consumer electronics, computer, software, and related industries would endure the cost of the new features, all for the benefit of copyright holders. For these reasons and others, commentators condemned the Hollings bill, and it never passed out of committee.

C. A Return to Acts-Based Copyright Law

In contrast to an approach that either supplants copyright law or tinkers with its balance, some commentators advocate a different response to the peer-to-peer crisis: a renewed focus on direct infringers, particularly those who upload copyrighted content onto file sharing networks (Dogan, 2003; Lemley & Reese, 2004). In this view, legal efforts should aim to reduce infringing behavior, rather than disabling distribution technologies or imposing a broad-based tax. Like the other proposals discussed above, this one comes with its own benefits and costs.

On the plus side, an infringement-oriented strategy preserves the *Sony* principle that copyright holders should not have veto power over the evolution of new technologies. If copyright holders manage to contain infringement on peer-to-peer networks through legal efforts aimed at infringers, then policymakers and judges will feel less compulsion to tamper with peer-to-peer technologies themselves. And "containing" infringement does not mean eliminating it—copyright holders have never enjoyed airtight control over uses of their works, and their incentives will remain intact if they can capture the core markets for their content. On a related point, an enforcement strategy that targets direct infringers leaves room for fair use and other unauthorized but legitimate uses of copyrighted content.

As an economic matter, moreover, one of the advantages of the historical acts-based copyright regime was its assurance that those who derived the most value from copyrighted works funded their production. A move away from the acts-based system—whether through a liberalized standard for contributory infringement or through a real or effective tax on technology—inevitably shifts some of these production costs to people who would prefer to spend their money elsewhere. While a society may well decide in favor of such subsidization, the choice is undeniably a value-laden one, with uncertain consequences in the markets for both content and technology.

Finally, recent developments in legal online distribution support the view that an acts-based approach may help to drive consumers to authorized products. Over the past two years, scores of legitimate online music distributors have emerged, offering per-song downloads of copyrighted products. As consumers test these and other legal alternatives to peer-to-peer networks, they may find that the convenience and value of buying authorized content outweighs even the slight risk of legal action against them for engaging in unauthorized file trading. Indeed, some measures suggest that the recording industry has prospered despite the continued availability of peer-to-peer services, indicating that the direct infringement efforts may be working (Borland, 2004).

Despite these advantages, a direct infringement approach is no panacea. As the levy proponents point out, any strategy designed to squash unauthorized file sharing will interrupt something of a cultural revolution among certain groups, particularly teenagers and college students. Economically, moreover, existing copyright law does favor entrenched publishers in traditional industries; a status quo approach would do little, over the short term, to challenge their dominance.[39] Finally, a direct infringement approach comes at substantial social cost and with unproven benefits. Locating and prosecuting individual file sharers is a monumental task; while deterrence theory teaches that copyright holders need not pursue all of them to have some effect on behavior, any realistic chance of deterrence will require a sustained, expensive legal campaign, which may, at the end of the day, prove ineffective.

Conclusion

The tension between copyright law and peer-to-peer technology is neither simply stated nor easily resolved. Peer-to-peer networks challenge copyright law because they make it more difficult for copyright holders to capture the value generated by their works. If copyright holders prove unable, over the long term, to recover that value, the peer-to-peer revolution could ultimately harm incentives for the creation and distribution of software, movies, music, books, and other expressive works. For that reason, judges and policymakers are struggling with how best to adapt copyright law to this new technology. They have several choices. They can shut down the technology, tax it, go after the infringers, or let the information be free. Which of these choices is preferable depends, in part, on guesswork, and in part on judgments of the relative importance of such values as economic incentives, technological freedom, and expressive choice. This chapter has attempted to provide some of the basic legal tools that can help individuals to form their own opinions on the peer-to-peer crisis, in the hope that it will contribute to a more informed public debate.

References

A&M Records, Inc. v. Napster, Inc., 114 F. Supp. 2d 896, 904 & n.8 (N.D. Cal. 2000).

A&M Records, Inc. v. Napster, Inc., 239 F.3d 1004, 1014-1019 (9th Cir. 2001).

A&M Records, Inc. v. Napster, Inc., 284 F.3d 1091 (9th Cir. 2002).

Borland, J. (2004). CD Shipments Surge After Lean Years. *CNET News*, Oct. 11, 2004. <http://news.com.com/CD+shipments+surge+after+lean+years/2100-1027_3-5419640.html> (downloaded Nov. 18, 2004).

Computer Assoc. Intern'l, Inc. v. Altai, Inc., 982 F.2d 693 (2d Cir. 1992).

Consumer Broadband and Digital Television Promotion Act, S. 2048, 107th Cong., 2d Sess. (2002).

Dogan, S. (2001). Is Napster a VCR? The implications of Sony v. Universal City Studios for Napster and other Internet-related actors. *Hastings Law Journal, 52,* 939.

Dogan, S. (2002). Infringement once removed: The perils of hyperlinking to infringing content. *Iowa Law Review, 87*, 829.

Dogan, S. (2003). Code Versus the Common Law. *Journal on Telecommunications & High Technology Law, 2,* 73.

Dreamland Ball Room, Inc. v. Shapiro, Bernstein & Co., 36 F.2d 354 (7th Cir. 1929).

Fisher, W. (2004). *Promises to keep: Technology, law, and the future of entertainment*. Palo Alto, CA: Stanford University Press.

Gershwin Publ'g Corp. v. Columbia Artists Mgmt., Inc., 443 F.2d 1159, 1162 (2d Cir. 1971).

Goldstein, P. (2d ed. 1996 & 2002 Supp.). *Copyright*. Boston: Little, Brown.

Harper & Row Publishers, Inc. v. Nation Enterprises, 471 U.S. 539, 566 (1985).

Irving Berlin, Inc. v. Daigle, 26 F.2d 149, 150 (E.D. La. 1928).

Ku, R.S. (2002). The creative destruction of copyright: Napster and the new economics of digital technology. *University of Chicago Law Review, 69*, 263.

Lemley, M., & Reese, A. (2004). Reducing copyright infringement without restricting innovation. *Stanford Law Review, 56*.

Lichtman, D., & Landes, W. (2003). Indirect liability for copyright infringement: An economic perspective. *Harvard Journal of Law & Technology, 16*, 395, 408.

Litman, J. (2001). *Digital copyright*. New York: Prometheus Books.

Lunney, G., Jr. (2001). The death of copyright: Digital technology, private copying, and the Digital Millennium Copyright Act. *Virginia Law Review, 87*, 813.

MAI Sys. Corp. v. Peak Computer, Inc., 991 F.2d 511, 518-19 (9th Cir. 1993).

Menell, P. (2002–2003). Envisioning copyright's digital future. *New York Law School Law Review, 46*, 63.

Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 259 F. Supp.2d 1029, 1038 (C.D. Cal. 2003).

Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 380 F.3d 1154, 1160-67 (9th Cir. 2004).

Netanel, N. (2003). Impose a noncommercial use levy to allow free P2P file sharing. *Harvard Journal of Law and Technology, 17*, 1.

Shapiro, Bernstein & Co. v. H.L. Green Co., 316 F.2d 304, 316 (2d Cir. 1963).

Sony v. Universal City Studios (1984) 464 U.S. 1417 (1984).

Stenograph LLC v. Bossard Assoc., Inc., 144 F.3d 96, 100 (D.C. Cir. 1998).

Strahilevitz, L.J. (2003). Charismatic code, social norms, and the emergence of cooperation on the file-swapping networks. *Virginia Law Review, 89*, 505–596.

United States Copyright Act, 17 U.S.C. §§ 101-1332 (2004). §§ 106, 107, 111, 119

White-Smith Music Publ'g Co., Inc. v. Apollo Co., 209 U.S. 1, 15-18 (1908).

Wingfield, N., & Smith, E. (2003, September 9). Record industry files suit against 261 music uploaders; Move may alienate customers. *Wall Street Journal*, B1.

Wu, T. (2003). When code isn't law. *Virginia Law Review, 89*, 679, 685.

Endnotes

1 The economic orientation of United States copyright law contrasts with the law in much of the rest of the world, including continental Europe and Latin America, where copyright is viewed as a natural right of the author, rather than in utilitarian terms. In France, for example, the exclusive economic rights to copy, distribute, and so forth are a mere subset of authors' rights, which also include the right to control certain uses that might harm the integrity of the work or harm the author's reputation. United States law offers only limited protection for such "moral rights" of authors, opting instead for an alienable, tradable bundle of economic rights that will theoretically enable widespread licensing and distribution of copyrighted works.

2 The fair use provision in the Copyright Act appears at 17 U.S.C. § 107. Rather than an exhaustive listing of classes of fair uses, the Copyright Act provides a series of factors that courts should consider in deciding whether to allow a particular use. Courts are instructed to consider "(1) the purpose and character of the use," including its commercial or noncommercial nature and whether the defendant has transformed the expression in some speech-enhancing way; "(2) the nature of the copyrighted work," with more expressive works entitled to stronger protection against unauthorized use; "(3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and (4) the effect of the use upon the potential market for or value of the copyrighted work" (17 U.S.C. § 107).

3 An infringer is one who, without the authority of the copyright owner, "do[es] or authorize[s] any of the following: (1) to reproduce the copy-

righted work in copies of phonorecords; (2) to prepare derivative works based upon the copyrighted work; (3) to distribute copies or phonorecords of the copyrighted work to the public by sale or other transfer of ownership; (4) in the case of literary, musical, dramatic, and choreographic works, pantomimes, and motion pictures and other audiovisual works, to perform the copyrighted work publicly; (5) in the case of literary, musical, dramatic, and choreographic works, pantomimes, and pictorial, graphic, or sculptural works, including the individual images of a motion picture or other audiovisual work, to display the copyrighted work publicly; and (6) in the case of sound recordings, to perform the copyrighted work publicly by means of a digital audio transmission" (17 U.S.C. § 106).

4 Some of these uses fell within the protection of the fair use doctrine of copyright law, while others may have technically constituted infringement but were not worth most copyright holders' while to pursue.

5 Professor Tim Wu explored this gatekeeper concept in a recent article, contending that "the copyright regime has achieved its goals through enforcement against specialized intermediaries—those capable of distributing creative works on a massive scale" (Wu, 2003, p. 685).

6 To be sure, not all sharing displaces legitimate sales, because consumers frequently have an interest in purchasing authorized products from their source. Unauthorized copies of software, for example, come without customer support or documentation, and music arrives without fancy CD cases or cover art. These distinctions, however, do not differentiate file sharing from previous methods of unauthorized distribution, which also frequently lacked all of the trappings of the legitimate product.

7 This chapter focuses on the application of copyright law to peer-to-peer services. For a thorough introduction to the multitude of issues raised by applying copyright to digital technology, see Menell (2002–2003).

8 In the early years of software, there was much debate over whether copyright law applied to the medium because of its inherently functional nature. By the late 1970s, Congress had made clear its intent that copyright extend to software, but since then, the courts have struggled to distinguish between expression and function in the software context. As one court noted, "Thus far, many of the decisions in this area reflect the courts' attempt to fit the proverbial square peg in a round hole" (Computer Assoc. Intern'l, Inc. v. Altai, Inc., 1992). The details of that struggle are beyond the scope of this chapter, however, because courts universally agree that copyright law protects against the duplication and distribution of full copies of program code—the behavior at issue in peer-to-peer file sharing.

9 Congress extended the copyright term twice in recent decades, and each time it made its extensions at least partly retroactive. All works created

after January 1, 1978, enjoy the life plus seventy year term (17 U.S.C. § 302).

10 Before the 1976 Copyright Act, United States copyright law required that copyright holders publish, register, deposit, and give copyright notice on their works as a condition of legal protection. Copyright holders also had to request a renewal of their 28-year copyright, if they wished to enjoy rights for the full 56-year potential term. Today, copyright holders can receive the full term of protection without renewal or formalities, although registration and notice continue to confer some legal benefits.

11 The 1909 Copyright Act provided for a 28-year term, which could be renewed for another 28 years.

12 For a more detailed introduction to the history of contributory and vicarious liability and their application in the online context, see Dogan (2002).

13 The doctrine evolved in the so-called dance hall cases in which proprietors of dance halls benefited financially when bands performed popular copyrighted songs in their establishments (Dreamland Ball Room, Inc. v. Shapiro, Bernstein & Co., 1929).

14 In Gershwin Pub’g Co. v. Columbia Artists Mgmt., Inc. (1971), for example, the court imposed contributory and vicarious liability against an agent for infringing performers. In Irving Berlin, Inc. v. Daigle (1929), the court held that the fact that the defendant “operated and controlled the place of public entertainment, charging admission and so operating for a profit, establishes his liability for permitting and authorizing the unlicensed use of plaintiff’s musical compositions in and on the premises.”

15 “Time-shifting” is “recording a program [the consumer] cannot view as it is being televised and then watching it once at a later time” (Sony v. Universal City Studios, 1984, p. 421).

16 The music industry filed the first wave of such lawsuits in the fall of 2003 (Wingfield & Smith, 2003).

17 Both the trial court and the court of appeals wrote lengthy opinions in the *Napster* case. This discussion refers primarily to the court of appeals decision.

18 Both contributory infringement and vicarious liability require that an act of direct infringement occur, so in order to proceed with their claims against peer-to-peer providers, copyright holders have had to establish that people who traded their files without authorization were engaged in infringement (A&M Records, Inc. v. Napster, Inc., 2001, p. 1013 & n.2).

19 Unlike more decentralized peer-to-peer networks, Napster maintained a centralized index of music files that were available for trading. When a user

signed onto the network, Napster automatically updated the index with the names of that user's music files (p. 1012).

20 239 F.3d at 1014.

21 Courts have held that even RAM copies fall within the Copyright Act's definition of "copies" (MAI Sys. Corp. v. Peak Computer, Inc., 1993; Stenograph LLC v. Bossard Assoc., Inc., 1998). In the face of that authority, it seems clear that downloading MP3 files into hard drives or other storage devices constitutes the making of copies under the Act.

22 Napster also contended that fair use protected its users who were engaged in permissive trading of music files, that is, trading with the copyright owner's permission. This was a curious argument because such use would be a licensed use and presumptively noninfringing. The Ninth Circuit found it irrelevant not on that basis but because the plaintiffs were not challenging these particular uses of the Napster service (p. 1014).

23 The Ninth Circuit's characterization of file sharing as "commercial," for example, arguably stretches the meaning of that term as intended by the Supreme Court in *Sony*. Nonetheless, because fair use analysis involves a subjective balancing of four different factors, the court could have reached the same result by calling the behavior noncommercial, but noting its severe negative impact on the plaintiffs' potential markets. The Supreme Court has described market impact as "undoubtedly the single most important element of fair use" (Harper & Row Publishers, Inc. v. Nation Enterprises, 1985, p. 566).

24 As the trial court noted, whether defendant viewed the promotion of new artists as a genuine goal or a smokescreen remains ambiguous (*A&M Records, Inc. v. Napster* [2000], p. 904 & n.8). Early versions of the program, for example, touted the ease with which users could obtain popular music without "wading through page after page of unknown artists" (p. 904).

25 The Ninth Circuit took issue with the trial court's focus on current uses: "We depart from the reasoning of the district court that Napster failed to demonstrate that its system is capable of commercially significant noninfringing uses.… The district court improperly confined the use analysis to current uses, ignoring the system's capabilities" (p. 1021).

26 I have proposed one possible approach to this problem (Dogan, 2001). Focusing on the objectives of the *Sony* Court, I suggest that courts evaluating the applicability of the staple article of commerce doctrine should balance the interests of copyright holders and technology providers, rather than adopting an all-or-nothing approach. If a technology provider's continuing relationship with its users make it feasible to reduce infringe-

ment without substantially interfering with noninfringing uses, I suggest that liability is appropriate, because imposing liability would not give copyright holders leverage over unrelated markets. Even if such differentiation is not possible, I suggest that liability is appropriate if the noninfringing uses would not have supported the development of the product or service, because in such circumstances the technology market depends on copyright infringement, rather than being independent of it (pp. 949–958).

27 Several passages in the *Napster* opinion support this interpretation. The court, for example, repeatedly makes "a clear distinction between the architecture of the Napster system and Napster's conduct in relation to the operational capacity of the system" (p. 1020). Elsewhere, it defines Napster's liability with reference to its ability to purge content from its system: "We agree that if a computer system operator learns of specific infringing material available on his system and fails to purge such material from the system, the operator knows of and contributes to direct infringement" (p. 1021). In finally stating its conclusion on contributory infringement, the court pairs Napster's knowledge with its ability to act: "The record supports the district court's finding that Napster has *actual* knowledge that *specific* infringing material is available using its system, that it could block access to the system by suppliers of the infringing material, and that it failed to remove the material" (p. 1022).

28 In addressing the claim, the court first noted that *Sony* was irrelevant to vicarious liability, because the case before the Supreme Court had involved only contributory infringement (pp. 1022–1023). Vicarious liability, in any event, would not fit well into the Ninth Circuit's knowledge-based reading of *Sony*. Under that reading, *Sony* counsels only against imputing knowledge of how a technology is used by its purchasers. Vicarious liability, however, does not require knowledge, so a rule that prohibits imputing knowledge would have no impact on a vicarious liability claim.

29 In the wake of the Ninth Circuit's decision, the parties battled over the relative responsibilities of Napster and the music plaintiffs in identifying particular infringing file names. The Ninth Circuit ultimately upheld a fairly restrictive injunction that placed much of the burden on Napster (A&M Records, Inc. v. Napster, Inc., 2002).

30 For an insightful introduction to the variations in these decentralized networks, see Wu (2003).

31 Because the software in Grokster did not limit itself to MP3 files but allowed the exchange of movies and other content files, the plaintiffs in the case included movie studios as well as recording labels and music publishers.

32 The court found it "undisputed that there are substantial noninfringing uses for Defendants' software—e.g., distributing movie trailers, free songs or other copyrighted works; using the software in countries where it is legal; or sharing the works of Shakespeare" (Metro-Goldwyn-Mayer Studios, Inc. v. Grokster, Ltd., 2003, p. 1035).

33 Because of the preliminary stage of the proceedings, the district court did not reach any final conclusions, but ruled only that the recording industry had shown a likelihood of succeeding in its claims against the defendants.

34 Having affirmed the preliminary injunction based on the contributory infringement claim, Posner did not reach the question of Aimster's vicarious liability. He suggested, however, that he might limit vicarious liability to more classic principal-agent type claims (p. 654).

35 In *Religious Tech. Ctr. v. Netcom On-Line Communication Servs., Inc.* (1995), for example, the court held that an Internet service provider could face liability for contributory infringement if it failed to remove infringing content from its servers, after receiving notice from a copyright holder. Congress later codified this rule, along with a set of "safe harbor" exemptions designed to protect service providers from copyright liability if they satisfied certain conditions, including the adoption of policies to help copyright holders to enforce their rights (17 U.S.C. § 512; Dogan, 2003).

36 As a variation on this point, some commentators have criticized copyright law as promoting the interests of a powerful set of media conglomerates, rather than supporting artists themselves (Litman, 2001).

37 Lunney (2001) had previously suggested such an approach as well.

38 Netanel, for example, would tax any technology whose value is substantially enhanced by file sharing, while Fisher would focus on technologies most closely related to trading and copying content files—broadband access, CD burners, and the like. Netanel would allow only noncommercial file swapping, while Fisher would insulate all forms of online dissemination. A fuller treatment of the levy proposals is beyond the scope of this chapter, but the Netanel and Fisher proposals offer extremely thorough and grounded descriptions of their rationale and the particulars of their plans.

39 Arguably, the emergence of competing distribution platforms—including legal distribution—is beginning to threaten the hegemony of at least the music industry. While unrestrained peer-to-peer technology would quicken the pace of their demise, it seems likely that over time, the increased range of distributional offerings will put competitive pressure on traditional publishers.

Section III

P2P Domain Proliferation: Perspectives and Influences of Peer Concepts on Collaboration, Web Services and Grid Computing

Chapter IX

Personal Peer-to-Peer Collaboration Based on Shared Objects

Werner Geyer, IBM T.J. Watson Research Center, USA

Juergen Vogel, University of Mannheim, Germany

Li-Te Cheng, IBM T.J. Watson Research Center, USA

Michael J. Muller, IBM T.J. Watson Research Center, USA

Abstract

This chapter describes the design and system architecture of a new peer-to-peer technology for personal collaboration based on the notion of shared objects. This approach allows users to collaborate in a rich but lightweight manner by organizing different types of shared artifacts into semistructured activities with dynamic membership, hierarchical object-relationships, and synchronous and asynchronous collaboration. This approach goes beyond simple peer-to-peer file sharing. It requires data replication and sophisticated consistency control to keep data consistent in a blended synchronous and asynchronous environment. The authors present the *design of a prototype system and then develop an enhanced consistency control algorithm that is tailored to the needs of this new environment. Simulation results demonstrate the performance of this approach. This chapter aims at informing researchers about both the potential and the complexity of more advanced peer-to-peer applications and shows the trade-offs in the design and implementation of these systems.*

Introduction

Peer-to-peer applications are often equated with file sharing software where users who are part of a peer-to-peer network grant access to files stored on their local computer. While being very popular, file sharing is only one type of application that can be built using peer-to-peer technologies. Peer-to-peer is not file sharing but rather a programming model in which each client in the network also becomes a server at the same time. This model can be used to implement much richer collaborative applications than file sharing.

This chapter describes a new peer-to-peer application for personal collaboration that sits midway between the informality of e-mail and the formality of shared workspaces. E-mail and other ad hoc collaboration systems are typically lightweight and flexible, but build up an unmanageable clutter of copied objects. At the other extreme, shared workspaces provide formal, structured collaboration, but are too heavyweight for users to set up. To bridge this gap between the ad hoc and formal, our approach introduces the notion of "object-centric sharing," where users collaborate in a lightweight manner but aggregate and organize different types of shared artifacts into semistructured activities with dynamic membership, hierarchical object relationships, as well as synchronous and asynchronous collaboration. Based on these principles, we have designed and built a peer-to-peer prototype system that does not burden the user with the overhead of manually creating shared workspaces or setting up conferences on centralized servers; it provides rich collaboration combined with the lightweight characteristics of e-mail by leveraging local processing power and storage and through direct peer-to-peer communication.

This chapter focuses on the peer-to-peer architecture and implementation of this system and the technical challenges that our design philosophy poses. Keeping replicated data consistent in an architecture that supports real-time and asynchronous collaboration at the same time is not trivial, and relatively little research has been done in addressing this problem. Our approach enhances a popular consistency control algorithm, which had been originally designed for real-time collaboration. We present and discuss various strategies for how this algorithm also can be used to maintain consistency in asynchronous modes or when people are working off-line. While our simulation results indicate that this approach works well, it has also some shortcomings. We discuss trade-offs in the architectural design of this system and suggest future enhancements and research opportunities.

Background

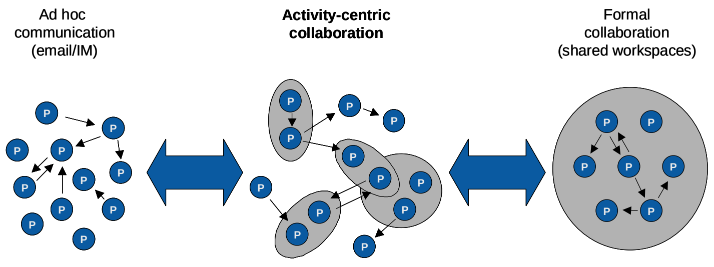
Bridging the Gap

Collaborative processes very often emerge from unstructured ad hoc communication activities to more structured types of formal collaboration (Bernstein, 2000). Groupware has focused on the two extremes of this continuum but neglected many of the possible stages in-between. E-mail at one extreme of this continuum can be considered as today's ultimate ad hoc communication support system. Recent studies indicate that e-mail is the place where collaboration emerges (Ducheneaut & Bellotti, 2001b; Whittaker & Sidner, 1996). A variety of e-mail uses are reported in the literature such as information management, document management and sharing, task management, and meeting management. Whittaker and Sidner (1996) coined the term "e-mail overload" as the phenomenon of e-mail being used for additional functions other than communicating.

While e-mail is extremely flexible, it also requires the user to do a lot of work, such as manually keeping track of the organizational process; users are mostly left unassisted with the contextual sense making of all the information contained in their cluttered inboxes. Despite these drawbacks, people keep on using e-mail instead of managing their collaborative processes with special purpose groupware systems such as shared team workspaces, decision-support systems, or meeting management systems. While these systems provide more structure and richer support for collaboration, people often shy away from using them because e-mail is readily available, always on, often the focus of attention, ad hoc, and does not require tedious setup procedures.

Little work has been done to offer richer collaboration in e-mail and to help support the progression of collaboration from ad hoc communication to more structured types of collaboration that are already supported in many special purpose groupware systems. In our research (Geyer & Cheng, 2002; Geyer, Vogel, Cheng, & Muller, 2003), we are investigating technologies for activity-centric collaboration that can help bridge this gap (see Figure 1).

Figure1. From ad hoc communication to formal collaboration


Ad hoc communication (email/IM)
Activity-centric collaboration
Formal collaboration (shared workspaces)
P

Design Approach

While e-mail is very good in supporting the ad hoc nature of collaboration and dynamic membership (Ducheneaut & Bellotti, 2001a), it is not very good in preserving the context and structure during a conversation; related messages and attached documents are typically scattered or buried in the inbox and they are hard to find. Moreover, e-mail does not support real sharing of content, let alone real-time collaboration. In order to support those aspects of a work activity, people have to "leave their inbox" and use other tools (shared workspaces, conferencing applications, etc.) that produce new artifacts that are related to the original work activity. When they do this, they are disconnected from their collaborative environment, because those artifacts reside somewhere else on the desktop, in the file system, or on a remote server. Users are forced to make a choice between ease of collaboration (but with resources *external* to the collaboration environment) versus ease of resource access (but with collaboration *external* to the resource management environment). The design of our system was mainly driven by the desire to combine the lightweight and ad hoc characteristics of e-mail and the rich support for sharing and structure in shared workspace systems.

Object-Centric Sharing

Traditional shared workspaces typically entail a lot of management overhead and are far from being lightweight or ad hoc. They are "place-based," that is, users first have to create a persistent shared online environment (a "place"), assign access rights, and then put content into that place in order to be able to share it. They are based on the assumption that "the team" already exists and that the purpose of collaboration is well known. However, when collaboration starts off, this is often not the case, and setting up a place can seem to be artificial if not obstructive at this stage. In our research (e.g., Cohen, Cash, & Muller, 2000; Whittaker & Sidner, 1996), people often prefer to think in terms of with whom to share and what to share. Also, collaboration in these early stages might be very short term and instantaneous and involve only little amounts of data to be shared, for example, exchanging one or more files, setting up a meeting agenda with people, or jointly annotating a document. These activities might or might not become part of a larger collaborative work process. However, people usually do not create heavyweight shared workspaces to perform these lightweight tasks.

Unlike providing one persistent place for sharing multiple pieces of information, our paradigm is rather "object-centric" or "content-centric," which is very similar to Dourish, Edwards, Lamarca, and Salisbury's (1999) notion of "placeless" documents. In this approach, sharing becomes a property of the content itself, that is, content is collaboration-aware. We use the term "shared object" for one shared piece of persistent information. Shared objects support membership, provide object-level awareness, and enable group communication. In other words, they define a set of people who are allowed to access the information, they indicate who is currently looking at the content, and they allow sending or broadcasting of data to members of the object.

Conversational Structure

In our approach, shared objects are building blocks of collaboration. We allow users to combine and aggregate them into hierarchical structures as their collaboration evolves. We call a set of related shared objects an *activity thread*, representing the context of an evolving collaborative activity. Aggregating objects allows people to add structure to their collaboration. We see this structure being defined by their ongoing conversation, that is, each object added to an existing object can be considered as a "reply" to the previous one. While this approach is similar to threads in e-mail (Kerr, 2003), discussion databases (Schoberth, Preece, & Heinzl, 2003; Yates & Orlikowski, 2003), or "thrasks" (Bellotti, Ducheneaut, Howard, & Smith, 2003), it is much richer because (1) objects are shared unlike in e-mail or thrasks; (2) activity threads may contain different types of objects, not only messages like in e-mail or discussion databases; (3) all objects are equal, unlike in e-mail or discussion databases where attachments are subordinates contained in the message; (4) membership is dynamic and may differ within an activity thread from object to object unlike in discussion databases—and membership can be redefined after an object has been created, unlike in e-mail; and (5) objects support synchronous collaboration through built-in real-time notifications and provide rich awareness information on who is currently working on what.

In contrast to shared workspaces, we intentionally do not provide an explicit object as a structural container for a collaborative activity. Each individual shared object can serve as a parent object in a hierarchical data structure, and can in this way act as a kind of a container. Each object can thus be considered as a "seed" for collaboration that either decays or grows to more structured forms with the structure being defined as people collaborate.

Our design also does not deliberately make a distinction between asynchronous and synchronous types of collaboration. If other people are present at the time of accessing an object, they can work synchronously; if not, work is asynchronous. From a more technical perspective, objects can be considered as an "infinite," persistent (conferencing) session bounded only by the lifetime of the object. Modifications to the object are broadcast to the members of that object if they are online.

Related Work

Our notion of activity-centric collaboration based on shared objects has been inspired by previous work in this area. This includes a variety of studies on e-mail and activities, collaborative improvements to e-mail, and "peer-to-peer-like" systems featuring replicated or synchronous shared objects. The following paragraphs cover related work mostly from an application perspective. Work on consistency control and peer-to-peer systems as it relates to our approach will be discussed as we present the architecture and implementation of the system.

The use of e-mail has been widely studied and research indicates that e-mail is a place where collaboration emerges (e.g., Ducheneaut & Bellotti, 2001a; Ducheneaut & Bellotti, 2001b; Mackay, 1988; Whittaker & Sidner, 1996). Ducheneaut and Bellotti (2001a) report on dynamic membership in e-mail work activities and on informal activities evolve to more formal ones, confirming Bernstein's (2000) description of emerging group processes.

A number of recent collaboration systems illustrate concepts similar to activity threads. Rohall, Gruen, Moody, and Kellerman (2001) show how automatic analysis of subject headers and reply relationships in different e-mails can group them into a coherent thread of messages.

Bellotti, Ducheneaut, Howard, and Smith (2003) introduce the notion of a "thrask" as a means for better organizing e-mail activities. Thrasks are threaded task-oriented collections that contain different types of artifacts such as messages, hyperlinks, and attachments and treat such content at an equal level. Along the same lines, Kaptelinin (2003) presents a system that helps users organize resources into higher-level activities ("project-related pools"). The system attempts to include not just e-mail, but also any desktop application. While all these approaches are similar to our use of activity threads, they are not shared and they lack any awareness of how others are manipulating the artifacts.

A variety of collaborative systems implement replicated shared objects and "collaborative building blocks" similar to our prototype. One example is Groove (Groove Networks, n.d.), a peer-to-peer system that features a large suite of collaborative tools (chat, calendar, whiteboard, etc.) and e-mail integration. However, Groove is workspace-centric (although creation of shared workspaces is quick) and collaboration is centered on the tools placed in the workspace (e.g., all shared files appear in the shared files tool), except for persistent chat which appears across all tools. In contrast, our system focuses collaboration around artifacts, which can be organized in a single connected hierarchy of different types. While Groove's design philosophy is different from ours, the architecture is very alike. Their system has to deal with problems similar to the ones described in the main section of this chapter. However, we have no technical information available to compare the two approaches.

Another example is Eudora's Sharing Protocol (Eudora Sharing Protocol, n.d.), which offers a replicated peer-to-peer shared file architecture completely based on e-mail and leverages special MIME headers. Kubi Spaces (Kubi Software, 2003) also offers replicated shared objects, with a richer suite of object types, including to-dos, contacts, and timelines. Microsoft Outlook (Microsoft Office, 2003) has a server-based "shared e-mail folder" capability. Lotus Notes (Lotus Notes, n.d.) offers a replicated object architecture, although typically e-mail objects are copies, not shared amongst different users. A powerful benefit of these "shared e-mail solutions" is that no additional infrastructure beyond e-mail is needed. However, synchronization only occurs on a triggered refresh interval and depends on a nonreal-time message delivery between intermediary e-mail servers. Thus, collaboration is entirely asynchronous (e.g., users cannot work simultaneously on a whiteboard in real time) with no real-time presence awareness.

While our work focuses on peer-to-peer sharing of objects in the context of collaboration around e-mail, there is an entire class of personal peer-to-peer collaborative applications that do not involve e-mail or shared objects. These are ad hoc, synchronous communication systems using a peer-to-peer networking infrastructure. Such systems enable lightweight collaboration (i.e., no server required to establish a shared session) by leveraging peer nodes to communicate point-to-point and collaborate in purely synchronous sessions. Collaboration often involves nonpersistent shared artifacts. Examples include shared whiteboards and chats found in conferencing applications like Microsoft NetMeeting (Microsoft NetMeeting, 2003), voice-over-IP telephony applications like Skype (Skype, 2003), and shared editors like SubEthaEdit (SubEthaEdit, n.d.). Skype and SubEthaEdit are particularly noteworthy with simple user interfaces that hide the complexities of the underlying peer-to-peer networking protocols. Skype uses a proprietary protocol, whereas SubEthaEdit uses Apple Rendezvous, also known as Zero Configuration Networking, which is being standardized by the IETF (Zero Configuration Networking, 2003). However, from the point of view of our "building block" approach of object-centric sharing, these applications would be treated as additional shared objects, with the added option to persist themselves for asynchronous use. For example, we do implement chat and a shared whiteboard as shared objects, and we later describe the challenges of replicating their events for live participants as well as for asynchronous access or late comers.

A Peer-to-Peer Collaborative System Based on Shared Objects

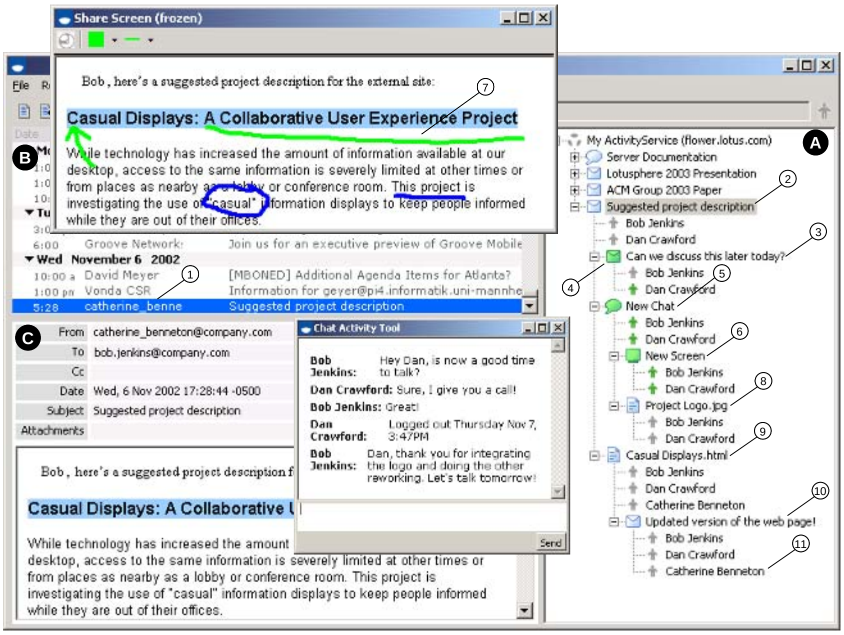
User Interface

The user interface to our peer-to-peer prototype system is integrated into an e-mail client. The client supports sharing of five types of objects: message, chat transcript, file, annotated screen shot, and to-do item. These objects are managed through a simple tree-like user interface that is contained in the right pane (A) in Figure 2. Each "branch" in that tree represents an activity thread.

Users interact with shared objects by right-clicking on the nodes of the tree which pops up a context menu. Users can create new objects, delete objects, add and remove members, and so forth. Our client supports POP3 e-mail messaging: The upper left pane is the inbox (B) and the lower left pane a message viewer (C). In the following, we use a scenario to illustrate how shared objects as building blocks can be used to collaborate in an activity that starts from an e-mail message. Please note that the activity thread displayed in Figure 2 is just a snapshot at the end of an activity from the perspective of one of the actors (Bob); the thread is built dynamically as the actors collaborate.

Bob is a project lead and he works with Dan on a project on "Casual Displays." Catherine is a Web designer in their company who is responsible for the external Web site. Bob receives an e-mail from Catherine containing a draft for a project description that she would like to put on their external Web site (1). She wants some feedback from Bob. Before getting back to her, Bob wants to discuss the design of that Web page with Dan. Instead of forwarding the message to Dan via e-mail, Bob decides to start a new

Figure 2. Prototype user interface: (A) activity thread pane, (B) e-mail inbox pane, (C) e-mail viewer pane



Bob , here's a suggested project description for the external site:

Casual Displays: A Collaborative User Experience Project

While technology has increased the amount of information available at our desktop, access to the same information is severely limited at other times or from places as nearby as a lobby or conference room. This project is investigating the use of "casual" information displays to keep people informed while they are out of their offices.

```
6:00      Groove Network:   Join us for an executive preview of Groove Mobile
▼Wed  November 6  2002
10:00 a   David Meyer       [MBONED] Additional Agenda Items for Atlanta?
1:00 pn   Vonda CSR         Information for geyer@pi4.informatik.uni-mannhe
5:28      catherine_benne   Suggested project description
```

From: catherine_benneton@company.com
To: bob.jenkins@company.com
Cc:
Date: Wed, 6 Nov 2002 17:28:44 -0500
Subject: Suggested project description
Attachments:

Bob , here's a suggested project description f

Casual Displays: A Collaborative U

While technology has increased the amount
desktop, access to the same information is severely limited at other times or from places as nearby as a lobby or conference room. This project is investigating the use of "casual" information displays to keep people informed while they are out of their offices.

Bob Jenkins: Hey Dan, is now a good time to talk?
Dan Crawford: Sure, I give you a call!
Bob Jenkins: Great!
Dan Crawford: Logged out Thursday Nov 7, 3:47PM
Bob Jenkins: Dan, thank you for integrating the logo and doing the other reworking. Let's talk tomorrow!

```
My ActivityService (flower.lotus.com)
  ⊞ Server Documentation
  ⊞ Lotusphere 2003 Presentation
  ⊞ ACM Group 2003 Paper
  ⊟ Suggested project description
      Bob Jenkins
      Dan Crawford
    ⊟ Can we discuss this later today?
        Bob Jenkins
        Dan Crawford
    ⊟ New Chat
        Bob Jenkins
        Dan Crawford
      ⊟ New Screen
          Bob Jenkins
          Dan Crawford
      ⊟ Project Logo.jpg
          Bob Jenkins
          Dan Crawford
    ⊟ Casual Displays.html
        Bob Jenkins
        Dan Crawford
        Catherine Benneton
      ⊟ Updated version of the web page!
          Bob Jenkins
          Dan Crawford
          Catherine Benneton
```

activity by creating a shared object based on this message. He right-clicks on the original message in his inbox, selects "share," enters Dan's e-mail address, and hits "Share." A new shared message object (with Bob and Dan as members) shows up in Bob's activity tree in the right window pane (2). Bob right-clicks on the shared object and adds a new shared message to the initial one because he wants to let Dan know that he would like to discuss this with him. Bob's message shows up as a child (reply) to the initial message similarly to a newsgroup thread (3).

A few hours later, Dan returns to his desktop, which is running the client, and notices Bob's newly created shared messages. He opens one message and while he is reading it, Bob sees that Dan is looking at the messages because the shared object is lit green along with Dan's name underneath the object (4). Bob takes this as an opportunity to begin a discussion with Dan within the context of the shared object. He right-clicks on the initial message and adds a chat object to this activity (5). A chat window pops up on Dan's desktop and they chat. In their chat conversation, Bob and Dan continue talking about the Web page over the phone. At some point during *the discussion, Bob wants to show directly how to change the Web page. He right-clicks on the chat object in his activity tree and adds a shared screen object (6). A transparent window allows Bob to select and "screen scrape" any region on his desktop. He freezes the transparent window over Catherine's draft Web page. The screen shot pops up on Dan's desktop. Bob and Dan begin annotating the Web page in real time like a shared whiteboard (7). As they discuss a few changes, Bob is asking Dan to integrate a project logo into the Web page. Dan agrees but is pressured now to run to another meeting. He says good-bye to Bob and tells him that he will check with him the next day. Dan closes all his windows and as he leaves, his name turns gray throughout all of his shared objects displayed on Bob's client.*

Now alone, Bob continues annotating the Web page. He also types in a few lines for Dan in the chat window before closing it. He then right-clicks on the chat object and creates a new shared file object. He picks the logo file from his local file system and the file object becomes part of Bob and Dan's activity thread (8). Bob closes all windows and leaves. Next morning when Dan returns to his office, he finds Bob's additional annotations, his chat message, and the project logo file. He starts working on the Web page and a few hours later, he puts the reworked page into the activity thread as a shared file object (9) and adds a message with some comments (10). He also shares these two objects with Catherine (11) so that she can download and deploy the newly revised Web page and logo.

This scenario demonstrates how our prototype moves seamlessly and effortlessly back and forth from private to public information, and from asynchronous to synchronous real-time collaboration, without manually creating a shared workspace or setting up a meeting. Collaboration starts off with a single shared object and evolves into a multi-object activity, which is structured by a dynamic group of participants as they create and add new shared objects. An activity thread provides the conversational context and awareness for an emerging collaboration; it allows aggregating a mix of different object types.

System Architecture

Our design approach and the envisioned use of the system contributed to the decision to implement our prototype as a peer-to-peer system. In particular, the high administrative cost of centralized shared workspace systems was a major factor in this decision. We wanted users to be able to create shared objects on the fly in order to facilitate instantaneous and effortless collaboration. Giving full control to the user implies that the system should function without any additional infrastructure.

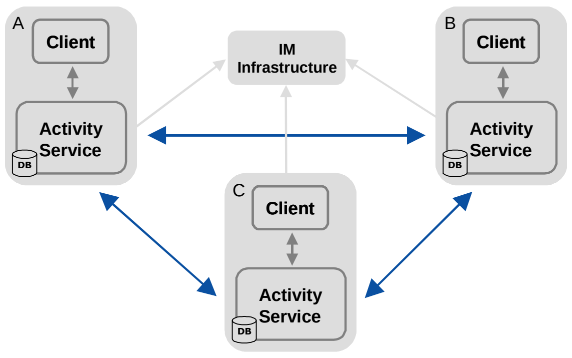
Another major requirement is that users be able to collaborate both synchronously and asynchronously. Asynchronous work may take place while people are online but also off-line when they are disconnected from the network. To provide off-line access at any time, shared objects need not only be persistent but to reside on the user's local machine (desktop, laptop, or PDA). In order to synchronize local replicas, the system must provide appropriate communication and consistency control mechanisms. Since the membership of shared objects can be highly dynamic, the system also has to support late-joining users (Vogel, Mauve, Geyer, Hilt, & Kuhmuench, 2000). Besides ubiquitous access to data, replication helps to achieve good responsiveness.

Finally, object-centric sharing as described in Section "Design Approach" implies that membership management occurs individually for each shared object. This fine-grained access to and control of shared objects might entail a scalability penalty in a server-based system. A replicated system scales better with respect to the number of shared objects and the number of users.

Out of these considerations, we opted against a client-server solution and decided to build a peer-to-peer system where each user runs an equal instance of the application locally. Figure 3 shows three applications instances (A, B, and C).

Each local instance consists of a client component that provides the user interface to the system (see Section "User Interface") and a service component, called ActivityService, that maintains a copy of all shared objects that are relevant to the local user. Peer discovery is accomplished by leveraging an existing instant messaging infrastructure (see Section "Peer Discovery"). We

Figure 3. System architecture



use a local database in each application instance to store and access shared objects. Changes to the set of shared objects (e.g., creating a new object) must be communicated to all peers that participate in an activity.

From the user's perspective, this peer-to-peer architecture means that, apart from acquiring and running the application, no extra steps are necessary before new shared objects can be created or existing shared objects can be accessed. New peers can be invited and integrated easily, for example, by an e-mail referencing a Web link that automatically installs the application.

Data Model

All shared objects with the current values of their attributes form the *state* of the application. Generally, the state of a shared object may change either due to user actions or due to the passage of time. An example for a state change due to the passage of time is the movement of an animated object along a flight path. Such state changes are deterministic and can be calculated locally by each application instance on basis of the object's current state (e.g., speed and direction) and the physical laws captured by the application (e.g., friction). Thus, they do not require that the peers exchange information. In contrast, state changes that are caused by user actions are nondeterministic and must be propagated to all participants of an activity in order to keep the local copies of the shared objects synchronized. Examples for such user actions are the creation of a new shared object or the modification of an existing object, for example, when adding a message to a chat object. State changes can be encoded either as complete states that contain the updated values of all attributes, or as events that contain the modification only. Our prototype encodes new objects as states and all other changes as events. In the following, we denote states and events that are propagated to the set of users as *operations*.

Objects that allow state changes due to user actions only are called *discrete*, and objects that also support time-based changes are called *continuous*. This distinction has to be taken into account when designing a consistency control mechanism (Mauve, 2000). While all shared object types currently supported by our prototype are discrete, our algorithms developed can be easily adapted to meet the needs of continuous objects.

Communication Protocols

The application-level protocol description for the distribution of states and state changes among the peers is based on XML, mainly to allow rapid development and easy debugging. Preliminary tests with our prototype have shown that the resulting performance is sufficient, but should the need arise, we plan to switch to a binary protocol. Using XML as a protocol description will also facilitate a possible future transition to standard XML-based communication protocols, such as XMPP, which is the basis for Jabber (Jabber Software Foundation, 2003). While this does not alleviate performance concerns, it does open an opportunity for interoperability with different collaboration applications. To support XMPP and interoperability, Miller (2001) discusses the infrastructural pieces required to facilitate collaboration amongst users and peer-to-peer applications, which includes unique identity, contextual presence, awareness of a community, protocol independence, discovery, and conversation management.

Since objects can be shared by an arbitrary number of users, application data usually needs to be delivered to more than one destination. Thus, the system has to employ a group communication protocol. We opted against IP multicast due to its insufficient deployment (Chu, Rao, Seshan, & Zhang, 2001) and because it would require the use of UDP as a transport protocol, which is blocked by firewalls in many organizations. Instead, a sender delivers application data directly via TCP to each receiver, forming a star-shaped distribution topology. Since we expect groups for most activities to be small (i.e., up to 10 participants), the network overhead imposed by this approach seems to be acceptable. An alternative would be to use an application-level multicast protocol such as Narada (Chu et al., 2001), which remains an issue for future work.

Peer Discovery

Building the application's communication on top of point-to-point unicast connections means that a sender has to contact each receiver individually. Therefore, a peer needs to know the IP addresses of all the peers it is collaborating with. Since our prototype is integrated into an e-mail client, the natural contact information of a peer is its user's e-mail address. This has to be mapped to the IP address of the corresponding user's computer. The address resolution is a dynamic process because an IP address might change when users connect via dial-up, work on different computers, or use dynamic IP.

To allow a peer to connect to the system for the first time, we use the existing instant messaging (IM) infrastructure in a company to resolve e-mail addresses (see Figure 3). Each peer connects to its user's corresponding instant messaging server. Each peer contacting the server also leaves its own address information—that is, the IM infrastructure serves as a means for peer discovery and address resolution. All addresses resolved by a peer are saved in a local address table together with the time the information was obtained. This address table is persistent and used on the first attempt to establish a connection to another peer. Should this fail, the instant messaging server is contacted to inquire whether the peer's IP address has changed. Once a peer makes contact to other peers, they can exchange addresses to complement their local address tables with new entries and to exchange outdated addresses. In this way communication with the address server can be limited and the peer-to-peer system will be able to function for some time in case the address server is not reachable.

We are not focusing on peer discovery protocols in this work. The above mechanisms could be also easily replaced with existing peer discovery protocols such as Gnutella (LimeWire, 2003) or JXTA (Wilson, 2003). Another alternative would be to make use of the e-mail integration and exchange IP addresses with special e-mails that are filtered by the e-mail client. The advantage of this approach would be that no additional infrastructure for address resolution is required. Its realization is an issue for future work.

Consistency Control

The replication of the application's state as described in the previous section requires explicit mechanisms to keep all state copies consistent. Much research has been done in keeping the state of synchronous multiuser applications such as whiteboards, shared editors, and so forth, consistent. Likewise, our prototype requires consistency mechanisms when people are working on shared objects at the same time. However, by design, our system also supports off-line use when people are disconnected from the network, and we allow people who are online to share objects with others who are currently off-line. Little work has been done on algorithms that support consistency in blended synchronous and asynchronous collaborative applications. We have chosen and modified an existing consistency mechanism for synchronous collaboration so that it also supports asynchronous collaboration. In the following, we first describe consistency control in the "ideal" case when everyone is online before we cover the asynchronous case.

Synchronous Collaboration

Consider the scenario described in Section "User Interface." Let us assume that Bob decides to change the name of the shared screen object that he and Dan were annotating to "project homepage" (see (7) in Figure 2). In order to execute this local state change in Dan's application, Bob's application needs to propagate the pertinent information. But since this transmission is subject to network delay, Dan could also change the object's name to "first draft" in the brief time span before Bob's update is received. In this case, Bob and Dan's changes are concurrent, and they conflict since they target the same aspect of the state.

Without further actions, the name of Bob's activity would be "first draft" and that of Dan "project homepage," meaning that the object's state is inconsistent. To prevent this, the application needs to employ an appropriate consistency control mechanism.

To be more specific, there are two consistency criteria that the application should observe: *causality* and *convergence* (Ellis & Gibbs, 1989). Causality means that an operation O_b that is issued at site i after another operation O_a was executed at i needs to be executed after O_a at all sites, so that the cause is always visible before the effect. For example, the shared screen object has to be created before its name can be changed. Convergence demands that the state of all peers is identical after the same set of operations $\{O_i\}$ was executed. This means in our example that Bob and Dan should see the same name of the screen activity.

Consistency control mechanisms that fulfill these two criteria can be classified into pessimistic and optimistic. Pessimistic approaches seek to prevent conflicts, such as the one described above, from happening by allowing only one user to change a certain aspect of the application's state at a certain point in time, for example, by locking (Munson & Dewan, 1996). Their drawback is that they constrain collaboration and reduce the responsiveness of the system. Optimistic algorithms allow conflicts to happen and repair them afterward in an efficient way. They work best under the assumption that conflicts occur only infrequently. Examples are object duplication (Sun & Chen, 2002), operational transformation (Ellis & Gibbs, 1989), and serialization (Sun, Jia, Zhang, Yang, & Chen, 1998). Object duplication creates a new version of an object whenever conflicting operations occur, thus presenting multiple versions of the same object to the user. We believe that this approach is confusing for the user and is opposed to the original goal of groupware, that is, to allow users the joint manipulation of the same object. Operational transformation converts concurrent operations so that they can be executed in the order in which they are received. However, the drawbacks of this approach are that it is not easily applicable to all multiuser applications (e.g., continuous objects), and that its implementation might be rather complex.

Thus, we decided to use serialization for establishing causality and convergence. The basic idea is to execute a set of operations that targets a certain object in the same order at all sites. As a prerequisite, an appropriate ordering relation has to be defined. Possible ordering relations are timestamps (Mauve, 2000) or state vectors (Lamport, 1978). When using timestamps, operations are ordered by their assigned execution time. Timestamp ordering can be applied to all types of objects, including continuous objects. However, it requires that the clocks at all sites are synchronized, which increases the administrative overhead and creates dependencies on a time synchronization infrastructure. Hence, we decided to use state vectors to order operations in our prototype. This is sufficient because all available object types are discrete. Should the need for continuous synchronization arise, the mechanisms described below can be switched easily to a timestamp-based ordering of operations.

A state vector SV is a set of tuples (i,SN_i), $i = 1,..,n$, where i denotes a certain peer, n is the number of peers, and SN_i is the sequence number of peer i. Whenever i issues an operation O, SN_i is incremented by 1 and the new SV is assigned to O as well as to the state that results after executing O. We define $SV[i] := SN_i$. With the help of state vectors, causality can be achieved as follows (Sun et al., 1998): Let SV_O be the state vector of an operation O issued at site a and SV_b the state vector at site b at the time O is received. Then O can be executed at site b when (1) $SV_O[a] = SV_b[a] + 1$, and (2) $SV_O[i] \leq SV_b[i]$ for all peers $i \neq a$. This means that prior to the execution of O all other operations that causally precede O have been received and executed. If this is the case, we call O *causally ready*. But if O violates this rule, it needs to be buffered until it is causally ready, that is, until all necessary preceding operations have arrived and have been executed.

Convergence can be achieved by applying the following ordering relation to all operations that are causally ready (Sun et al., 1998): Let O_a and O_b be two operations generated at sites a and b, SV_a the state vector of O_a and SV_b the state vector of O_b, and $sum(SV) := \Sigma\, SV[i]$. Then $O_a < O_b$, if (1) $sum(SV_a) < sum(SV_b)$, or (2) $sum(SV_a) = sum(SV_b)$ and $a < b$.

In the following, we denote the set of operations that was received by a peer and executed in the order defined above as *operation history*. Due to the propagation delay of the network, it might happen that an operation O_a that should have been executed before an operation O_b according to the ordering relation is received only after O_b has been executed, that is, O_a would not be the last operation when sorted into the history. This means that applying O_a to the current state would cause an inconsistency. Thus, a repair mechanism is required that restores the correct order of operations. Timewarp (Mauve, 2000) is such an algorithm and works as follows: Each peer saves for each shared object the history of all local and remote operations. Moreover, snapshots of the current state are added periodically to the history. Let us assume that an operation O_a is received out of order. First, it is inserted into the history of the target object in the correct order. Then the application's state is set back to the last state saved before O_a should have been executed, and all states that are newer are removed from the history. After that, all operations following O_a are executed in a fast-forward mode until the end of the history is reached. To avoid confusion, only the repaired final state of the application should be visible for the user.

The advantages of the timewarp algorithm are that it functions exclusively on local information and does not require additional communication among peers; it is robust and scalable; it is applicable to all peer-to-peer applications; and the operation history can be reused for other purposes such as versioning and local recording. One major drawback is the memory usage of the operation history which is determined to a large part by the frequency of state snapshots. While a low frequency saves memory, it increases the average processing time for the execution of a timewarp. From our experience gained with the prototype implementation, we opted to save a state snapshot every 10–15 operations, depending on the shared object.

The size of the operation history can be limited by analyzing the information included in the state vector SV_O of an operation O issued by peer j and received by i: $SV_O[k]$ gives the sequence number of the last operation that j received from k, that is, j implicitly acknowledges all older operations of k. Under the condition that O is causally ready, there can be no timewarp triggered by operations from j that are concurrent to operations from k with $SN_k \leq SV_O[k]$. This means, that from j's perspective, i can remove all operations of k with $SN_k \leq SV_O[k]$ from its history. Once i has received similar acknowledgments from all peers, old operations can be cleared. This process can be sped up by periodically exchanging status messages with the current state vectors of a peer.

In the unlikely event that a new peer late-joins an ongoing collaboration on a shared object, obtains the current state, and issues a state change concurrently to the operation of another peer, the aforementioned improvement might fail since peers clearing their history have no knowledge about the late-joining peer. This can be prevented by keeping outdated operations for some extra time span or by using an additional repair mechanism such as state request (Vogel & Mauve, 2001).

Another drawback of the timewarp algorithm is the processing time to recalculate the application's state. But since each shared object has its own operation history and since states are inserted frequently into the history, the time required for a timewarp is usually below 40 msec on an AMD 2.0 GHz PC, which is not noticeable for the user. Also, as found in Vogel and Mauve (2001), the number of timewarps can be reduced dramatically by analyzing the semantics of concurrent actions: Let O_a be a late operation and $\{O_i\}$ the set of executed operations that follow O_a's correct position in the operation history. In case O_a and all operations O_i change independent aspects of an object, O_a can be executed without a timewarp and without violating the convergence criterion.

The actual effect of a timewarp might be disturbing for a user: It is possible that the recalculated state differs to a high degree from the object's state that was visible before, for example, when objects are deleted. In this case, an application should provide the user with information about the cause of the visual artifact, and highlight the objects and the users involved (Vogel & Mauve, 2001).

Asynchronous Collaboration

A key aspect of our prototype is that shared objects can be accessed by users anytime and independently of others. They are persistent even when all peers are off-line, and they facilitate synchronous as well as asynchronous collaboration. The latter means for the scenario described in Section “User Interface” that Bob can access and manipulate data of the shared screen object (see (7) in Figure 2) even when Dan is off-line and his ActivityService is not running (note that in case Dan is not currently working on a shared object but his application is running, no additional actions are required since Dan’s ActivityService still receives all state updates). This raises the question how Dan’s application can acquire the information that was missed once it is active again so that it possesses the current state and Dan is able to continue the collaboration. For easier discussion, we denote the process of reconnecting after operations have been missed as late-join, the peer that missed operations as late-join client, and any peer that provides information to the late-join client as late-join server (Vogel et al., 2000).

One design option is whether the late-join client should be provided with all operations missed or, alternatively, with a copy of the current state. The latter option has the advantage that a state transmission is more efficient in terms of data volume and processing time whenever the number of missed state changes is rather high (depending on the nature of the object’s state and the state changes). But at the same time, this approach requires additional measures to ensure the consistency of the late-join client’s state (Vogel & Mauve, 2001): The late-join server providing the object’s state cannot guarantee its consistency since concurrent operations that are not included yet could be under way. Thus, we chose the first option and provide all missed operations. This is more robust since the late-join client will possess a complete operation history under the condition that all missing operations will be delivered eventually. The late-join client is then able to ensure the consistency of the object’s state as described in the previous section.

Other design options concern whether the data transmission is initiated by the late-join client or the server, and when the transmission takes place. Initiating and controlling the transmission by the late-join client is the obvious choice since the client has to ensure the consistency of its own state. When starting the application instance, the client therefore contacts all peers with whom it collaborates one by one, and hands over the current state vectors of its shared objects. Comparing those state vectors to its own, a peer can decide which operations from its history need to be sent to the late-joining peer (if any). Thus, the late-join client can obtain missed state changes from any peer that participates in a shared object, not only from the original source of an operation. This is especially useful when that source is off-line at the time the late-join client connects. Additionally, it increases the robustness of the system and spreads the burden of initializing clients to all peers. To further increase the probability that all missed operations are received, information about the current state of objects is also exchanged whenever a user joins a shared object.

However, this approach succeeds only if a peer that was already participating in a shared object missed some state changes. But another possibility is that a new shared object is created or that a new member is invited to join an existing object (as in step (1) of the scenario in Section “User Interface”). Notice that either the peer creating a new shared object or the peer receiving a new shared object could be offline. So when reconnecting, the late-join client does not know about the new shared object and can therefore not ask for the transmission of missed operations. Even worse, it might be that the late-join client does not know the other peers that are members of this shared object if they did not collaborate before. Thus, in case a peer misses the creation of an object or the invitation to an existing object, the responsibility for the retransmission must be with the originator of that action. Should both peers connect for the transmission of any other data, the missed operations can be delivered to the late-join client. The late-join server also tries to contact the client periodically (depending on the current application and network load).

As in the previous case, the probability for quickly updating the late-join client can be increased by placing the responsibility for updating the late-join client not solely on the original source but on all members of a shared object. The more members a shared object has, the higher the likelihood is for a quick update. For this purpose, the peer that created an object or invited new members to an object informs all available peer members that the relevant operations could not be delivered to one or more peers. All peers are assigned the task to try to transmit the operations to the late-join client either by making regular contact or by probing periodically. They stop as soon as they receive a state vector from the client that indicates that it possesses the required information. In case the late-join client receives the same operations more than once, it simply ignores them.

In summary, our approach to achieve a consistent application state (also in case of asynchronous/off-line use) is based on the retransmission of missed operations, accomplished by the aforementioned mechanisms in our prototype. Retransmissions can be initiated by the late-joining peer, by the peer who was the source for an operation, or by other peer members of a shared object. All operations received by the late-joining peer are then executed in the order defined in the previous section. They might trigger a timewarp if necessary so that consistency can be achieved.

Simulation Results

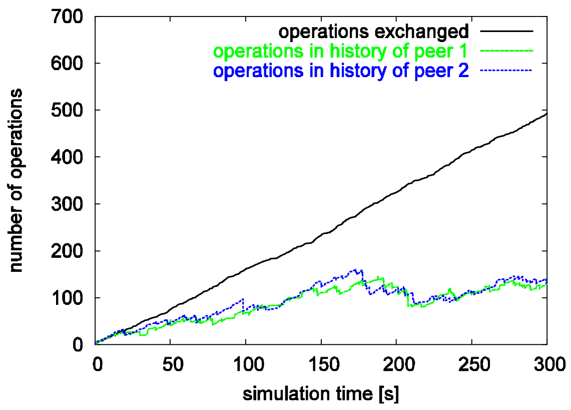
In order to evaluate the correctness and the performance of the algorithms described above, we simulated different scenarios for the synchronous and asynchronous collaboration of two and three peers, respectively. In each scenario we randomly reproduce the activities of a typical work week with 5 days. During one simulated workday, the following actions take place on average: A total of three shared objects are created, each peer views the data of an object 15 times (i.e., opens and closes the view window of an object 15 times), and each peer modifies the state of an object 10 times. The simulated scenarios differ in respect to the time span that each peer is working either online or off-line. For easier handling, we simulated each workday in 60 seconds. Please note that this increases the probability for a timewarp when peers work synchronously. Because of the limited number of operations in our scenarios, we insert a state snapshot every five operations into the history.

Results for the Timewarp Algorithm

When all peers are collaborating synchronously over the entire simulation time, a timewarp can happen only under the rare condition that two or more operations targeting the same shared object are issued concurrently, that is, within the time span that is necessary to propagate these operations. For a scenario with two peers, a total number of five timewarps was triggered with an average duration of 26 msec and an average number of 6.4 operations that needed to be executed in order to regain a consistent state (please note that the sequence of operations to be executed in the case of a timewarp also included operations that were not causally ready before). When both peers are working online for only 80% of the simulated time, a timewarp becomes much more likely since concurrent actions can now happen over the whole time span where at least one peer is off-line. Consequently, a total number of 27 timewarps occurred that took an average execution time of 28 msec and an average sequence of 6.7 operations to rebuild the application's state. When the time that peers spend online was reduced further to 50%, 118 timewarps happened with an average time of 32 msec and 13.2 operations to be executed. In the last scenario, the two peers worked synchronously only for 20% of the simulated time. Since in this case many shared objects and subsequent actions were actually created when being off-line, the total number of timewarps decreased to 44 with an average duration of 30 msec and 9.5 operations in a timewarp sequence. In all cases, the average time to execute a single timewarp was below 32 msec which is not noticeable for the user.

The effect of the algorithm to reduce the size of the operation history can be seen in Figure 4, which depicts the total number of operations exchanged in compari-

Figure 4. Size of operation history



son with the actual size of the history of the two peers, assuming an online time of 80%. In this scenario, the algorithm was able to limit the size of the histories to approximately 25% of the total number of exchanged operations. For the other scenarios, this number lay between 20% and 40%.

Results for the Delivery of Late-Join Data

While a peer is working off-line, operations originating from or targeting that peer cannot be delivered immediately. Instead, the algorithms described in Section 5.2 transmit the missed operations when the peer reconnects. Figure 5 shows the time periods for peers collaborating synchronously in the scenario where the three peers spend 80% of the simulation time online. The number of operations that actually need to be delivered belatedly because the receivers are unreachable, is depicted in Figure 6. The late-join algorithm managed to update peers as soon as they were online again. The curves include those operations that are cached by all receivers that were online at the time they were issued (see Section "Asynchronous Collaboration"). On average, peer 1 stored 5.3 operations (peer 2: 7.2, peer 3: 5.3) for the other peers, and it took about 3.9 sec (peer 2: 4.4 sec, peer 3: 4.0 sec) between generating and delivering a certain operation (please note that peers were offline at the end of all simulations (see Figure 5) and operations that had not been delivered were not included in the analysis of the delivery time span). These numbers increased considerably for the scenario where peers were online for 50% of the simulation time: Peer 1 cached on

Figure 5. Online time of peers

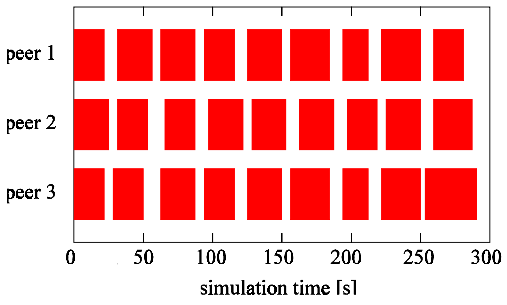
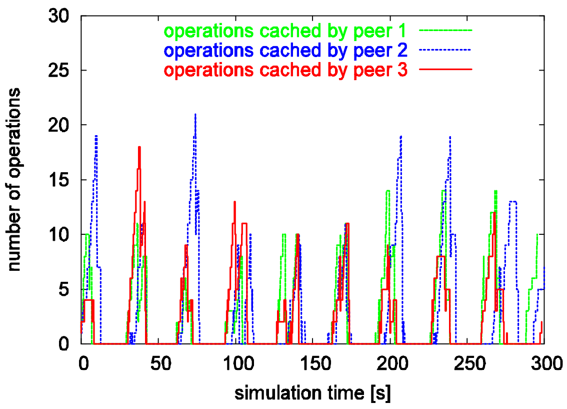


Figure 6. Number of undeliverable operations



average 25.4 operations (peer 2: 15.3, peer 3: 22.5) and missed operations were transmitted 22.7 sec (peer 2: 13.5 sec, peer 3: 20.5 sec) after they were issued. In the last scenario where peers were online for only 20% of the simulation time, the first peer met the other peers only rarely. Consequently, peer 1 stored on average 107.4 operations for 60.6 sec (peer 2: 80.4 operations for 22.8 sec, peer 3: 66.4 for 27.6 sec). These numbers show that the state of a shared object might diverge to a considerable degree when peers are working offline for a longer period of time. The algorithm for the distributed caching of missed operations most likely will alleviate this problem, if the number of members of a shared object increases. However, its performance also depends on the work patterns of the members of a shared object.

Lessons Learned and Discussion

Our design approach and usage scenarios were a major factor in the decision of building a peer-to-peer system that would function without much administrative overhead. The most challenging aspect of this system is to make it work for asynchronous/offline collaboration as well as for real-time collaboration. Our approach builds upon an existing consistency mechanism for synchronous collaboration. In Section "Asynchronous Collaboration," we discussed several strategies that we implemented to make sure that this algorithm would also work in case of asynchronous/off-line work. They are based on recovering the operation history after periods of off-line/asynchronous work. It is interesting to note that all mechanisms proposed are independent from a specific shared object type (e.g., chat or annotated screen shot); they work on the basis of generic properties such as state vectors and operations. Thus, our prototype can be easily extended to other object types. Our simulation results in the previous section indicate that our system design works and performs adequately. However, our approach has also some shortcomings.

The time for reaching a consistent state after periods of off-line use depends very much on user behavior. In the worst case, local states of a shared objects might diverge for a very long period of time, for example, if two users are alternating between being off-line and online. Each user works on a state that is not the current state and then, after eventually merging the operation history, they might see an unexpected result. This reflects a major problem of this approach: While the state is now consistent for both users, the machine cannot guess what their actual intention was when they were independently modifying the shared objects. The algorithm only makes sure that both can see the same end result. However, for the user it is difficult to understand how the resulting state of the object came to be. We believe that it is crucial to provide some mechanisms that assist the user in this task. For example, the application could animate parts of the history to visualize the sequence of operations that led to the current state. Also, the application could provide access to older versions of a shared object and allow a user to modify the current object state in coordination with conflicting participants. Such aids can be realized easily, since each peer maintains the operation history locally, containing all the information required.

There are also two rather technical solutions that would alleviate the aforementioned problem: Adding additional servers, which cache operations and update late-joining peers, would help to decrease the time between resynchronizations of states. Whenever peers are connected to the system, their application can send operations to the caching server, if the collaborating peers are currently off-line. When the other peers connect, they first contact the caching server to collect missed operations. The drawback of this approach is that it requires additional infrastructure that needs to be maintained. Another technical approach could be to put more semantics into the consistency algorithm itself. The ordering of operations in state vectors is based on sequence numbers, that is, when people work off-line for long periods of time, the system only looks at the sequence numbers to restore a consistent state. But the sequence numbers do not reflect when operations actually took place. If two users work off-line at different times, ordering of operations could be prioritized by the recency of the state change, which would probably help users in better understanding the resulting state after merging the operation histories. This could be achieved either by using globally synchronized clocks as a means for ordering (which again requires infrastructure) or, more elegantly, by using a modified state vector scheme that incorporates local time into the numbering.

One issue we did not address deeply in this work, but is of concern with any collaborative application, is security. From an architectural and user perspective, we do offer object-centric access control, with two roles: member and creator. A user creating a shared object (i.e., a "creator") in our system assigns members at creation time, as well as afterward, thus defining who has permission to access that object. Members can add additional objects and other members, but they cannot drop other members except themselves, unless the member who performs the drop operation is the creator of the shared object. The creator is also the only one who has the right to delete the object. An area for improvement is to support different kinds of roles (e.g., reader, cocreator), but a challenge is to ensure that such measures do not make the security management heavyweight for the user. For the sake of time, we built our prototype with only a simple password scheme to authenticate, and transmitted data is not encrypted. For a commercial system that would be trafficking personal information and messages, securing the protocol is important. Udell, Asthagiri, and Tuvell (2001) discuss the issues involved in securing a peer-to-peer workspace, which involves creating access keys for individuals as well as on-the-fly workgroups.

It is noteworthy that when we started this project, we were not anticipating the complexity of such a peer-to-peer system despite the fact that some of the authors of this paper had several years of experience in developing distributed systems. The implementation of only the consistency mechanisms took three full person months. And, before starting to work on the peer-to-peer aspects, we already had a server-based prototype available.

When building a peer-to-peer system, the advantages of this approach have to be carefully traded off against the benefits that a server-based solution offers. We believe that there is no definite answer to the question of peer-to-peer versus server-based. The design philosophy and the user experience of our system require off-line use. Hence, we need replication of data to the local machine. However, the aspect of off-line use would also greatly benefit from having a server that helps synchronize divergent copies of the distributed state as discussed above. We are currently investigating hybrid architectures that use servers for resynchronization, caching, address resolution, and other useful more lightweight services. While they hold a master copy of the state, local applications would still communicate directly peer-to-peer. In this topology, the server would be just another peer with a different role. The presence of a server facilitates synchronization. However, please note that even with a server, there is still a need for consistency algorithms like the one described in this work. When working offline, local replicas and operations need to be merged with the master copy on the server. With some modifications, the algorithms described earlier could also be used in a hybrid architecture. We are aware that a hybrid architecture again comes with the problem of an additional infrastructure that needs to be introduced and accepted within an organization, which may take a long time. In the meantime, pure peer-to-peer systems help pave the road to get there.

We believe that ultimately the user experience has to be peer-to-peer. E-mail is actually a very good example for such a hybrid system that feels peer-to-peer from a user's perspective but is implemented through a network of mail servers (which talk peer-to-peer to one another). Most e-mail clients today also support off-line use by keeping local replicas of their inboxes. However, e-mail does not support shared states and real-time collaboration and thus does not face the consistency challenges like in our system.

This chapter has mainly focused on the technical aspects of a system that supports object-centric sharing. However, this new paradigm also deserves and needs to be studied from a human-computer interaction (HCI) perspective. A usage study was only recently completed (Muller, Geyer, Brownholtz, Wilcox, & Millen, 2003), and we do not want to anticipate results here. However, there is some preliminary feedback that is worth mentioning. The system was shown to more than 50 people through demos on three consecutive days. We also used the system informally in our own group for a period of six weeks for discussions and bug reporting specific to the system itself.

Many people reported that the notion of activity-centric work is a substantial part of their everyday work practices. They liked the capabilities in our prototype that support this notion, such as aggregating related items and seamlessly moving back and forth between different modes of collaboration. While the design philosophies of object-centric sharing and conversational structure seem to resonate very well, people had some concerns about our user interface design and other shortcomings that we had not addressed then. Our user interface, for example, shows tree-like structures the way they are stored in the system. We envision that there are better ways of presenting and navigating that information. More conversational interfaces that are structured by time could help understanding the history of an activity. People-centered views could help focus on whom you want to collaborate with and could display shared objects that relate to a certain user or a group of users. A disadvantage of the tree-like approach is also that this tree-hierarchy is forced on all users, including newcomers who might not understand how to navigate this structure. Whether this is a serious hindrance or not will be determined by our ongoing user evaluation.

Another major issue was the management of shared objects. People reported that they are engaged in numerous lightweight activities every day. They were concerned that the user interface can easily become cluttered and that it would be hard to find information in the tree. More work needs to be done to face this challenge. Activities might grow quickly and evolve to agile, more formal public spaces. We need to provide means so that people can transform these to a shared workspace system. Many activities though will never reach that state. So as a major desired feature of our system, people were asking for some kind of archiving mechanism that helps them get completed stuff out of their way in the tree, with the option of still having access to the information for later perusal.

Conclusion

The domain of peer-to-peer cannot be characterized solely by file sharing and network protocols. Peer-to-peer is a powerful programming model that can benefit everyday users with lightweight, synchronous, and asynchronous collaboration. This chapter described a new peer-to-peer collaboration technology that takes one step to help realize this promise. Our peer-to-peer system targets collaborative activities sitting midway between ad hoc communication in e-mail and more formal collaboration in shared workspace systems. As a major new design principle, we introduced the notion of object-centric sharing, which allows people to aggregate and organize shared objects into activity threads, providing an emerging context that evolves and engages a dynamic group of participants. Being "placeless," this approach imposes little overhead and allows for lightweight, ad hoc collaboration. A peer-to-peer implementation under the hood was a natural choice for a system with these characteristics.

However, developing advanced peer-to-peer systems whose functionality goes beyond simple file sharing is a challenge. In this chapter we have described various technical aspects and implementation issues of a novel peer-to-peer prototype system. As an implication of our design approach of object-centric sharing, we need to maintain consistency in a blended synchronous and asynchronous collaborative system. To achieve this goal, we have enhanced a popular consistency control algorithm, which had been originally designed for real-time collaboration. Our simulation results indicate that this approach works well. However, a major difficulty is that during long phases of asynchronous work the application state might diverge significantly. To alleviate this problem, we are currently looking into improved versions of our consistency algorithm and we are also investigating new hybrid architectures that use lightweight caching servers to make the system more robust. There are several other unresolved technical challenges that we have not touched upon deeply in this chapter. Security of shared objects and scalability of fine-grained sharing of content remain interesting future directions in the evolving domain of peer-to-peer applications.

References

Bellotti, V., & Smith, I. (2000). Informing the design of an information management system with iterative fieldwork. *Proceedings ACM DIS 2000*, 227–238.

Bellotti, V., Ducheneaut, N., Howard, M., & Smith, I. (2003). Taking e-mail to task: The design and evaluation of a task management centered e-mail tool. *Proceedings ACM CHI 2003, 5*, 345–352.

Bernstein, A. (2000). How can cooperative work tools support dynamic group processes? Bridging the specifity frontier. *Proceedings ACM CSCW 2000*, 279–288.

Chu, Y., Rao, S.G., Seshan, S., & Zhang, H. (2001). Enabling conferencing applications on the Internet using an overlay multicast architecture. *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOM)*, 55–67.

Cohen, A.L., Cash, D., and Muller, M.J. (2000). Designing to support adversarial collaboration. *Proceedings of ACM CSCW 2000*.

Dourish, P., Edwards, W.K., Lamarca, A., & Salisbury, M. (1999). Presto: An experimental architecture for fluid interactive document spaces. *ACM Transactions on Computer-Human Interaction, 6*(2), 133–161.

Ducheneaut, N., & Bellotti, V. (2001a). A study of e-mail work activities in three organizations. Manuscript in preparation.

Ducheneaut, N., & Bellotti, V. (2001b). E-mail as habitat: An exploration of embedded personal information management. *ACM Interactions, 9*(5), 30–38.

Eclipse Project – Universal Tool Platform (n.d.). Retrieved October 13, 2003, from http://www.eclipse.org/platform/index.html

Ellis, C.A., & Gibbs, S.J. (1989). Concurrency control in groupware systems. *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data (ACM SIGMOD), 18*(2), 399–407.

Eudora Sharing Protocol (ESP) (n.d.). Retrieved October 13, 2003, from http://www.eudora.com/e-mail/features/esp.html

Geyer, W., & Cheng, L. (2002). Facilitating emerging collaboration through light-weight information sharing. *Proceedings of the 2002 ACM Conference on Computer-Supported Cooperative Work: Conference Supplement ACM CSCW 2002*, 167–168.

Geyer, W., Vogel, J., Cheng, L., & Muller, M. (2003). Supporting activity-centric collaboration through peer-to-peer shared objects. *Proceedings of the ACM 2003 International Conference on Supporting Group Work.*

Groove Networks (n.d.). Retrieved October 13, 2003, from http://www.groove.net

Jabber Software Foundation (2003). Retrieved October 14, 2003, from http://www.jabber.org

Kaptelinin, V. (2003). UMEA: Translating interaction histories into project contexts. *Proceedings ACM CHI 2003, 5*, 353–360.

Kerr, B.J. (2003). Thread arcs: An e-mail thread visualization. *Proceedings Infovis 2003*.

Kubi Software – Collaborative E-mail for Teams (2003). Retrieved October 13, 2003, from http://www.kubisoft.com

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM, 21*(7), 558–565.

LimeWire: Running on the Gnutella Network (2003). Retrieved October 13, 2003, from http://www.limewire.com

Lotus Notes (n.d.). Retrieved October 13, 2003, from http://www.lotus.com/products/product4.nsf/wdocs/noteshomepage

Mackay, Wendy E. (1988). More than just a communication system: Diversity in the use of electronic mail. *Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work (ACM CSCW'88)*, 344–353.

Mauve, M. (2000). Consistency in replicated continuous interactive media. *Proceedings of the 2000 ACM Conference on Computer-Supported Cooperative Work (ACM CSCW 2000)*, 181–190.

Microsoft NetMeeting (2002). Retrieved October 14, 2003, from http://www.microsoft.com/netmeeting

Microsoft Office – Outlook Home Page (2003). Retrieved October 13, 2003, from http://www.microsoft.com/outlook/

Miller, J. (2001). Jabber: Conversational technologies. In A. Oram (Ed.), Peer-to-peer: Harnessing the benefits of a disruptive technology (pp. 77–88). Sebastopol, CA: O'Reilly.

Muller, M., Geyer, W., Brownholtz, B., Wilcox, E., & Millen, D.R. (2003). One hundred days in an activity-centric collaboration environment based on shared objects. Manuscript submitted for publication.

Munson, J., & Dewan, P. (1996). A concurrency control framework for collaborative systems. *Proceedings of the 1996 ACM Conference on Computer-Supported Cooperative Work (ACM CSCW1996)*, 278–287.

Rohall, S. L., Gruen, D., Moody, P., & Kellerman, S. (2001). E-mail visualizations to aid communications. *Proceedings IEEE Symposium on Information Visualization 2001 (INFOVIS'01): Late Breaking Hot Topics and Interactive Posters*, 12–15.

Schoberth, T., Preece, J., Heinzl, A. (2003). Online communities: A longitudinal analysis of communication activities. *Proceedings HICSS 2003*.

Skype (2003). Retrieved October 14, 2003, from http://www.skype.com

SubEthaEdit (n.d.). Retrieved October 14, 2003, from http://www.codingmonkeys.de/subethaedit

Sun, C., & Chen, D. (2002). Consistency maintenance in real-time collaborative editing systems. *ACM Transactions on Computer-Human Interaction, 9*(1), 1–41.

Sun, C., Jia, X., Zhang, Y., Yang, Y., & Chen, D. (1998). Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transactions on Computer-Human Interaction, 5*(1), 63–108.

Udell, J., Asthagiri, N., & Tuvell, W. (2001). Security. In A. Oram (Ed.), Peer-to-peer: Harnessing the benefits of a disruptive technology (pp. 354–380). Sebastopol, CA: O'Reilly.

Vogel, J., & Mauve, M. (2001). Consistency control for distributed interactive media. *Proceedings of the Ninth ACM International Conference on Multimedia (ACM MM2001),* 221–230.

Vogel, J., Mauve, M., Geyer, W., Hilt, V., & Kuhmuench, C. (2000). A generic late join service for distributed interactive media. *Proceedings of the Eighth ACM International Conference on Multimedia (ACM MM2000)*, 259–267.

Whittaker, S., & Sidner, C. (1996). E-mail overload: Exploring personal information management of e-mail. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Common Ground (CHI 1996)*, 276–283.

Whittaker, S., Swanson, J., Kucan, J., Sidner, C. (1997). TeleNotes: Managing lightweight interactions in the desktop. *ACM Transactions on Computer-Human Interaction, 4*(2), 137–168.

Wilson, B.J. (2003). JXTA Book. Retrieved October 13, 2003, from http://www.brendonwilson.com/projects/jxta/

Yates, J., Orlikowshi, W.J., Woerner, S.L. (2003). Virtual organizing: Using threads to coordinate distributed work. *Proceedings HICSS 2003*.

Zero Configuration Networking (zeroconf) (2003). Retrieved October 14, 2003, from http://www.ietf.org/html.charters/zeroconf-charter.html

Chapter X

"Let Me Know What You Know" ReachOut as a Model for a Peer-to-Peer Knowledge Sharing Network

Vladimir Soroka, IBM Haifa Research Lab, Israel

Michal Jacovi, IBM Haifa Research Lab, Israel

Yoelle S. Maarek, IBM Haifa Research Lab, Israel

Abstract

Peer-to-peer (P2P) technology has spread through the Web over the last few years through several incarnations, ranging from search for extraterrestrial intelligence to multimedia file sharing, or resource sharing in the general sense. Collaboration systems, expert finding systems and recommender systems are all examples of resource sharing tools where the basic resource is "knowledge," yet they are not viewed as peer-to-peer due to their centralized architecture. We claim that the process of knowledge sharing in such systems is ofter P2P, regardless of their architecture. This *chapter analyzes knowledge sharing and collaboration models through the prism of P2P technology. We propose a framework for determining when knowledge sharing systems follow a P2P model, review existing collaboration and knowledge sharing systems, and verify whether they qualify as P2P under the criteria we define. We then introduce our "Second-Degree Peer-to-Peer" model for knowledge sharing and illustrate it with ReachOut, a tool for peer support and community building developed at the IBM Haifa Research Lab.*

Introduction

Peer-to-Peer (P2P) technology has clearly crossed the chasm[1], and some of its applications have been so widely adopted that they clearly can be seen as killer applications of the Internet era. P2P is used for a wide variety of purposes. Examples include search for extraterrestrial intelligence (SETI@Home), peer-to-peer instant messaging (P2PMessenger), and telephony services (Skype). Yet the most prominent usage of P2P is multimedia file sharing[2] (mostly for songs and video clips), as offered by numerous applications such as Napster, Kazaa, Gnutella, and eMule. Multimedia is not the only type of resource that may be shared. If P2P systems are effective for multimedia sharing, they should also be effective for the sharing of other resources such as knowledge. Collaboration systems (Groove), expert finding systems (Ackerman, 1994; Ackerman & McDonald, 1996), and recommender systems (Terveen and Hill, 2001) are all examples of knowledge sharing tools. However, these systems have rarely been viewed as peer-to-peer due to their centralized architecture.

The question we address here is whether a centralized architecture really disqualifies a tool from belonging to the P2P class. In Napster, for instance, while the actual media sharing is done directly between two users, the initial lookup is performed on the central server. This leads us to believe that a more generic definition is needed for qualifying systems as P2P systems.

We argue that knowledge can be seen as a resource suitable for sharing and exchange under a P2P model. Many current knowledge-sharing tools have elements of P2P in their design and implementation. We will review here these existing tools and assess how they may benefit from P2P aspects. In order to illustrate our purpose, we consider and study within this context, ReachOut—a tool for peer support, knowledge sharing, and community building created at the IBM Haifa Research Lab.

The rest of this chapter is organized as follows: We first define knowledge and justify why it is an exchangeable resource in P2P systems. We then propose a framework for determining when resource-sharing systems follow a P2P model. We continue by reviewing and analyzing existing knowledge sharing systems so as to verify how they fit the proposed framework. Finally, we define a “Second-Degree Peer-to-Peer” model for knowledge sharing and investigate how our example P2P knowledge sharing tool, ReachOut, matches the model. We conclude by drawing some generic guidelines for creating P2P knowledge sharing systems.

Knowledge as a Resource

Many definitions of knowledge (Alexander, Schallert, & Hare, 1991; Nonaka & Takeuchi, 1995; Polanyi, 1966) have been proposed. For the sake of clarity, we will use here the basic definition of knowledge provided by Ackoff (1989). According to Ackoff, the content of the human mind may be divided to five categories: data, information, knowledge, understanding, and wisdom. The first four categories are of special interest to us. Data consist of symbols; information is associated with processed data, which can provide answers to questions such as who, what, where, and when; knowledge is an application of data and information (answers to “how” questions); and understanding is a result of a cognitive and analytical process that allows people to evaluate “why” questions. A good example of the difference between knowledge and understanding is the multiplication table. Elementary school children **know** the table and thus can easily answer how much 6x8 is, but will fail to find the result of 234x67, since they do not **understand** the process of multiplication.

From this definition, it is obvious that people prefer receiving knowledge rather than raw data or information. Knowledge can be provided by others who possess the relevant knowledge (and thus can help in answering the “how” questions) or people who have a deep understanding of the problem area and can thus provide help in answering the “why” questions. This characteristic of knowledge is, in our opinion, what qualifies it as a valuable resource suitable for trading.

Human knowledge is deeply contextual and is triggered by circumstance (Polanyi, 1966; Snowden, 2002). As a consequence, all attempts to map the knowledge possessed by members of any large group to an explicit data store are doomed to failure. If we try to make people state their knowledge verbally, they will most probably miss a large part of what they know—this part is often referred to as “tacit knowledge”. Tacit knowledge is background knowledge that is used to accomplish the task in focus (Polanyi, 1966). This definition only emphasizes the fact that tacit knowledge is highly circumstantial and as such cannot be easily mapped.

Another important characteristic of knowledge is the fact that people are often reluctant to share the piece of knowledge they posses. Actually the problem of "tragedy of the commons," where people use the common resource without actually contributing to it, is a major problem in virtual communities (Adar & Huberman, 2000; Kollock & Smith, 1996). The value of information (Rafaeli & Raban, 2003) and the value of knowledge have become a popular research topic, which seems to indicate that if knowledge is valuable in the online economy, it is worth exchanging.

Tiwana (2003) argues that the only way to harness collective knowledge is to acknowledge the nature of relationships between people. Moreover, he states that "peer-to-peer networking naturally supports knowledge management by closely adopting the conventions of face to face human communication" (p. 78). Indeed, as supported by many studies and findings, virtual communities are successful precisely because of their mimicking real-life relationship patterns (Rheingold, 2000). P2P does not only imitate real-life communication, it also creates a complex social network, which brings additional value to the knowledge community (Wellman, 1997). People are interested not only in turning to immediate contacts, but also in reaching out to unknown friends of friends and those who can be reached by exploring each individual's social network. This reflects the importance of weak ties—people who are only weakly connected to a person have different information and different knowledge to contribute (Granovetter, 1973).

To summarize, knowledge is a valuable sharable resource. As such, it is attractive enough to be exchanged in P2P networks. Moreover, P2P networks can contribute a lot to knowledge sharing systems by introducing some possible solutions to potential knowledge sharing problems. The following section describes a framework for defining P2P systems.

Peer-to-Peer Assessment Framework

P2P Litmus Test

Shirky (2000) proposes the following litmus test to determine whether a technology qualifies as P2P:

1) Does it treat variable connectivity and temporary network addresses as the norm?
2) Does it give the nodes at the edges of the network significant autonomy?

In order to qualify as P2P, a technology needs to satisfy both conditions. In addition, there is a third, implicit condition, stating that clients are supposed to communicate with each other to make the system work.

Interestingly enough, this test does not include a decentralized architecture as a prerequisite condition (as P2P is often grasped). It rather assumes that each node of the network is more or less an independent player and is not required for the whole system to work. This test is pretty simple and can be easily used for assessing existing knowledge sharing systems to see whether they can be (at least partially) considered peer-to-peer.

P2P Classification

To finally reinforce the option of a centralized architecture of P2P systems, we offer the following classification by Lv, Cao, Cohen, Li, and Shenker (2002). They divide P2P architectures into three types:

a) Centralized, which maintains a resource directory at central locations (e.g., Napster)

b) Decentralized Structured, which does not have a central repository but uses network structure to maintain resource location (e.g., Zhao, Kubiatowicz, & Joseph, 2001)

c) Decentralized and Unstructured, which do not have either a central server or structure maintenance (Cohen and Shenker, 2002; Gnutella, n.d.)

Tiwana (2003) introduces the notion of “hybrid P2P systems” and defines them as systems that while supporting peer-to-peer communication, take advantage of centralized servers for directory services. He focuses on P2P knowledge networking and compares the characteristics of knowledge in the pure and hybrid peer-to-peer models. We will later show how this hybrid model can be extended to account for dynamically created groups.

This comparison, while showing the slight disadvantages[3] of the centralized architecture, still maintains its strong relations to the general P2P model. We will use the proposed litmus test for determining whether a system is P2P and will classify systems using the classification proposed by Lv et al. (2002).

Knowledge Sharing Systems

The once extremely popular Usenet newsgroups (that still have some visibility within Google Groups) are the most prominent example of peer support and knowledge sharing. In Usenet, people share knowledge and opinions to create a common knowledge archive (just like Napster users create a common multimedia archive). Many behavioral patterns in Usenet usage resemble patterns of P2P file sharing systems. Newsgroups are sometime used only as an initial channel; additional communication is done via different channels, so the actual knowledge exchange can be seen as P2P.

Let us apply our litmus test to the newsgroups' knowledge exchange process. Newsgroups are an asynchronous medium, so intermittent connectivity is easily tolerated and is a norm. Each node in the network for our purpose is a potential adviser. Due to the abundance of potential advisers, one individual contributor is not assigned a special or unique value: the absence of a potential adviser will not cause the system to fail. On the other hand, the nodes are certainly autonomous; they are not dependent on the network—not even for the purpose of knowledge seeking, as they may easily switch to other media. Thus, the Usenet newsgroups' knowledge sharing mechanism can definitely be seen as P2P.[4] As for the architecture classification, in spite of the newsgroups' contents being replicated on various distributed servers for performance reasons, they all represent a mirror of a virtual centralized server: askers approach a central group in order to reach potential advisers. In fact, newsgroups also fit the extended P2P model for knowledge sharing, which we discuss later, where a second-degree P2P process takes place among a group of nodes. Indeed, with the evolution of a discussion thread in the newsgroup, peers exchange knowledge and collaboratively form a coherent piece of knowledge that is hopefully new to, at least, a subset of the participants.

Expert finding systems embody another example of knowledge sharing with systems such as Answer Garden (Ackerman, 1994; Ackerman and McDonald, 1996). These systems assist in locating relevant experts and connecting them with a requesting user. Here, again, the nodes are potential advisers, and the same considerations that applied in the newsgroups' case apply here as well. As for the classification, Answer Garden certainly makes use of a centralized server for escalating questions. It does not fit the extended model of second-degree P2P, as once the intermediary step is over and connection between requester and adviser is made, a simple one-on-one connection takes place.

Recommender systems represent yet a different kind of knowledge sharing systems (Terveen & Hill, 2001). The idea behind a recommender system is to accumulate feedback from different users and leverage it in order to recommend resources—such as movies (IMDb) or books (Amazon.com)—to other users. Recommender systems may match close user profiles and recommend resources based on similar user feedback. The architecture of such systems is very similar to resource virtualization,[5] where the resource here is people's assessments of a specific object. Recommender systems typically follow a centralized architecture, and, in fact, the nodes do not engage in any communication among themselves. They are not P2P.

Finally, advanced synchronous collaborative environments, such as Groove, allow end users to share yet another type of knowledge in a synchronous mode, be it one's desktop or desktop area, or more informal knowledge as expressed in instant messaging, shared drawings, and so forth. Groove is implemented as P2P software (Groove Whitepaper), while another similar system, IBM Sametime Server-based e-Meetings (Sametime) is server oriented. However, the process of knowledge sharing in such systems regardless of their physical architecture is often P2P, as can be easily verified by using our proposed framework.

A Model for P2P Knowledge Sharing

We define a knowledge sharing network as a graph where the nodes are people, and resources for sharing along the edges are the implicit knowledge stored in their minds. Some of this knowledge is accessible on demand, at the user's request (via a pull mechanism), while some of it pops out only in specific contexts (as in push mechanisms). We argue here that knowledge sharing networks, according to the above definition, are sometimes configured as pure peer-to-peer but often offer variants, even beyond the hybrid peer-to-peer mentioned earlier.

Knowledge sharing networks, just like in classical peer-to-peer systems such as Kazaa or Napster, allow users to query the system for a specific resource they need, namely a piece of information or knowledge, and all "relevant" clients receive this query. The exchange of resource then takes place between the requester and the providers of knowledge. However, the difference lies in the manner in which relevance is established.

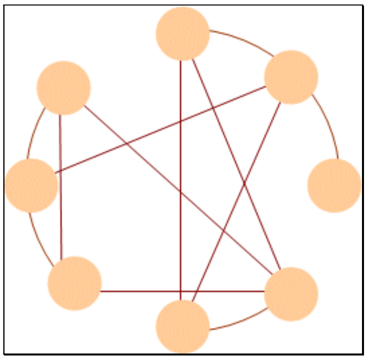
Relevance is usually established as a measure of similarity between an information need and the piece of information that qualifies, thus in search engines, it is a measure between the need expressed as a free-text query and a candidate textual document. When dealing with knowledge encoded in people's minds, relevance can only be approximated since it is difficult to evaluate precisely whether people can really provide the needed knowledge. As mentioned earlier, the needed knowledge is often tacit and multiple interactions are needed in order to extract it. A convenient way to approximate relevance is to ask people to declare their interests manually, either in free-text format or according to a predefined taxonomy in the Usenet newsgroups manner. Subscribing to newsgroups is equivalent to declaring the general areas in which one has interest in the resources (as a seeker or provider). This is different from the perfect match usually sought when looking for MP3 files for instance. Once relevance at this approximate level is established though, peers who are identified as relevant (i.e., who share the common subscription) are put in direct contact to negotiate and discuss between each other in order to isolate the needed piece of knowledge, for instance through a discussion thread in a Newsgroup. We thus see that the exchange of the "knowledge" resource is done neither in a pure peer-to-peer manner nor in the hybrid manner discussed earlier. There seems to be a third model, that we call the "second-degree peer-to-peer," where relevance is established by declaring one's interest either to a centralized directory (like in the hybrid model) or to secondary directories like in the Newsgroups NNTP model, for instance, and then the actual exchange of resources is done at a smaller scale, in a pure peer-to-peer manner, within the boundaries of a group dynamically assembled for the occasion.

We thus propose to extend Tiwana's classification introduced earlier, which in our view covers the most prominent peer-to-peer scenarios, with a third type of peer-to-peer which is mostly encountered in knowledge sharing networks.

Indeed, the first pure peer-to-peer model, as illustrated in Figure 1 (where nodes represent people and edges the conduit along which resources are exchanged), is typically encountered in one-on-one instant messaging-style dialogs. A dialog (if successful) results in knowledge transfer that satisfies the demand of one of the users.

The hybrid peer-to-peer model introduced by Tiwana uses a central directory as a broker, as illustrated in Figure 2, where the central node is a special type of

Figure 1. Pure peer-to-peer knowledge sharing network



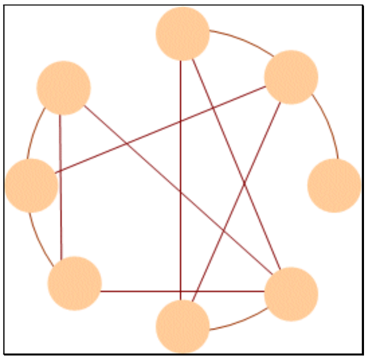
declare their interests manually, either in free-text format or according to a predefined taxonomy in the Usenet newsgroups manner. Subscribing to newsgroups is equivalent to declaring the general areas in which one has interest in the resources (as a seeker or provider). This is different from the perfect match usually sought when looking for MP3 files for instance. Once relevance at this approximate level is established though, peers who are identified as relevant (i.e., who share the common subscription) are put in direct contact to negotiate and discuss between each other in order to isolate the needed piece of knowledge, for instance through a discussion thread in a Newsgroup. We thus see that the exchange of the "knowledge" resource is done neither in a pure peer-to-peer manner nor in the hybrid manner discussed earlier. There seems to be a third model, that we call the "second-degree peer-to-peer," where relevance is established by declaring one's interest either to a centralized directory (like in the hybrid model) or to secondary directories like in the Newsgroups NNTP model, for instance, and then the actual exchange of resources is done at a smaller scale, in a pure peer-to-peer manner, within the boundaries of a group dynamically assembled for the occasion.

We thus propose to extend Tiwana's classification introduced earlier, which in our view covers the most prominent peer-to-peer scenarios, with a third type of peer-to-peer which is mostly encountered in knowledge sharing networks.

Indeed, the first pure peer-to-peer model, as illustrated in Figure 1 (where nodes represent people and edges the conduit along which resources are exchanged), is typically encountered in one-on-one instant messaging-style dialogs. A dialog (if successful) results in knowledge transfer that satisfies the demand of one of the users.

The hybrid peer-to-peer model introduced by Tiwana uses a central directory as a broker, as illustrated in Figure 2, where the central node is a special type of

Figure 1. Pure peer-to-peer knowledge sharing network



second-degree peer-to-peer model is illustrated in Figure 3. Gnutella, for instance, supports a way for users to cooperate in receiving the content they need, thus accelerating the transfer rate of resources (Ripeanu, 2001). People look for different parts of a resource in other users' repositories and thus can assemble the required resource piece by piece from different peers.

While we see via this example that our model is already embodied in other types of networks, we believe that it is implemented de facto in many knowledge sharing networks, simply because group discussions are more productive than one-on-one discussions. Initializing the interaction and making it visible to others who expressed some kind of interests in the topic, allows knowledge seekers to get quality control on the provided knowledge "for free." Indeed, participants can issue comments on the resource exchanged and therefore providers invest more efforts controlling the quality of the resource they generate since they know they are being observed. In addition, passive participants in the discussion benefit from new knowledge resources simply by being present and thus exposed to this knowledge. While they could be perceived as parasites of the process, they play an important role, as they provide de facto quality control as mentioned above.

Table 1. P2P models comparison table

Knowledge Characteristics	**Pure P2P**	**Hybrid P2P**	**2nd Degree P2P**
Flow	Serverless	Initiated through central directory, continues one-on-one	Initiated through central directory, continues in group pure P2P
Search efficiency	Low	High for explicit knowledge	High for explicit and tacit knowledge as well
Autonomy	Highly self-organizing	Intermediate	Intermediate
Reciprocity measurement	Not possible	Possible through directory server	Possible through directory server, and even further—as the group phase is "organized" through the server, a server agent may follow it to measure reciprocity
Knowledge network externalities	High	Low	Medium
Value-creation potential	Medium	Low	High
Expertise maintenance overhead	Low	High	Medium
Scalability	High	Centralized directory server a choke point	Centralized directory server a choke point for location. Following P2P process is scalable
Applications	Internal, codified knowledge sharing	Tacit knowledge integration in dispersed business networks	Improved tacit knowledge access and creation in dispersed business networks

In order to get a complete picture of this model as compared to the two others mentioned earlier, we extend Tiwana's knowledge characteristics table (Tiwana, 2003, p. 78) and add a third column for comparison purposes.

As can be seen from the table above, our model is closely related to the hybrid P2P model but adds some new dimensions crucial for knowledge sharing process. We now proceed to describe our sample application, ReachOut, introduced elsewhere (Ribak, Jacovi, & Soroka, 2002) to emphasize the advantage of second-degree P2P model for knowledge sharing systems.

An Example: ReachOut

The ReachOut Tool

ReachOut is a tool for peer support and community building within organizations, created and deployed at the IBM Haifa Research Lab. Following the early newsgroups approach, it provides an environment for posting questions to predefined interest groups, but unlike newsgroups, it supplies a synchronous instant messaging environment for group discussions, and uses push technology to notify people of new or updated questions in synchronous mode.

When first joining the ReachOut experience, a user is invited to subscribe to groups of interest. The taxonomy of interest groups is a shallow representation of the knowledge in the organization, used for the definition of relevance of potential advisers. A user subscription may be considered as a user profile, indicating areas of interest. It should be noted that subscription by no means indicates expertise, but rather implies some knowledge in the area, or even the interest of a novice who wishes to gain by absorbing other people's knowledge.

Once subscribed, the user may keep ReachOut in its minimized mode. In this mode, ReachOut appears as a system tray icon, which is overlaid by an exclamation point when a notification arrives (Figure 4).

Figure 4. ReachOut in the system tray

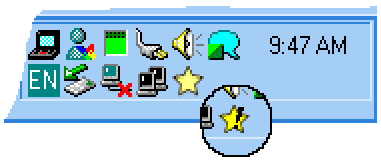
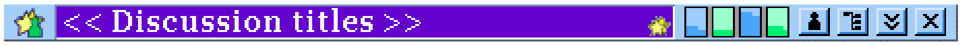


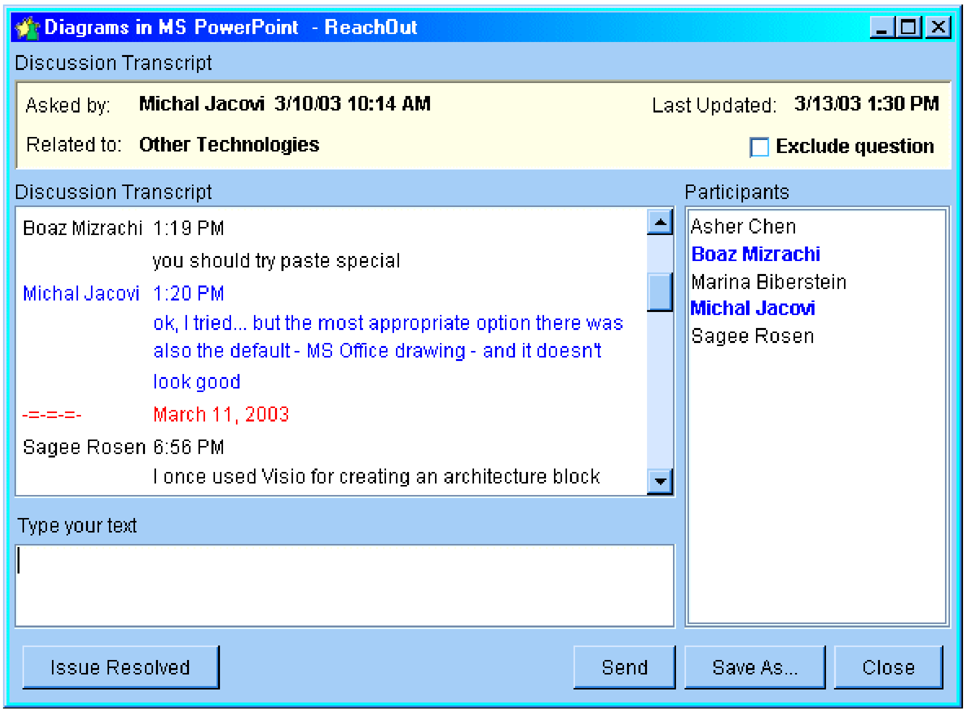
Figure 5. The ReachOut bar



When encountering an information need—where the knowledge of others in the organization may be useful——the user may initiate a ReachOut discussion. For this purpose, the user fills out a short form, supplying a short title for the question, as well as additional details that can help focusing answers. The user then selects target groups for discussing the issue at hand; these groups are used to identify the set of relevant users who will be invited to the discussion. Once submitted, a new ReachOut chat room is created, and notifications are sent to the subscribers.

To open the ReachOut application for taking part in discussions, the user double-clicks the icon and a narrow bar opens (Figure 5) where new and updated discussion titles fade in and out one after the other, decorated with icons that indicate their status. The user may navigate through discussions in several ways,

Figure 6. A ReachOut discussion window



Asked by: Michal Jacovi 3/10/03 10:14 AM
Last Updated: 3/13/03 1:30 PM
Related to: Other Technologies

Boaz Mizrachi 1:19 PM
you should try paste special
Michal Jacovi 1:20 PM
ok, I tried... but the most appropriate option there was also the default - MS Office drawing - and it doesn't look good
-=-=-=- March 11, 2003
Sagee Rosen 6:56 PM
I once used Visio for creating an architecture block

Asher Chen
Boaz Mizrachi
Marina Biberstein
Michal Jacovi
Sagee Rosen

based on groups of interest, discussion title, name of asker, or status of discussion.

A ReachOut discussion is very similar to a conference chat room; however, it is persistent through time. This room has all the regular functionality found in other chat applications as well as some additional features required for the type of blended synchrony that ReachOut offers. Like in other chat room applications, users are aware of the other users who are in the same room with them; ReachOut, however, allows them to also see those who contributed to the discussion but are no longer in the room (see the names in black plain font in Figure 6). Likewise, the full history of the discussion is shown, regardless of the time the user joined it.

Timestamps, including date change where needed, are translated to reflect the viewer's time zone. ReachOut discussions persist over days, even when there is no user in the discussion room, and are closed by the system three working days[7] after the last append is made to them, that is, when they cease to evolve. These features enable a vibrant group discussion, supporting synchronous collaboration, while still allowing people from different time zones and schedules to take part in the discussion.

As in the case of newsgroups, ReachOut complies with the litmus test and is classified as a second-degree peer-to-peer system. However, the notification mechanism of ReachOut increases the chances of quality potential advisers to take part in the discussion, and the synchronous mode and light tone of discussion supply a more "modern" appropriate environment to foster fruitful discussions that conclude faster.

Exemplifying P2P Processes in ReachOut

All ReachOut activities, be they publicly written questions and responses or private activities such as entering discussions for reading, are stored in a log file. (Please see below.) This makes ReachOut an ideal laboratory for studying

A. raises the following question:

A.:

Where do I find an expert on EJB's on 390?

Four different people add their reply, giving different referrals:

B.:

Have you tried the eTZ folks in POL? I think Dan Bazil is the manager.

C.:

You can try Joe White out of Orlando. He may be able to assist depending on your need.

D.:

I do know that Diana Cole did run a JAVA competency group. She would probably be able to point you in the right direction.

E.:

The guy we use in the UK is Peter Benson, based in the Netherlands.

collaborative activities (Jacovi, Soroka, & Ur, 2003; Soroka, Jacovi, & Ur, 2003). For identifying P2P processes, though, the log file is not required, as discussion transcripts suffice for the location of cases where knowledge was exchanged and even created. Many ReachOut discussions fall under the classification of hybrid P2P process, where a person with an information need receives a reply (or several replies) that give direct answer or a referral for good sources of knowledge. The following example was presented in Ribak, Jacovi, and Soroka (2002).

ReachOut is full of such examples where people will locate people, even if the initial question is not an explicit search for an expert. This expert referral feature shows how people look for knowledge by exploring the peer network. As we can see in the above example, the proposed answers are not related. Though the asker can potentially benefit from the group discussion (for example, if other people reinforce an expert recommendation), essentially it is a hybrid P2P

Y. raises the following question:

Y. 7/4, 12:02 PM:

How to reschedule the antivirus client

She adds the following details:

Hate it when it runs on Monday noon as it just did and stucked my laptop. Heard that you can either download the server to change the settings, or twickle the registry. What's the fastest/easiest way?

Two minutes later, V. comes up with the following solution:

V. 7/4, 12:04 PM:

I know, I know! Go to c:\program FIles\NAVNT and run NAVWHEN.exe

Y. tries V.'s suggestion, but does not find the file:

Y. 7/4, 12:09 PM:

Nope, no Navnt dir on my PC (XP)... I have a C:\Program Files\Symantec_Client_Security\Symantec AntiVirus directory and there no Navwhen.exe

M. has a completely different suggestion:

M. 7/4, 12:10 PM:

Click on the properties of the app itself for the schedule

Y. 7/4, 12:13 PM:

Doesn't help either. I have no schedule in its config (seems to be run from the server???) do I make sense at all here?

A. is now curious to try V.'s suggestion, discovers the NavWhen.exe file on a different location. He contributes this info in the hopes that it helps Y.

A. 7/4 12:25 PM:

I have NavWhen in c:\Program Files\Symantec Client Security\Tools

Y. is willing to look in that directory as well

Y. 7/4, 12:26 PM:

let me try it

By viewing replies from different sources, V. realizes that different installations may put the file in different locations. He suggests to not look in the Tools directory, but rather to search for the file via Windows search

V. 7/4, 12:27 PM:

Maybe it appears in different directories in different installations. Try searching for Navwhen.exe

Trying this, Y. locates the files, runs it, and gets the results she was looking for.

Y. 7/4, 12:37 PM:

Hallelujah it seems it worked!! Thank you!

scenario. Each peer provides a piece of information he or she possesses. No other communication takes place between the members of the group.

There are many scenarios, though, that in fact reinforce the second-degree P2P model. Scenarios in which the group as a whole reaches a solution that no one participant could have contributed alone. The discussion in the table on the previous page is one such example (typos and grammar errors are left as in the original).

The fact that the discussion takes place publicly, as a group discussion, allows several people to raise different solutions, and then allows them all to inspect the series of trials, analyze them, and come up with a conclusion that actually solves the problem. Not only did Y. get her solution, but all participants—Y., V., M., and A.—as well as those who silently viewed the discussion, gained knowledge in the process. In fact this piece of knowledge was not in possession of any one of them prior to the discussion. The process that takes place within the discussion is a pure P2P one. The fact that the group discussion is facilitated by the ReachOut server makes it a second-degree P2P process.

Conclusion

King (2001) emphasizes that in order to understand any process researchers should look at the infrastructure of this process rather than at its visual problems. He advises to see the things that do not change, in order to understand the changes that happen and address them. While King talks about ubiquitous computing, his observation is valid for any form of research. Indeed, often the nature of any process lies in its very infrastructure, which was designed to support the very basic challenges and processes that arise from the nature of the process. This is why these infrastructures do not change and that is why they are interesting to look at.

Knowledge sharing is an ancient process. People have been exchanging information and knowledge since the dawn of mankind. In this chapter, we focused on the core process of knowledge exchange and analyzed it from the technological point of view by applying to it the principles of P2P computing. We have shown that the process of knowledge sharing is essentially a peer-to-peer process by itself. In real life, it is sometimes a pure P2P process, but much more often it cannot be effective without finding the right peers to share knowledge with. Thus, a hybrid P2P model suits the process of knowledge sharing better by introducing a way for expert location.

We noticed that the knowledge creation process can be significantly improved by turning to several advisers at once. We thus proposed a slightly different model for P2P that we called second-degree P2P model. In this model, after initially searching for the group of experts, creation of ad hoc groups is allowed, where in turn group members engage in a P2P knowledge sharing process. We have described ReachOut, a tool for peer support that was developed at the IBM Haifa Research Lab, which uses our model to practice knowledge sharing among work-based peers. We have provided an example discussion analysis to exemplify the second-degree P2P model knowledge sharing process.

As knowledge is a core resource in business-oriented collaboration, many tools will be created in the future to facilitate knowledge sharing. We suggest that before designing new tools for knowledge sharing, designers should study in depth the underlying process in a given setting and use those studies in their design. We argue that the second-degree P2P model we introduced for knowledge sharing is quite universal and can be used to get additional insight on knowledge flow networks inside any organization. Our experience with ReachOut shows that by using a well-defined underlying model of the supported process, more efficient tools can be created.

References

Ackerman, M.S. (1994). Augmenting the organizational memory: A field study of answer garden. *Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW '94)*, 243–252.

Ackerman, M.S., & McDonald, D.W. (1996). Answer garden 2: Merging organizational memory with collaborative help. *Proceedings of ACM Conference on Computer Supported Cooperative Work (CSCW '96)*, 97–105.

Ackoff, R.L. (1989). From data to wisdom. *Journal of Applies Systems Analysis, 16*, 3–9.

Adar, E., & Huberman, B.A. (2000). Free riding on Gnutella. *First Monday, 5*(10). from http://www.firstmonday.dk/issues/issue5_10/adar/index.html

Alexander, P.A., Schallert, D.L., & Hare, V.C. (1991). Coming to terms: How researchers in learning and literacy talk about knowledge. *Review of Educational Research, 61*, 315–343.

Amazon.com. http://amazon.com/

Cohen, E. & Shenker, S. (2002). Replication strategies in unstructured peer-to-peer networks. *Proceedings ACM SIGCOMM 2002*, 177–190.

eMule. http://www.emule-project.net/

Gnutella. http://gnutella.wego.com/

Granovetter, M. (1973). The strength of weak ties. *American Journal of Sociology, 78*(May), 1360–1380.

Groove. http://groove.net/

Groove Whitepaper (n.d.). From http://static.userland.com/gems/magistris/introducingGroove.pdf

IMDb. http://imdb.com/

Jacovi, M., Soroka, V., & Ur, S. (2003, November 9–12). *Why do we ReachOut? Functions of a semi-persistent peer-support tool*. Proceedings of the ACM International Conference on Supporting Group Work (GROUP '03), Sanibel Island, FL.

Kazaa. http://www.kazaa.com/

King, J.L. (2001). Everywhere is nowhere. Presented at Explorations within the Next Wave: The Design and Exploitation of Ubiquitous Computing Environments Workshop in Case Western University, Cleveland, OH, October 26–28, 2001. from http://weatherhead.cwru.edu/pervasive/2001/content/everywhere%20is%20nowhere.pdf

Kollock, P., & Smith, M. (1996). Managing the virtual commons: Cooperation and conflict in computer communities. In S. Herring (Ed.), *Computer mediated communication: Linguistic, social, and cross-cultural perspectives* (pp. 109–128). Amsterdam: John Benjamins.

Lv, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002). *Search and replication in unstructured peer-to-peer networks*. Proceedings of the 16th International Conference on Supercomputing, Linköping, Sweden.

Napster. http://www.napster.com/

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company*. New York: Oxford University Press.

P2PMessenger. From http://p2pmessenger.com/

Polanyi, M. (1966). *The tacit dimension*. London: Routledge & Kegan Paul.

Rafaeli, S., & Raban D.R. (2003). Experimental investigation of the subjective value of information in trading. *Journal of the Association for Information Systems (JAIS), 4*(5), 119–139.

Rheingold, H. (2000). *The virtual community: Homesteading on the electronic frontier*. Cambridge, MA: MIT Press.

Ribak, A., Jacovi, M., & Soroka, V. (2002). *"Ask before you search": Peer support and community building with ReachOut*. Proceedings of ACM Computer Supported Cooperative Work (CSCW 2002), New Orleans, LA.

Ripeanu, M. (2001, August). Peer-to-peer architecture case study: Gnutella network. *Proceedings of International Conference on Peer-to-Peer Computing*.

Sametime. http://lotus.com/products/lotussametime.nsf/wdocs/homepage

SETI@Home. http://setiathome.ssl.berkeley.edu/

Shirky, C. (2000, November 24). What is P2P… and what isn't. Retrieved July 13, 2003, from http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

Skype. http://www.skype.com/

Snowden, D. (2002). Complex acts of knowing: Paradox and descriptive self-awareness. *Journal of Knowledge Management, 6*(2), 100–111.

Soroka, V., Jacovi, M. & Ur, S. (2003). We can see you: A study of the community's invisible people through ReachOut. Proceedings of Communities and Technologies Conference.

Terveen, L., & Hill, W. (2001). Beyond recommender systems: Helping people help each other. In J. Caroll (Ed.), *HCI in the new millennium*. New York: Addison-Wesley.

Tiwana, A. (2003). Affinity to infinity in peer-to-peer knowledge platforms. *Communications of the ACM, 46*(5), 77–80.

Usenet. http://groups.google.com/

Wellman, B. (1997). An electronic group is virtually a social network. In S. Kiesler (Ed.), *Culture of the Internet* (pp. 179–205). Mahwah, NJ: Lawrence Erlbaum.

Zhao, B.Y., Kubiatowicz, J., & Joseph, A.D. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141. Berkeley, CA: University of California.

Endnotes

1 Phrase borrowed from Geoffrey A. Moore's book, *Crossing the Chasm*

2 This usage of P2P is also the most controversial due to the legal issues involved (such as copyright infringements), as widely discussed in the press.

3 There are also some advantages for a centralized directory server such as the ability for reciprocity management and efficient search.

4 Newsgroups (NNTP) server architecture is also P2P. Newsgroups server exchange information between each other and have backups if a certain server is down, thus creating a giant grid of information.

5 Resource virtualization, also known as Grid, is a concept of using multiple resources of the same kind as one virtual resource.

6 Note that expert referral can also be implemented in a pure peer-to-peer model, when peers refer the asker to a person who potentially has the relevant knowledge; they simply have less coverage than a central directory.

7 The expiration window is a configurable parameter that should be tuned to the environment. We fixed it to three working days in our corporate environment after careful analysis of the duration of dialogs.

Chapter XI

Ten Lessons from Finance for Commercial Sharing of IT Resources

Giorgos Cheliotis, IBM Research GmbH, Switzerland*

Chris Kenyon, IBM Research GmbH, Switzerland

Rajkumar Buyya, University of Melbourne, Australia

Abstract

Sharing IT resources within and among organizations is an attractive value proposition in terms of efficiency and flexibility, but despite this, commercial practice is limited. In contrast, financial and commodity markets have proved very successful at dynamic allocation of different resource types to many different organizations. Thus to understand how the potential benefits of sharing IT resources may be promoted in practice, we analyze enabling factors in successful markets. We present 10 basic lessons for IT resource sharing derived from a financial perspective and modified by considering the nature and context of IT resources. From each lesson we derive the *required software or process capability required to support it. We then evaluate the maturity of the respective capabilities within the peer-to-peer and grid environments using a simple framework based on the standard Capability Maturity Model approach. We conclude with a description of the largest capability gaps and the lowest hanging fruit for making IT resource sharing a more viable business proposition.*

Introduction

Sharing IT resources within and among companies is an attractive value proposition for many organizations in terms of efficiency and flexibility, but despite this, commercial practice is limited. The scope of potentially sharable IT resources includes computation, storage, and data. Network bandwidth has been shared for some time, but this is typically done without a defined quality of service. It will also be necessary to create appropriate packages of the different resources to be shared, but beyond the scope of this article. The context we are considering is large-scale sharing among separate budget entities, for example, within a large life-sciences company, an oil company, or a financial institution, or indeed, among them all. Common technical paradigms for enabling resource sharing have been established in terms of the peer-to-peer (P2P), the cycle-harvesting, and more generally, the whole grid movement. Whilst the value proposition for resource sharing may be compelling in the abstract, sharing is still at a rudimentary stage in practice.

Although business executives and IT managers would surely welcome the ability to extract more value from available resources, especially in light of shrinking IT budgets, they have been slow to adopt such practices, presumably because many of them are not yet convinced that these new sharing paradigms can deliver in practice. Technically, it is indeed possible to allocate resources as needed and to change this allocation on very short time scales. However, the ability to dynamically align resource allocations with changing business objectives is largely absent. Thus the principal reason for the slow commercial adoption of P2P and related technologies is that although such technologies enable sharing, they do not help an organization decide how to best allocate the resources it owns. In contrast, financial and commodity markets have proved very successful in terms of both scale and scope regarding the dynamic sharing and allocation of many different types of resources among many organizations.

Thus, to understand how the potential benefits of sharing IT resources may be realized in practice, we consider 10 lessons learned from the financial sector. From each lesson we derive one or more software or process capabilities required to support it. We then evaluate the maturity of the respective capabilities within the P2P and grid environments using a simple framework based on the standard Capability Maturity Model (CMM) approach from the Software Engineering Institute (2004).

We do not claim that the 10 lessons considered here and the respective capabilities we derive are exhaustive and definitive but rather that they have been fundamental for enabling commercial efficiency and flexibility in other resource-sharing environments. Thus, these lessons are important for management and engineers who aim to promote efficient and flexible usage and sharing of IT resources in a commercial context.

These lessons provide management with a checklist of the relevant business issues and required capabilities for the successful implementation of appropriate sharing strategies in their organization. Practitioners will appreciate the scope of the challenge remaining for business alignment of sharing technologies and be able to identify which issues to prioritize in practice.

The objective of this chapter is not to derive an academic research agenda, although there are clearly many opportunities in this field.

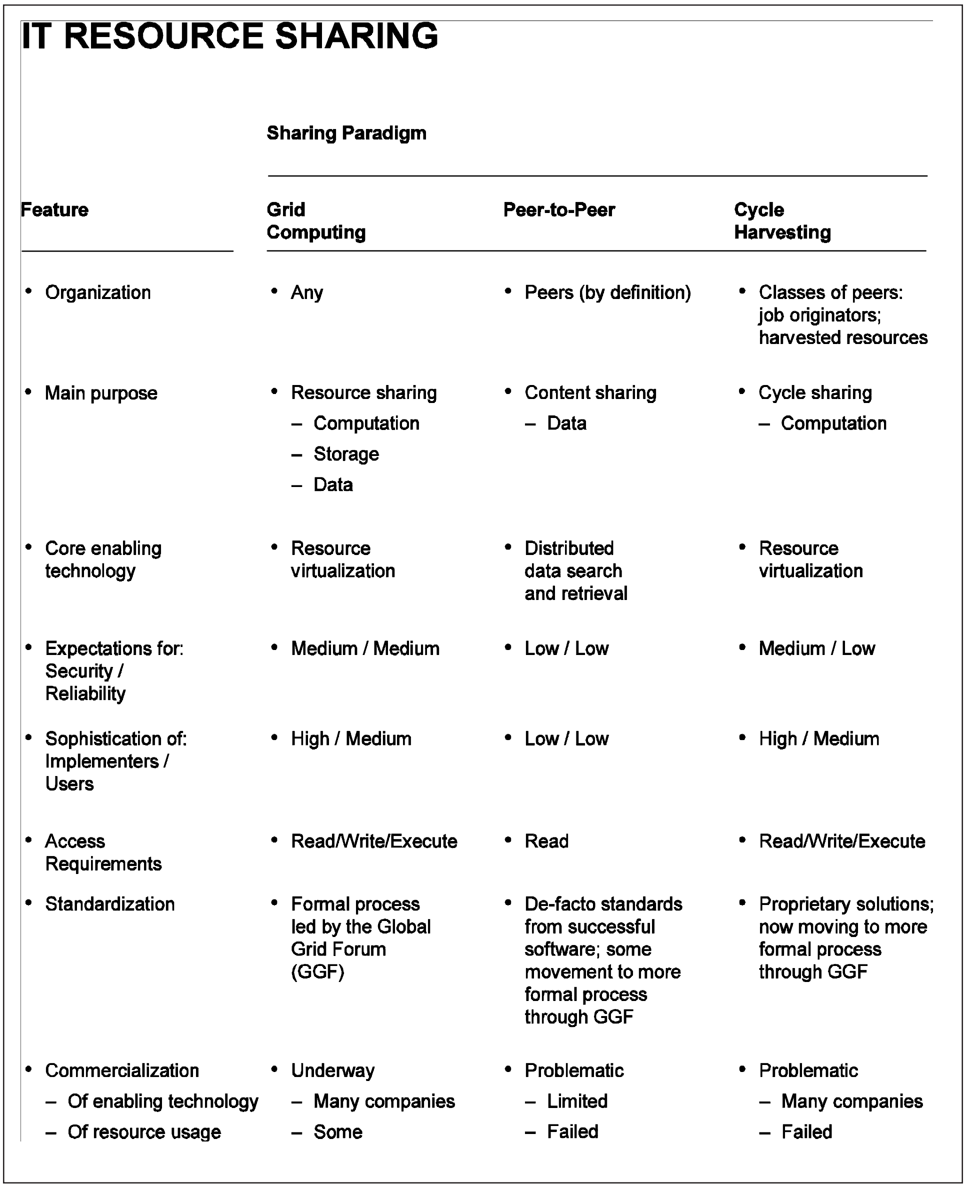
Grid, P2P, and Cycle-Harvesting: Three Converging Paradigms

Resource sharing can be implemented in many different ways, and several technical approaches, each with its own community, literature, software, and so forth (e.g., Forster, Kesselman, Nick, & Tuecke, 2002; Kamkar, Shahmehri, Graham, & Caronni, 2003; gnutella2.com; Thain, Tannenbaum, & Livny, 2002). Of these approaches, we identity three main movements: grid computing, P2P, and cycle-harvesting. These three movements have large areas of overlap, but given that all three terms are frequently used to describe similar but not identical technologies, we briefly discuss and compare them to set the stage for the rest of the chapter. A summary of this high-level comparison is shown in Table 1.

We compare resource-sharing paradigms according to the following categories: organization, function, enabling technology, user expectations, user sophistication, access requirements, and commercialization of enabling technology and of relevant resources. The category that deserves particular attention in the context of this chapter is the last, commercialization. As the other categories serve to illustrate that these technologies may be different but in essence refer and lead to the same objective, that is, large-scale resource sharing, we will rarely distinguish between them in this chapter.

Under the term *commercialization*, we consider the commercial success of companies selling enabling technologies, and separately the success of compa-

Table 1. Comparison of common paradigms for IT resource sharing



IT RESOURCE SHARING

	Sharing Paradigm		
Feature	**Grid Computing**	**Peer-to-Peer**	**Cycle Harvesting**
• Organization	• Any	• Peers (by definition)	• Classes of peers: job originators; harvested resources
• Main purpose	• Resource sharing – Computation – Storage – Data	• Content sharing – Data	• Cycle sharing – Computation
• Core enabling technology	• Resource virtualization	• Distributed data search and retrieval	• Resource virtualization
• Expectations for: Security / Reliability	• Medium / Medium	• Low / Low	• Medium / Low
• Sophistication of: Implementers / Users	• High / Medium	• Low / Low	• High / Medium
• Access Requirements	• Read/Write/Execute	• Read	• Read/Write/Execute
• Standardization	• Formal process led by the Global Grid Forum (GGF)	• De-facto standards from successful software; some movement to more formal process through GGF	• Proprietary solutions; now moving to more formal process through GGF
• Commercialization – Of enabling technology – Of resource usage	• Underway – Many companies – Some	• Problematic – Limited – Failed	• Problematic – Many companies – Failed

nies selling the resources themselves (internally or externally). Whilst some P2P software is or has been pervasive, for example, Napster or Kazaa, the business success of the companies supporting the software is less clear. In contrast, there are many companies successfully selling grid and cycle-harvesting software or offering a significant services business around these technologies, for example, Platform Computing, Entropia, United Devices, Sun, IBM, and others. In terms of sales of the resources themselves for money, there are some internal company examples in the grid space (although we cannot cite company names at this point). There have been a number of attempts to commercialize resources on a P2P or cycle-harvesting basis, but we are unaware of any significant successes.

From the comparisons in Table 1, it is clear that in terms of delivering resource-sharing capabilities, P2P and cycle-harvesting are functional subsets of the grid paradigm although they do not usually share the Global Grid Forum's technical standards at this point. The paradigms are often directed toward different market segments and there is little technical commonality in their most successful implementations (e.g., content sharing for P2P versus CPU/storage use by applications for grid). However, the high-level business propositions are basically the same and we can expect the functionality to merge with increasing commercial interest and further standardization. It follows that the 10 lessons we discuss are relevant for all three sharing paradigms.

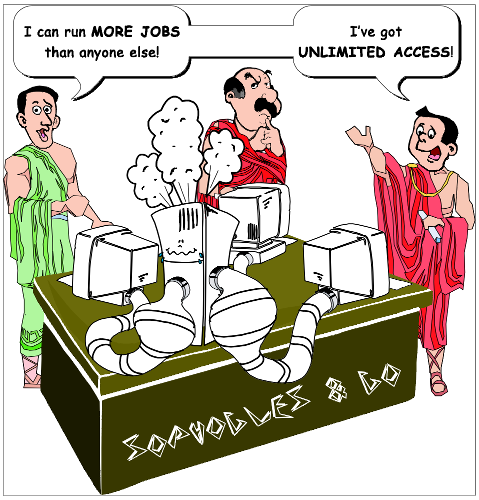
Ten Lessons from Finance

In this section, we describe each of the 10 lessons and the capability required to derive the benefit from each. In the following section, we assess the maturity of the respective capabilities in IT sharing systems.

Avoid the Tragedy of the Commons

Usage increases whenever there is an increased need, that is, when the marginal benefit to the user of utilizing an extra unit exceeds the cost of consuming that unit. In an uncontrolled or free-for-all situation this marginal cost may be very low. This is especially true in the context of IT resources (see Shapiro & Varian, 1999), and is potentially a source of trouble. If companies consider only individual user's budgets and preferences in determining resource value, they may neglect a very important factor: what economists call (network) externalities. An externality can be defined as the impact of one person's actions on the well-being of a bystander and it can be positive or negative.

Figure 1. The tragedy of the commons


I can run MORE JOBS than anyone else!
I've got UNLIMITED ACCESS!
SOPHOCLES & CO

This socioeconomic phenomenon whereby the individually "rational" actions of members of a population have a negative impact on the entire population is often called the tragedy of the commons (see Figure 1). Common recipes for dealing with this issue target the internalization of negative externalities into every individual's decision process. Mankiw (1997) and Shapiro and Varian (1999) state that this can be achieved by taxation, regulation (e.g., TCP congestion control), private solutions, or prices for access rights, for example, permits.

Shared IT infrastructure is particularly prone to negative externalities because there is currently no scalable and dynamic standard mechanism for limiting system (ab)use. Local priority rules are efficient only in returning grids to their pre-grid, that is, nonshared, state whilst free-access spaces suffer from the tragedy of the commons. Static policies are particularly inappropriate for dynamic virtual organizations and do not scale well as the number of participating entities increase. Pricing access to IT resources and permitting resale is a direct and scalable way to preclude such a tragedy of the commons for grid deployments that deserves serious consideration.

Discover Dynamic Value

Given even the most cursory awareness of conventional resources and commodities such as copper, electricity, and petrol (gas), it is clear that resource value at the wholesale level is dynamic. What is perhaps less clear to some casual observers is that resources on grids have dynamic value.

Value derives from a combination of need and scarcity. User needs are not constant; they change over time, and the changes also depend on the time scale and granularity of observation. During a project life cycle, a single user working on that project will have varying workloads in different phases of development. The number of projects that a user is involved in also changes with time. Needs are also driven by external and irregular events, for example, reactions to advertising campaigns, seasonality, requests for bids that require data analysis. Variations in user needs change resource value very little if the resources are not scarce, that is, if the capacity of the shared infrastructure is never exhausted. However this happy state is rarely present for users with computationally heavy applications.

After recognizing that the value of any given resource changes with time, the obvious question is how to discover this dynamic value at any point in time. One approach is to use a dynamic "price formation" mechanism. The mapping of needs to prices, called price formation, has no single solution, but there is an extensive body of work precisely on this topic: auctions (see Klemperer, 1999). Whilst prices must be fed back to users (see next lesson), there is no corresponding need for the price formation mechanism to be visible to users. This can be handled for the most part by automated software, but a mechanism is still required and there are significant design challenges for it.

The lesson from auction theory and practice is that effective price discovery is difficult: the choice of price formation mechanism can either promote market efficiency or hamper it. Generally, it is difficult to achieve a balance between the needs of producers and consumers. Recent examples that illustrate this difficulty very well are the 3G mobile telephony spectrum auctions of Klemperer (2002) and von Weizsäcker (2003). High-profile auctions for 3G licenses have been carried out in many European countries. Two distinct problems arose in these auctions: bidder busts ("winner's curse") and auctioneer flops. 3G auctions in Germany and the UK yielded enormous profits for the local authorities at the expense of the bidders, whereas in Switzerland, The Netherlands, Italy, and Austria, prices remained well below expectations, disappointing the respective auctioneers.

In IT resource sharing, we want to avoid the winner's curse. Moreover, these resources are perishable (capacity not used now is worthless in the next moment), needs are dynamic and applications require bundles with multiple units of items (CPU, RAM, permanent storage, network bandwidth). Krishna (2002) states that with these conditions in mind, potentially suitable auction models for IT resources include continuous double auctions, Vickrey, Dutch, multiunit, and multi-item (or combinatorial) auctions. However, individually these approaches alone do not offer a comprehensive and precise price formation solution. The optimality of an auction mechanism will always depend on the particular deployment environment; there are no one-size-fits-all solutions.

Communicate Dynamic Value

Should dynamic value, that is, price, be communicated to users, or should value only be used internally by the resource-sharing system to produce allocations of resources to users? Should users (or their respective managers) pay only a fixed subscription fee for accessing a shared resource space? Isolating users from price dynamics makes sense when users never see—or cause—scarcity, that is when they have low and uncorrelated needs. For example, bread price dynamics at supermarkets have little relation to corn futures markets. On the other hand, electricity companies seek methods to pass intraday price dynamics on to consumers because of the enormous loads consumers produce through correlated responses to events (e.g., extremes of temperature) even though each individual consumes little relative to the capacity of an electricity generator.

Most users of grid infrastructures are heavy resource consumers almost by definition, so dynamic prices must be communicated at some level. Fixed pricing may only make sense in a limited number of cases, for example, in pure P2P file sharing environment with a large population of uncorrelated user demands.

Use Real Money

An issue that concerns the grid community is the definition of a grid currency (see Barmouta & Buyya, 2003). This issue is generally more important for commercial IT-sharing environments that cover many different budget entities than for earlier distributed systems that did not have a clear notion of, or connection with, budget entities. In addition, managers will face the issue of whether they should buy resources on accessible shared spaces or boxes. Managers will also need to decide whether, and how, to make their boxes available to the shared spaces to which their organization is linked. Shared IT resources are typically heterogeneous and potentially of arbitrary scale. Scale and heterogeneity are exactly the drivers which led to the establishment of standard monetary units and currency exchange rates in the real economy.

The administration of a particular shared space may choose to introduce prices for a local artificial currency. The administration must then act as a national bank by guaranteeing the convertibility of the currency into units of value, that is, resources or real money. Now who sets the exchange rates and to which unit of value? A currency board? A fixed exchange rate? IT administrations should quickly choose to skip the intermediate step of an artificial currency with its trust and convertibility problems and use real money straight away. Using a real currency for shared resources additionally brings the following benefits: buy/build/lease or upgrade/retire decisions are simplified and the allocation of IT budgets is directly meaningful.

Guarantee Property Rights

What quality of service (QoS) is required for tradable value and convertibility? Most IT systems today do not support hard QoS guarantees, that is, they do not guarantee specific properties of a service to the user. Often best-effort service is provided. Approaches that go beyond best-effort typically introduce job/packet marking so that different priorities can be assigned to different tasks (Blazewicz, Eaker, Pesch, Schmidt, & Weglarz, 1996; Ferguson & Huston, 1998). How much better the service will be for differentiated service classes is generally hard to determine in advance for large-scale heterogeneous systems and even harder to characterize in absolute terms.

Despite the difficulties of guaranteeing QoS (especially end-to-end), commercialization of shared IT resources requires guaranteed property rights at the level at which pricing is done. Best-effort service has near-zero economic value. In fact the value would be exactly zero if it were not for the assumption that there is a common understanding between the buyer and seller of the service on the quality level to be delivered (see Figure 2).

Advocates of IT resource sharing envision dynamic near-real-time negotiation and provisioning of distributed resources (Benatallah, Dumas, Sheng, & Ngu, 2003). This vision may appear very ambitious at first sight, but it is actually very similar to existing financial and commodity markets. Such markets typically operate at electronic speed and rely on the use of extremely detailed processes and contracts to determine the allocations of large, heterogeneous sets of resources to an equally large and heterogeneous population of users. Complexity is no barrier to value for a good. The definitions of some resources traded on the

Figure 2. Best effort

"BEST EFFORT" means we'll do our BEST to serve you, but no guarantees. Any questions?
Well ... I just wonder whether that means I can make "BEST EFFORT" payments?

Chicago Mercantile Exchange (CME) run for many pages, and even then, reference external tests and standards (see The Chicago Mercantile Exchange, 2004; The Chicago Mercantile Exchange Rulebook, 2004).

Computers and applications may be complex but they also have unambiguous definitions. A high level of detail in contract specifications and a hard guarantee regarding these specifications are necessary elements to create the appropriate confidence among users of a highly distributed cross-organizational system that what they get is exactly what they expected to receive. In some cases, a tradable asset must be described in statistical terms, but it is still feasible to provide hard guarantees in this sense. This has been applied to cycle-harvesting (see Kenyon & Cheliotis, 2003).

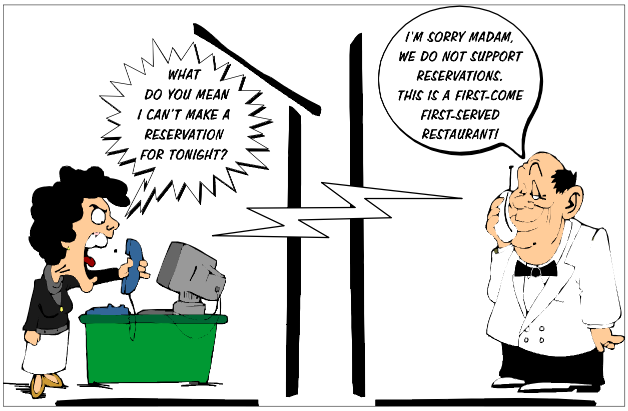
Use Futures Markets

IT resources are generally not storable, in the sense that capacity not used today cannot be put aside for future use. Since the resources cannot be stored, there need be no link between the price for a resource now and the price that the resource (if available) will fetch at any future time (even in 1 second!). Given that it is impossible to build up inventories to smooth out the differences between supply and demand, prices can be arbitrarily volatile (this has been observed in practice for other nonstorable commodities by Pilopovi'c [1998]). To avoid this price volatility and to enable planning and risk management, conventional nonstorable commodities have developed futures markets, that is, markets for reservations.

The most significant non-IT commodity that is also nonstorable is electrical power (with the notable exceptions of hydroelectric and pumped storage). In electricity markets, as in several others for nonstorables (e.g., live cattle, interest rates), contracts for future delivery (forward or futures contracts) are the most used and have much higher trading volumes than those for immediate delivery (spot contracts). The notable electricity market failure in California was directly tied to poor use of forward contracts (see California ISO [2002]; see Hull [2003] for an introduction to futures markets). The experience of electricity markets is clear (e.g., California, UK) and has led to their redesign with the aim to move everything possible off the spot market and onto the futures markets.

IT resources—like any other resource—have dynamically changing value. Given that they are not storable, availability cannot be guaranteed without a formal mechanism, that is, reservations. This availability guarantee acts as a substitute for inventories and helps to prevent unwanted excessive value fluctuations.

Figure 3. No reservations?


WHAT DO YOU MEAN I CAN'T MAKE A RESERVATION FOR TONIGHT?
I'M SORRY MADAM, WE DO NOT SUPPORT RESERVATIONS. THIS IS A FIRST-COME FIRST-SERVED RESTAURANT!

The fact that there is work within the Global Grid Forum for supporting advance reservations is encouraging (Roy, 2002; see also Figure 3) since this capability is vital in commercial practice for nonstorable commodities. Some broadly used schedulers also have reservation (or deadline) capability (e.g., Maui's Advanced Reservation interface) and reservations are common practice in supercomputer environments. However, linking reservations to value and exchangeable reservations between users is largely missing.

Create Incentives

In the period 1999–2001, more than 1,000 new Internet-based business-to-business (B2B) exchanges were set up, according to IDC. Almost all of them failed, not because markets have poor theoretical properties, but because their specific value proposition was unconvincing. This was made manifest in "low liquidity," that is, no activity.

The success stories in B2B markets are primarily in specialized national or regional exchanges for electricity (e.g., NordPool). These usually have something in common: regulation accompanying persuasion to get people to sign up. Some of the results may be good for everyone, but the startup adoption costs must still be paid. The other area of outstanding B2B exchange success is the traditional financial and commodity markets with their vast turnover. Here the startup costs were paid long ago.

Where does IT sharing and exchange fit into this spectrum of experience? A detailed answer to that question is beyond the scope of this chapter, but certainly the value proposition of cost savings and greater flexibility is generally accepted. Commoditization is also accepted: there are many fewer computer flavors than there are different companies on, say, the New York Stock Exchange. In addition to stocks, a wide variety of standard instruments are traded on commodity, FX, and derivatives exchanges.

A company can decide to migrate to an internal market in the same way that it can decide to outsource. This is an executive decision to be made on business grounds: units may protest for whatever reason, but the needs of the business are the deciding factor. This is equivalent to regulation in the case of electricity markets mentioned above.

Public IT resources exchanges are unlikely in the short to medium term because of complexity of implementation and of the required changes in business processes. Within a single company or a closed group, the prospects for having appropriate incentives to overcome the startup costs and general inertia are much brighter.

From the examples above, it is clear that practical incentives are vital to promote resource sharing and that incentives are a meta-capability. That is, softwares and systems can support incentive mechanisms but incentive design occurs at a higher business level: the strategic level.

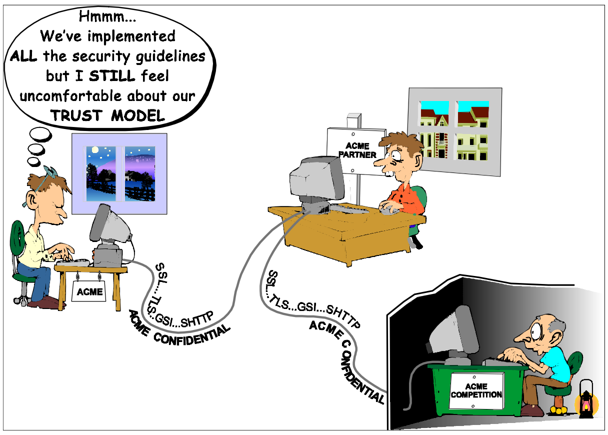
Ensure Trust

What is a good trust model for IT resource sharing? We note that trust is different from security and we are concerned here with trust. Security is just one enabler of trust (see Figure 4).

A good trust model for an online bookstore is not the same as a good trust model for a financial exchange. In fact, online bookstores effectively outsource their trust model to credit card companies for the most part. All the bookstore has to do is provide a basic level of security.

A financial exchange such as the CME has a more complex trust model. First, all transactions on the exchange between different parties are guaranteed by the exchange, not by the individual parties. Thus, the traders only need to trust the exchange to do business, not each other. On the other hand, the exchange trusts the traders because it monitors their actions and requires them to provide a (cash-equivalent) deposit, which is a function of the risk that each trader represents to the exchange for default. That is, the exchange trusts the traders because it has their money. All other people wishing to trade must do so via the traders. Thus, we see a two-tier trust model with distributed risk.

Figure 4. Trusted or just ... secure?



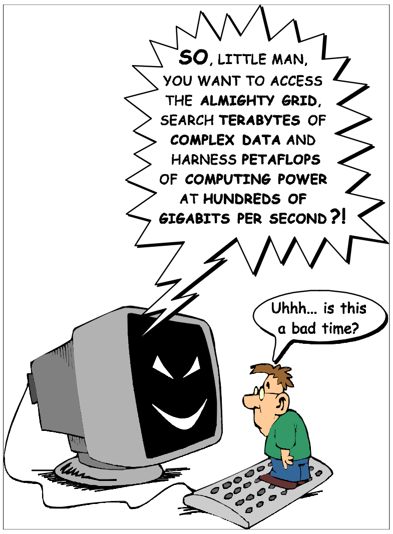
Systems without proportional consequences do not engender trust: this is why contracts exist and are specific and clear methods to invoke financial and legal penalties for improper actions. IT-sharing paradigms require trust models adapted to their environments (e.g., single company, group, etc.). The capability required to deliver trust is a system of proportional consequences for breaches of trust, both in terms of magnitude and, more important, time scale.

Support Process Tools

A typical portfolio directed by a fund manager can easily run to hundreds of stocks selected according to maximizing a specific objective and limited by equally precise constraints on number of stocks to hold, position limits, cash flow obligations that must be met on specific dates, hedging against worst-case scenarios, and so forth. Financial firms do not optimize their resource allocations (portfolios) by hand. They use sophisticated processes to support their decisions; there is an extensive literature that has grown up since the original work on portfolio theory more than 40 years ago by Markowitz (1959) and Birge and Louveaux (1997).

The dynamic system enabled and embodied by resource-sharing paradigms for a typical large company is a significant challenge for users to be able to exploit efficiently and economically. Without sophisticated tools users will find that the more potentially useful and flexible the system is, the less practically usable it will be (see Figure 5). The scarcest resource for many users is attention: they already

Figure 5. User in need of process tools


SO, LITTLE MAN, YOU WANT TO ACCESS THE ALMIGHTY GRID, SEARCH TERABYTES OF COMPLEX DATA AND HARNESS PETAFLOPS OF COMPUTING POWER AT HUNDREDS OF GIGABITS PER SECOND?!
Uhhh... is this a bad time?

have jobs and do not want to be bothered with unnecessary details of resource allocation systems.

The process tools for resource allocation must enable users to express their resource needs in a simple way together with the users' uncertainties about these needs over time. They should also enable resource trading (buying and selling of blocks of time and capacity) and capture the effective price dynamics of both spot and futures prices together with changing availabilities. However, the users should not be bothered by these details. Building such tools which integrate budgets and business objectives with resource allocation decisions may seem overly ambitious but it is a situation that is tackled every day for fund managers balancing opportunities and obligations over time. The technical area that these tools build from is multistage stochastic optimization that is generally applicable to large-scale resource systems whether the underlying resources are financial or IT (see Neftci, 2000; Markowitz, 1959; Birge & Louveaux, 1997).

Design for Value

When making a business case for the adoption of a resource-sharing technology, one commonly involves the following arguments: increased utilization, cost savings, greater allocation flexibility, feasibility of previously impossible computational tasks, and so forth. These benefits may be theoretically possible, but to what extent can an organization practically realize these potential values?

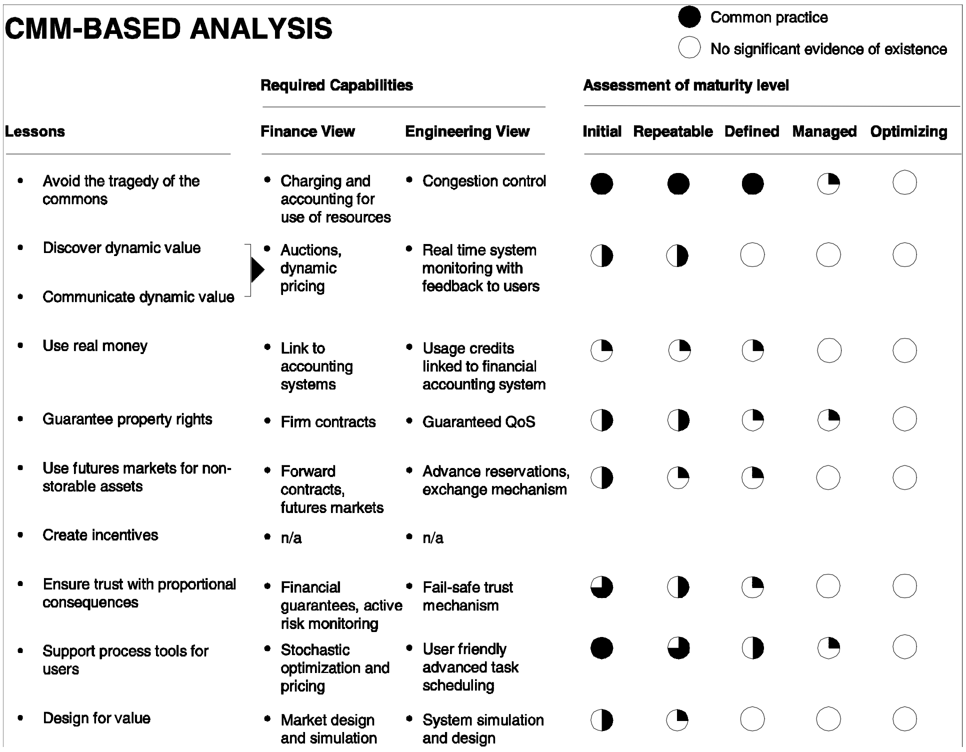
To achieve economic objectives, laissez-faire alone is not enough. As mentioned in the previous section on futures markets, a market and resource product structures require engineering to achieve results. Economic engineering for resource sharing covers two main areas: market design and product design. The following questions are just a small selection of the economically significant market design issues. Should the IT department of an organization be operated as a profit or as a cost center? How much reselling of resources should be allowed? Should short-selling be allowed? Is speculation permitted? What level of financial and project risk are users and departments permitted to take?

Product design is just as important as market design in practical systems. For simplicity, it can be important for most of the usual user and manager needs to be available directly as products rather than having to be built each time they are needed. Spot and forward contracts (reservations) can be useful decompositions for describing and controlling the theoretical basis of value. Alternatively these contracts can be automatically assembled into products to match user, application, and department requirement profiles using process tools. For example, a requirement profile could be described using a stochastic process, thus capturing both variability over time and the uncertainty of this variability. Birge & Louveaux (1997) state that sets of resource requirements could be expressed as a portfolio of liability profiles and resources allocated through the application of standard multistage stochastic portfolio optimization techniques. However the markets and the products are designed, they need to be crafted to ensure that the maximum value is realized from the underlying resources and business context. This design capability is vital to deliver confidence to executive decision makers for implementation and subsequent improvements to their bottom line in practice.

Maturity of Proposed Capabilities

We qualitatively assess the maturity of the capabilities required to benefit from each of the lessons using a five-stage framework derived from the CMM (see Software Engineering Institute [2004]). This is not a formal quantification but rather the personal appreciation of the authors derived from our experience and discussions around the tutorial on this subject we gave at GGF5 (see Cheliotis & Kenyon [2002]). There are two differences between a typical use of CMM and our approach: (a) in practice, CMM analysis often focuses on the capabilities of a particular organization whereas we assess maturity levels in the broader field of IT sharing and (b) CMM levels typically refer to the maturity of software development processes whereas we focus on the maturity of IT-sharing processes/capabilities derived from our 10 lessons. We, therefore, need to redefine these levels for the context of this chapter. CMM defines five levels

Table 2. Framework for assessing capability levels



CMM-BASED ANALYSIS

● Common practice
○ No significant evidence of existence

	Required Capabilities		Assessment of maturity level				
Lessons	**Finance View**	**Engineering View**	**Initial**	**Repeatable**	**Defined**	**Managed**	**Optimizing**
• Avoid the tragedy of the commons	• Charging and accounting for use of resources	• Congestion control	●	●	●	◔	○
• Discover dynamic value • Communicate dynamic value	• Auctions, dynamic pricing	• Real time system monitoring with feedback to users	◑	◑	○	○	○
• Use real money	• Link to accounting systems	• Usage credits linked to financial accounting system	◔	◔	◔	○	○
• Guarantee property rights	• Firm contracts	• Guaranteed QoS	◑	◑	◔	◔	○
• Use futures markets for non-storable assets	• Forward contracts, futures markets	• Advance reservations, exchange mechanism	◑	◔	◔	○	○
• Create incentives	• n/a	• n/a					
• Ensure trust with proportional consequences	• Financial guarantees, active risk monitoring	• Fail-safe trust mechanism	◕	◑	◔	○	○
• Support process tools for users	• Stochastic optimization and pricing	• User friendly advanced task scheduling	●	◕	◑	◔	○
• Design for value	• Market design and simulation	• System simulation and design	◑	◔	○	○	○

which we adopt, label, and (re)define below: initial, repeatable, defined, managed, and optimizing.

1. Initial: Basic elements of the capability have been observed in practice.
2. Repeatable: it is possible to reliably repeat the implementation of the capability, and there is a common body of work supporting the understanding of the capability.
3. Defined: There is a defined process for implementing the capability and industry or de facto standards exist.
4. Managed: The processes of implementing and maintaining the capability can be managed in terms of time-varying desired objectives and requirements.
5. Optimizing: The processes of implementing and maintaining the capability can be continuously optimized in terms of the desired objectives and requirements.

Our assessment is given in Table 2, which can be used to identify gaps in current capabilities, as a management checklist for implementation features, and to help practitioners prioritize new technical developments. We will give brief comments on each of the capability assessments but not an in-depth review, as this would be beyond the scope of the current chapter.

- Avoid the tragedy of the commons. Congestion of commonly accessible resources is well recognized, and current grid and cycle-harvesting software have support for increasingly sophisticated policies covering resource usage (e.g., in Sun's Grid Engine or in Platform Computing's LSF products). However, there is still very little support for policy design in terms of desired outcome versus stochastic demand models (which are not supported either). In this sense sharing software is well behind, say, network design tools.
- Discover and communicate dynamic value. Price discovery mechanisms are almost absent from the IT resource sharing space. Whilst top-down allocation is possible via policies, the ability for users to express their urgency and for systems to provide real-time feedback has only been seen in some research demonstrations.
- Use real money. The authors are aware of only a very small number of resource-sharing systems where resource usage (or reservation) is directly tied to accounting systems (although we cannot mention company names at this point), but this is a long way from common practice. Selling resources, for example, supercomputer time, for money is a well-established practice, as is monitor-and-bill for variably priced outsourcing contracts. This knowledge and practice is not yet integrated into resource-sharing software.
- Guarantee property rights. Guaranteed QoS for allocated resources is present on very few systems, mostly mainframes and supercomputers (where this is provided in the operating systems, for example, IBM zOS for z900 series, or alternatively provided at a machine-level granularity). Policy-based systems can generally support a basic version of property rights for certain privileged users. Management of rights in order to achieve objectives is largely absent.
- Use futures markets. The first operational futures market in computer time was a manual system in 1,968 states (Sutherland, 1968), but little has been implemented in practice since then apart from certain research demonstrations. Ernemann and Yahyapour (2004) state that there is some understanding of how to run futures markets for IT resources but this is far from common practice.

- Create incentives. The incentive structure around resource sharing is, in our view, a meta-capability in that it is implemented based on the careful design of a business model. Incentives are not inherent in software/system mechanisms as the many hundreds of failed business-to-business and business-to-consumer auction sites demonstrated in the dotcom era.
- Ensure trust. In a commercial environment, trust is based on legal and financial consequences, both immediate (penalties) and deferred (reputation, litigation). Whilst there is a body of work around reputation in agent economies, for example, Xiong and Liu (2003), this has yet to be applied to resource sharing except in some limited P2P examples emphasizing fairness. There is no equivalent of the real-time, risk-adjusted margin accounting of financial exchanges (see Section "Ensure Trust").
- Support process tools. The need for user-friendly interfaces is well understood in all IT-sharing paradigms with a range of answers from portals to dedicated software. However, the process tools contained in the interfaces are generally limited, especially in terms of achieving business objectives.
- Design for value. Both IT professional and financial experts understand the need to design systems to achieve financial goals and there are some, generally proprietary, tools to assess the return on investment from installation of different IT-sharing technologies (e.g., from IBM and Platform Computing). However there is no generally accepted process for this assessment and certainly no tools available for designing how a sharing system should run in terms of financial objectives.

Summary and Conclusions

Past developments in the IT resource-sharing field have been primarily technology driven. In those cases where needs were the driving force, it has been mostly the needs of scientists (grids) or home users (P2P). This modus operandi has produced a plethora of inventive-sharing mechanisms, but corporate IS managers still do not know how their organizations can derive value from these technologies. This should not come as a surprise, as resource-sharing mechanisms per se are indeed of little business value in the absence of tools that will assist companies with the implementation of a suitable sharing strategy.

From our analysis of IT-sharing software and systems in Table 2, it is clear that there is a large gap between current practice and the potential for value in this area. By value we mean the realization of potential business benefits from the efficient use and dynamic allocation of resources. The biggest gaps are around the quantification of the value and designing systems so that maximum value is achieved in practice. That both are missing together is not surprising: it is hard to maximize value if you have no means of making it visible and quantifying it. Thus, the lowest-hanging fruit is in making value visible to users and allowing users to express their urgency back. Once value can be seen, then it can be maximized, and the other lessons regarding design (i.e., use of reservations) and the use of real money can be applied in a meaningful way, assuming that a viable incentive structure is in place.

References

Barmouta, A., & Buyya, R. (2003). Gridbank: A grid accounting services architecture (gasa) for distributed systems sharing and integration. *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'03)*, 245.

Benatallah, B., Dumas, M., Sheng, Q.Z., & Ngu, A.N.H. (2002). Declarative composition and peer-to-peer provisioning of dynamic web services. *Proceedings of 18th International Conference on Data Engineering*, 297.

Birge, J. & Louveaux, F. (1997). *Introduction to stochastic programming*. New York: Springer.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Weglarz, J. (1996). *Scheduling computer and manufacturing processes*. Springer-Verlag.

California ISO. Amendment 44 Docket Nos. EL00-95-001 and ER02-1656-000 (Market Design 2002). Retrieved March 4, 2004, from http://www.caiso.com/docs/2002/05/29/200205290858531076.html

Cheliotis, G., & Kenyon, C. (2002, July 25). Grid economics. *Tutorial, Global Grid Forum 5, Edinburgh*. Retrieved March 4, 2004, from http://www.zurich.ibm.com/grideconomics/refs.html

The Chicago Mercantile Exchange. Retrieved March 4, 2004, from http://www.cme.com

The Chicago Mercantile Exchange Rulebook. Retrieved March 4, 2004, from http://www.cmerulebook.com

Ernemann, C., & Yahyapour, R. (2004). Applying economic scheduling methods to grid environments. In J. Nabrzyski, J. Schoph, & J. Weglarz (Eds.), *Grid resource management: State of the art and future trends* (pp. 491–507). Kluwer.

Ferguson, P., & Huston, G. (1998). Quality of service on the Internet: Fact, fiction or compromise? *Proceedings of Inet 98*. Retrieved March 4, 2004, from http://www.isoc.org/inet98/proceedings/6e/6e_1.htm

Foster, I., Kesselman, C., Nick, J., & Tuecke, S. (2002, June 22). The physiology of the grid: An open grid services architecture for distributed systems integration. Retrieved March 4, 2004, from http://www.globus.org/research/papers.html#OGSA

gnutella2.com. The Gnutella 2 specification. Retrieved March 4, 2004, from http://www.gnutella2.com/tiki-index.php? (page=Gnutella2)

Hull, J. (2003). *Options, futures, and other derivatives* (5th ed.). Prentice Hall.

Kamkar, M., Shahmehri, N., Graham, R.L., & Caronni, G. (2003). Third international conference on peer-to-peer computing (P2P'03).

Kenyon, C., & Cheliotis, G. (2003). Creating services with hard guarantees from cycle-harvesting systems. *Proceedings of CCGrid 2003*.

Klemperer, P. (1999). Auction theory: A guide to the literature. *Journal of Economic Surveys, 13*(3), 227–286.

Klemperer, P. (2002). How (not) to run auctions: The European 3G telecom auctions. *European Economic Review*. Retrieved March 4, 2004, from http://www.paulklemperer.org

Krishna, V. (2002). *Auction theory*. New York: Academic Press.

Mankiw, N. (1997). *Principles of economics*. Dryden Press.

Markowitz, H. (1959). *Portfolio selection: Efficient diversification of investments*. New York: John Wiley & Sons.

Neftci, S. (2000). *An introduction to the mathematics of financial derivatives* (2nd ed.). New York: Academic Press.

Pilopovi´c, D. (1998). *Energy risk: Valuing and managing energy derivatives*. New York: McGraw-Hill.

Roy, A., & Sander, V. (2003, April 30). Advance reservation API. GGF draft. Retrieved March 4, 2004, from http://www.gridforum.org

Shapiro, C., & Varian, H.R. (1999). Information rules: A strategic guide to the network economy. Boston: Harvard Business School Press.

Software Engineering Institute. Capability maturity model. Retrieved March 4, 2004, from http://www.sei.cmu.edu/about/about.html

Sutherland, L. (1968). A futures market in computer time. *Communications of the ACM, 11*(6).

Thain, D., Tannenbaum, T., & Livny, M. (2002). Condor and the grid. In F. Berman, G. Fox, & T. Hey (Eds.), *Grid computing: Making the global infrastructure a reality*. New York: John Wiley & Sons.

Vishnumurthy, V., Chandrakumar, S., & Sirer, E.G. (2003). Karma: A secure economic framework for peer-to-peer resource sharing. *Workshop on Economics of Peer-to-Peer Systems. Berkeley. CA*. Retrieved March 6, 2004 from, http://www.sims.berkeley.edu/research/conferences/p2pecon/papers/s5-vishnumurthy.pdf

von Weizsäcker, C.C. (2003). The Swiss UMTS auction flop: Bad luck or bad design? In H. Nutzinger (Ed), *Regulation, competition, and the market economy* (pp. 281–293). Göttingen, Germany: Vandenhoeck & Ruprecht.

Xiong, L., & Liu, L. (2003). A reputation-based trust model for peer-to-peer e-commerce communities. *Proceedings of IEEE Conference on E-Commerce (CEC'03).*

Author Note

* With McKinsey and Company at the time of publication.

Chapter XII

Applications of Web Services in Bioinformatics

Xin Li, University of Maryland Baltimore, USA

Aryya Gangopadhyay, University of Maryland Baltimore, USA

Abstract

This chapter introduces applications of Web services in bioinformatics as a specialized application of peer-to-peer (P2P) computing. It explains the relationship between P2P and applications of Web service in bioinformatics, states some problems faced in current bioinformatics tools, and describes the mechanism of Web services framework. It then argues that Web services framework can help to address those problems, and gives a methodology to solve the problems in terms of composition, integration, automation, and discovery.

Introduction

P2P Web Services for Bioinformatics

The idea behind Web services is to enable one computer program to call another computer program automatically through the Internet, and its basic advantages are integration and automation. The features of the Web services framework can be used to solve some of the current problems faced in bioinformatics, specifically, integration and automation. (We will discuss the details of both current problems faced in bioinformatics and Web services in the following sections.) In the integration process, we can combine different types of bioinformatics tools scattered over the Internet into one comprehensive set of Web services. This implies that different types of bioinformatics tools can call each other and get final results. In the process of calling each other, those bioinformatics tools (or, services in Web services framework) are integrated together. It is obvious that these bioinformatics tools (services) are peers. Hence, this mechanism for combining different bioinformatics tools is a specialized application of peer-to-peer computing (P2P).

Automation is also needed when people try to minimize human interference in using bioinformatics databases locally. The basic idea behind automating the retrieval and analysis of data in bioinformatics using the Web services framework is integrating the bioinformatics tools and laboratory information system together, and performing data input and output between these two systems automatically. It may seem hard to view laboratory information systems and the bioinformatics tools as peers because most common models view this relationship as client/server, in which laboratory information systems only request information from the services (bioinformatics tools). However, it is usual for biologists to upload their research results (bioinformatics data) to bioinformatics database tools, and ideally they would like to use their lab information systems to submit their results. Therefore, lab information system is not only pure data retriever but also sometimes data provider. So one can view the relationship between lab information system and bioinformatics tool as peer-to-peer.

From the above explanation, we can conclude that the Web services application on bioinformatics is a specialized application of peer-to-peer application. In the following two sections, this specialized application will be discussed in detail from two perspectives: bioinformatics and Web services.

Current Problems in Bioinformatics

The number of researchers in biology who regularly use tools to search and analyze Web-based bioinformatics data sources (such as NCBI, EMBL, etc.) is increasing at a rapid pace. However, several problems surface when using such data. For example, most of these tools are separated from each other as isolated islands having their own distinct interfaces and data structures. If a biologist wants to use several different types of services (such as retrieving a gene sequence and doing a BLAST search) during research, or compare search results from different data providers, she/he will have to perform the same procedures several times.

Furthermore, even for the same type of bioinformatics tool (e.g., BLAST), there are differences in implementation and data storage. Thus, it is necessary to synchronize among these databases to avoid repeated works.

Third, nearly all currently available bioinformatics tools need human interactions for data analysis. Hence, researchers need to download data sets from the Web site of each service provider and put them into their own system before they can upload their results to other tools at another Web site. This back-and-forth human computer interaction obstructs the in-depth data analysis and consumes valuable time of researchers. Also, service discovery is another major challenge in this field.

Web Services

In general, Web services framework is a natural consequence of the evolution of the traditional Web, which can be simply described as "machine read and processed Web," that is, Web services framework is concerned with the problems of enabling systemic application-to-application interactions over the Internet. The use of Web services on the World Wide Web is expanding rapidly as the need for application-to-application communication and interoperability is growing fast.

Web service is a developing concept that has been defined in multiple ways. However, a core feature of a Web service requires that it should be described by an XML-based description language and invoked by an XML-based message. The W3C (World Wide Web Consortium) defines a Web service as a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. The definition of a Web service can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols. At present, the most common description language is WSDL (Web Service Description Language), and the most common communication message is SOAP (Simple Object Access Protocol).

Based on W3C's definition of Web services and its most common application at present, we can conclude that Web services framework is a concept of systematic application-to-application interaction, and the basic idea is that one application is described by its WSDL file, and other applications (stub program) call it automatically by using SOAP messages after they "know" the described application's structure by "reading" its WSDL file. The four major advantages of Web services are discovery using UDDI (Universal Description Discovery and Integration) registry, composition, integration, and reduction of human interaction.

The Web services architecture involves many layered and interrelated technologies. There are many ways to visualize these technologies, just as there are many ways to build and use Web services. Figure 1 provides an illustration of some of these technology families. Web services are built on the basis of existing industry standardization—XML, as described above, both description language and communication message are based on XML standards such as WSDL and SOAP. WSDL is used to describe a Web service, and SOAP is used as the communication media between Web service and the caller (stub program).

There is another important role in the Web services framework called *Web services registry*, which is a kind of reservoir storing Web service systematic information. It can help users or stub program to find their desired Web service

Figure 1. Web services technologies stack diagram

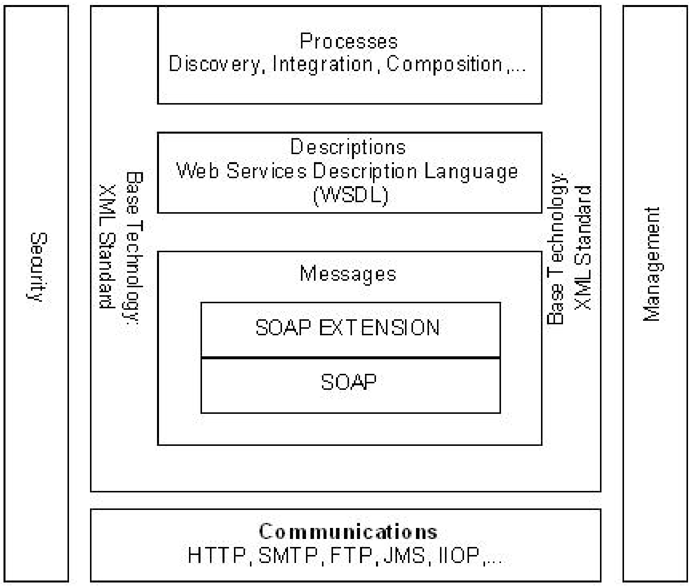
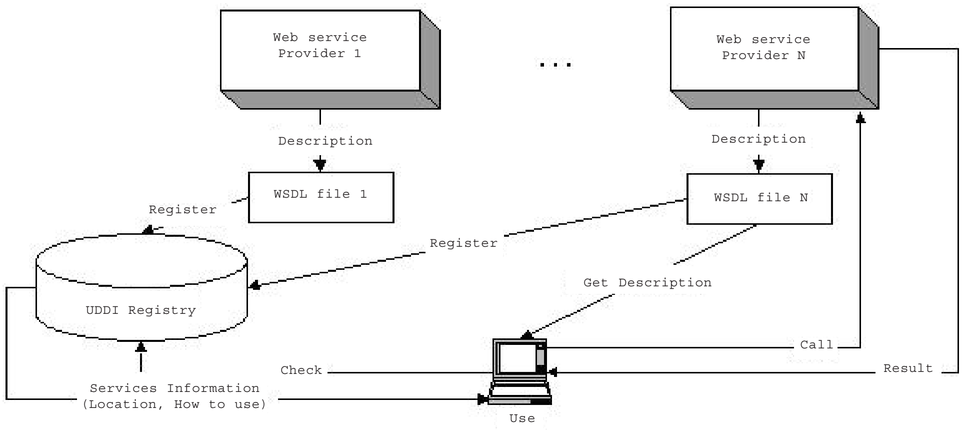

Security
Base Technology: XML Standard
Processes
Discovery, Integration, Composition,...
Descriptions
Web Services Description Language (WSDL)
Messages
SOAP EXTENSION
SOAP
Base Technology: XML Standard
Management
Communications
HTTP, SMTP, FTP, JMS, IIOP,...

Figure 2. How UDDI registry works in Web services processing



easily and systematically. At present, the most common Web services registry standard is UDDI. UDDI is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces that may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a Web services-based software environment for both publicly available services and services only exposed internally within an organization. In reality, "exposed" its services to its users, the Web service provider will try to register for three kinds of information mentioned above of its service in the UDDI registry; and the Web service users can find the proper services they need, and find out how they can call these services by searching the UDDI registry. Figure 2 shows how the UDDI registry works in processing Web services.

As shown in Figure 2, service providers register their Web services in UDDI registry first, then clients look up in the UDDI registry to search. After that, the UDDI registry returns proper service information, and finally, clients use this information to locate the WSDL file of service and call the service to get desired results.

As mentioned before, Web services have four major advantages: reducing human interaction, discovery, integration, and composition. It is easy to understand that Web services can reduce human interaction because this mechanism enables application-to-application connection where programs can recognize and call each other automatically. By using UDDI registry, clients can get proper services and get the information to call these services automatically. Since the

Web services use the open XML-based WSDL description and SOAP message, it is easy to integrate or compose programs together by constructing WSDL files and SOAP message pipes. These advantages can be used to alleviate the problems faced by bioinformatics users.

Previous Work

The Web services framework has a lot of potential to enhance bioinformatics in term of integration (composition, synchronization, and automation). However, there is lot of existing work on automation and integration of bioinformatics without using Web services.

Kemp, Angelopoulos, and Gray (2002) adopted a federated architecture with a five-level schema to support ad hoc queries across multiple data resources and correlate data retrieved from these. They built a mediator based on the functional data model database, P/FDM, which integrates access to heterogeneous distributed biological databases. And the five-level schema arises from the use of the ANSI-SPARC three-level schema to describe both the existing autonomous data resources and the mediator.

Stevens, Goble, Horrocks, and Bechhofer (2002) described a solution derived from the semantic Web, the ontology inference layer (OIL), as a solution for semantic bioinformatics resources. They also argued that another similar ontology language called DAML would merge with OIL under the name DAML+OIL.

We also found some research about agent application on bioinformatics, and there are some relationships and functional overlaps between agent applications and Web services application on bioinformatics.

Karasavvas, Baldock, and Burger (2002) gave a brief overview of a Multi-Agent Bioinformatics Integration System that aims at helping users make the most of the available integration facilities while also increasing their trust in the system. Bryson, Luc, Joy, and Jones (2001) introduced GeneWeaver, a multiagent system for bioinformatics, and described the agent interactions that allow the integration and management of analysis methods and data in detail. Moreau (2002) compared Web services and agents in the context of bioinformatics grid at transport layer. He described their abstract communication architecture based on an agent communication language (ACL), and then mapped it onto Web services' backbone—XML protocol. After comparing two frameworks, he concluded that first, agent may be seen as Web services, and second, XML protocol is expressive enough to support an agent communication language.

Although Web services framework has high potential for enhancing bioinformatics' performance by means of integration and automation, we found that there are not many published studies describing how Web services can be used to enhance bioinformatics application.

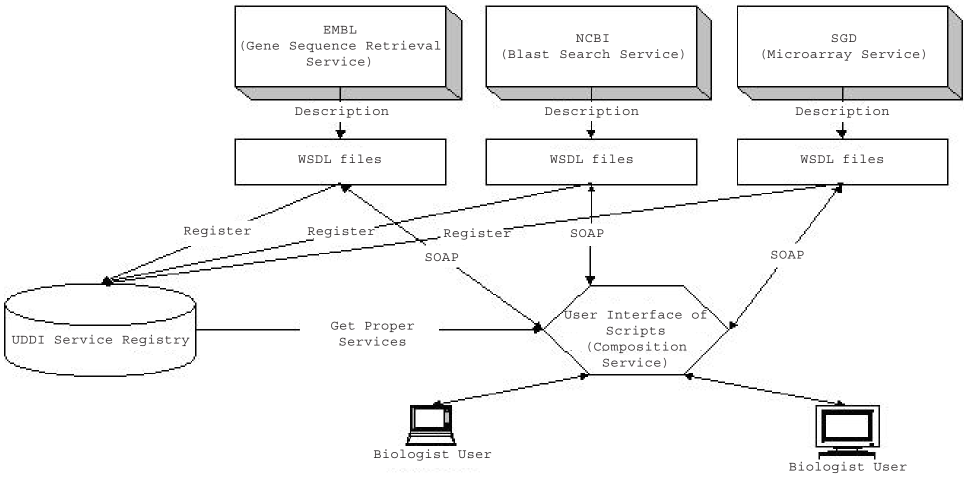
Stein (2002) proposed a model to use Web services framework to integrate different bioinformatics tools together. He argued that current bioinformatics databases are isolated islands. He also proposed that a Web services model could link disparate systems, and thereby create a more unified set of bioinformatics tools and databases. In his model, a third party can be used as a hub which can unify different bioinformatics' interfaces and a UDDI registry can act as the resources reservoir. However, he did not provide detailed instructions on how his Web services-based model can enhance the biology research processes.

Buttler, Coleman, Critchlow, Fileto, Han, Pu, Rocco, and Xiong (2002) identified challenges in the discovery, search, and access of multiple bioinformatics data sources, and tried to provide a solution by using the semantic Web in conjunction with database technologies and techniques. They examined several issues that biologists are facing: first, it is very difficult to use hundreds of available bioinformatics tools because the inputs must be in the correct format with correct semantics and the tools must be invoked in specific, standard ways; second, cutting-edge work can be made available directly form the research lab where it is created long before it ends up in the large bioinformatics databases; and third, because of involving human interaction, even at well-known sources such as NCBI or EMBL, the data are not necessarily available for in-depth analysis. They also described a scenario in which they integrated different bioinformatics tools into biologists' research process by semantic Web. They conclude that there is great potential for the semantic Web to alleviate the challenges they presented. Although in their scenario they involved semantic Web into a biology research process, there are several limitations. First, this process is static, and it means that the biologist must know in advance which tools to use for each step in the process. In other words, biologists should prearrange the research process manually. Second, in their model they used service class description to execute the source discovery process; however, it is only at a conceptual level, and they did not present how they describe and what standards they used to described the services. Last, they did not give a solution to organize and maintain those service description classes.

Methodology

Web services framework has potential to enhance the performance and efficiency of using bioinformatics tools by eliminating or reducing the three

Figure 3. Web services' application framework on bioinformatics



abovementioned drawbacks. As described before, Web services can help in composition, integration, and discovery. These advantages or functions can be used to reorganize bioinformatics tools, thereby reducing human interaction during the process and improving performance.

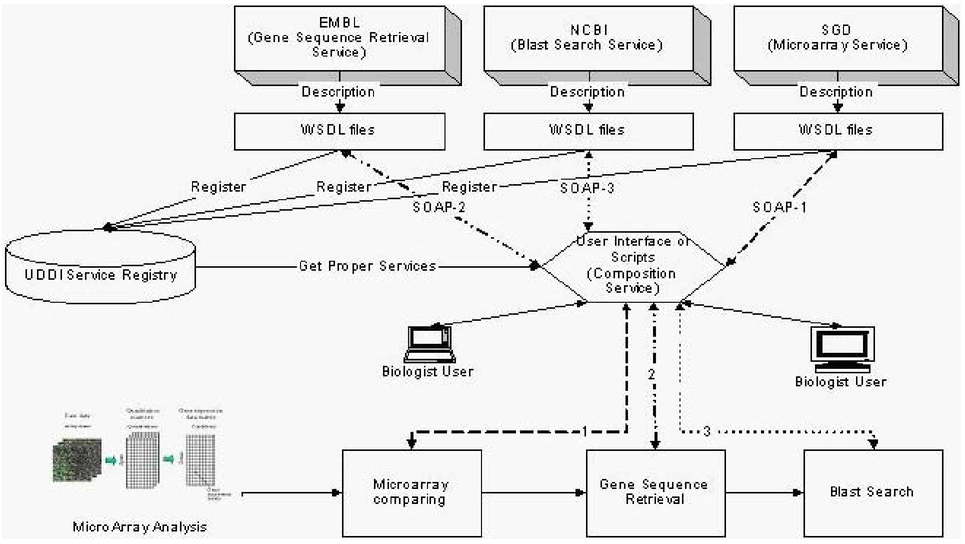
Figure 3 shows a possible framework to apply Web services using bioinformatics data providers. The central "User Interface or Scripts" can be viewed as a third-party Web site or service composition provider that users interact with. All bioinformatics services are described by WSDL files, which are maintained by a third party. All services are registered in the UDDI services registry. When a biologist requests a kind of service such as BLAST search or sequence retrieval, the central site goes to the UDDI registry to find proper bioinformatics providers' service(s), and then automatically calls them by using WSDL description and SOAP messages. It then shows organized result(s) to users. The central site can also combine several different types of services (such as microarray, sequence retrieval, Blast search, etc.) into one service. The central site also has a description in WSDL files, and works as a Web service. Thus, the software or programs in biologists' laboratories can call it automatically, that is, biologists do not need to operate on this central site manually, and they can just manipulate their own software or programs installed in their laboratories' computer to use service providers' bioinformatics tools.

Composing Individual Bioinformatics Services into Comprehensive Ones

One or several kinds of bioinformatics tools will be involved in the biological research process. During the research process, biologists may use the some microarray service(s) (such as SGD), sequence fetch service(s) (such as EMBL), BLAST search service(s) (such as NCBI), and so on. By using Web services framework, these services can be combined, and from the users' view they look like one service. Figure 4 shows how Web services are incorporated in biology research.

As shown in Figure 4, multiple services may be used in a biology research process. First, supposing a biologist is doing a microarray analysis, after dyeing and laser scanning, she/he wants to compare the result with previous ones. She/he calls the central site with her/his microarray submission, and the central site searches the UDDI to find that SGD is the proper service provider. The site then calls SGD to compare two microarrays and finally returns the results to the central site. Next, the central site returns to the user the gene that has the most similar expression profile. Further, if the biologist wants to retrieve the whole sequence of the expressed gene, the central site will help to select sequence retrieval provider (e.g., it selects EMBL), submit gene ID and return the sequence back from EMBL to the biologist. If she/he still wants to find homologs by gene matching, the central site can also help with that. It can help to obtain a bundle of homologs from NCBI by Blast search. Even if the biologist does not

Figure 4. The process of using Web services in bioinformatics



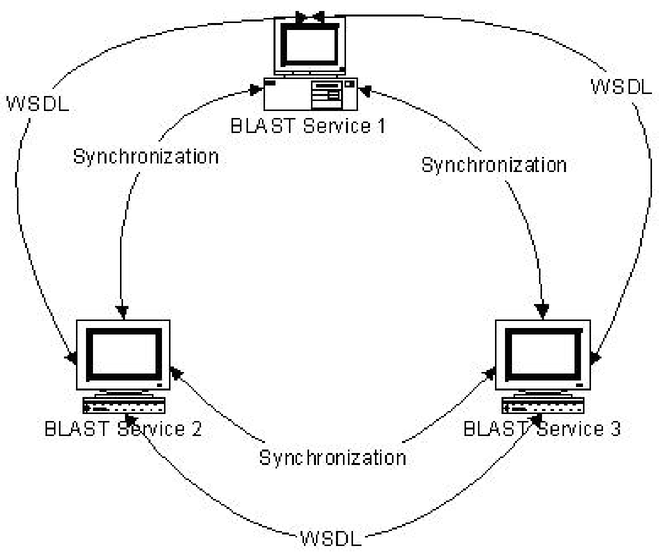
find the same gene, it is possible to find a new or promoter one, which can be verified through database calls and submit it to one or several databases. To submit the results, one needs to format the information into the database's data model. However, the central site can retrieve the selected database data model description (WSDL) files, format the results into required standards, and finally submit them.

It should be clear from the above that the search and discovery process using Web services is a "blackbox" to end users such as biologists. They need only perform biological research shown in the bottom layer in Figure 4 and submit their requests to the central site. In other words, biologists do not need to contact each bioinformatics tool service to submit requests to obtain desired results; instead, these services are integrated into their biology research automatically when needed.

Synchronizing Different Bioinformatics Tools on Same Function

As described in previous sections, there are many tools (databases or services) managed by different companies, institutes, and organizations delivering the same bioinformatics functionality (e.g., BLAST), and the data formats in these bioinformatics tools are different even though some of them can be mapped to each other. It is necessary to synchronize these different bioinformatics tools

Figure 5. The synchronization among three services on the same functionality



providing the same functionality, because biology researchers cannot review all the tools available to them; they either miss a useful service or are unable to compare the results from different sources if the tools are not synchronized. Therefore, it may cost more money and time for the researchers to do the repetitive work.

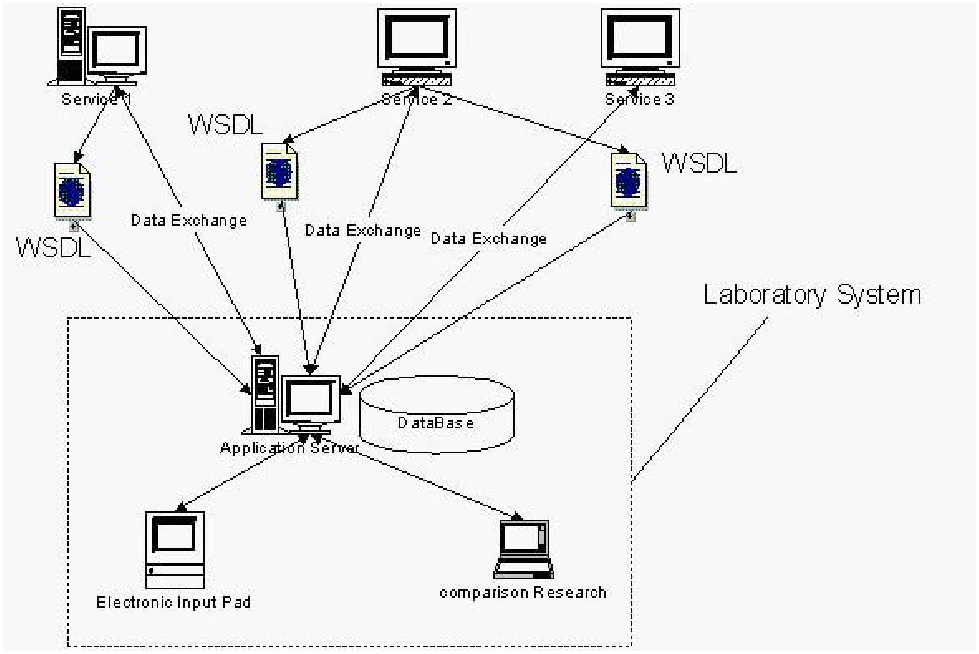
One can do the synchronization without the Web services framework. However, there will be one program for every two bioinformatics tools to map their data formats to each other. If we want to synchronize *n* tools, we need to write C_n^2 particular programs, and that is a very time- and money-consuming job.

Using Web services, one can use files to describe each tool's data format, and if it is necessary to synchronize data, the tools will try to read the WSDL files of other tools by first understanding the data format, and then mapping the data to its own data format. Thus, the data can be synchronized among multiple tools.

Automating Laboratory System by Integrating Services

As described in a previous section, in the Web services for bioinformatics framework, the central site itself can be made into a Web service. So based on the Web services framework, the central site can be integrated with the software

Figure 6. The integration between lab system and bioinformatics tools



system in the researchers' local lab. As mentioned before, we need to write WSDL files for each bioinformatics tool. A biology laboratory system can contact and read this file, and based on the instruction of the WSDL files, it can call the services (bioinformatics tools), thereby connecting the services and reducing human interaction.

The advantages of this integration are that (1) it allows biologists to use familiar software in their local lab; (2) biologists can process data from internal experiments as well as external sources together, in same format, thereby saving time and effort in data transformation and integration. It will be much easier to do deeper analysis when the data are in the same format and stored in the same place; (3) it can help the biology research process to become more coherent, eliminating unrelated tasks such as migrating external data to local system.

To implement this integration by Web services framework, a WSDL file should be made for the central site first, and by using the service description in the WSDL file, a stub should be created as part of the laboratory information system. Finally, the retrieved data must be stored in the local system.

It should be stated that integration is not limited only to the central site and the lab's local system. It also includes the composed bioinformatics service provider, and actually, in our model, the system is a bi-level Web services-based system. The central site uses Web services framework to integrate the bioinformatics into its service, and the lab also integrates the central site's service into its local system.

Registering and Searching Web Services in UDDI Registry

The UDDI registry acts as a bioinformatics service metadata server, and it can be public or private. The bioinformatics services register themselves in public UDDI registry, and biologists can find the WSDL file of proper service, then locate and utilize the service. Service users can also construct their own UDDI registry to save both external services and internal services.

Conclusion

This paper stated the relationship between peer-to-peer (P2P) computing and Web services, and summarized that application of Web services in bioinformatics is a specialized application of P2P. It then reviewed several problems in current bioinformatics, and based on the problem characteristics we presented a methodology using Web services framework and UDDI to enhance bioinformatics service performance on composition, integration, automation, and discovery. All those improved aspects are related to P2P computing, and we can say that P2P can make a considerable contribution to enhance the performance of bioinformatics. It should be noted that the Web services framework is not the only way to enhance bioinformatics services for solving the problems mentioned in this article. However, because of its open standardized protocols and compatibility with other systems, it is a good choice for data and service integration in bioinformatics. In summary, as a specialized application of P2P computing system, application of Web services in bioinformatics is feasible, effective, and efficient.

References

Achard, F., Vaysseix, G., & Barillot, E. (2001). XML, bioinformatics and data integration. *Bioinformatics, 17,* 115–125.

Baxevanis, A., & Quellette, F. (2001). Bioinformatics: A practical guide to the analysis of genes and proteins (2nd ed.). John Wiley & Sons, Inc.

Brazma, A., Parkinson, H., Schlitt, T., & Shojatalab, M. A quick introduction to elements of biology—cells, molecules, genes, functional genomics, microarrays. from http://www.ebi.ac.uk/microarray/biology_intro.html

Bryson, K., Luck, M., Joy, M., & Jones, D. (2001). Agent interaction for bioinformatics data management. *Applied Artificial Intelligence, 15*(10), 917–947.

Buttler, D., Coleman, M., Critchlow, T., Fileto R., Han, W., Pu, C., Rocco, D., & Xiong, L. (2002). Querying multiple bioinformatics information sources: Can semantic Web research help? *SIGMOD Record, 31*(4), 59–64.

Karasavvas, K., Baldock R., & Burger, A. (2002). A multi-agent bioinformatics integration system with adjustable autonomy: An overview. *AAMAS,* 302–303.

Kemp, G.J.L., Angelopoulos, N., & Gray, P.M.D. (2002). Architecture of a mediator for a bioinformatics database federation. *IEEE Transactions on Information Technology in Biomedicine, 6,* 116–122.

Stein, L. (2002). Creating a bioinformatics nation. Cold Spring Harbor Laboratories NATURE, *417*(9), 119–120.

Stevens, R., Goble, C., Horrocks, I., & Bechhofer, S. (2002). OILing the way to machine understandable bioinformatics resources. *IEEE Transactions on Information Technology in Biomedicine, 6,* 129–134.

Chapter XIII

Content Delivery Services in a Grid Environment

Irwin Boutboul, IBM Corporation, USA

Dikran S. Meliksetian, IBM Corporation, USA

Abstract

In this chapter, we propose a new approach for content delivery services by meshing together the best of grid computing and peer-to-peer (P2P) computing. The goal is to design a secure, reliable, and scalable system for efficient and fast delivery of content. The system consists of a combination of nondedicated servers and peers to provide the proposed service. We describe the challenges of designing such a system and discuss possible solutions and trade-offs. We detail the necessary interlacing of grid and P2P feature to achieve the goal. We present a prototype that is built based on the proposed approach.

Introduction

The concept of a computational grid was introduced in the mid 1990s to describe a distributed computing environment for science and engineering. Since then, a number of computational grids have been successfully developed and utilized to solve big science problems, and the concept of grid (Foster, 2002) has been extended to other domains. In this chapter, we present the use of a grid for distributed content delivery.

The increasing use of online rich-media content, such as audio and video, has created new stress points in the areas of content delivery. Similarly, the increasing size of software packages puts more stress on content delivery networks. New applications are emerging in such fields as bioinformatics and life sciences that have increasingly larger requirements for data.

In parallel to the increasing size of the data sets, the expectations of end users for shorter response times and better on-demand services are becoming more stringent. Moreover, content delivery requires strict security, integrity, and access control measures.

All those requirements create bottlenecks in content delivery networks and lead to the requirements for expensive delivery centers.

The technologies that have been developed to support data retrieval from networks are becoming obsolete. These technologies can be categorized into three groups. In the first category are file transfer mechanisms such as FTP, HTTP, and SMTP. The second category consists of distributed file systems such as DFS, NFS, and WINS. Finally, the third category includes streaming retrieval mechanism such as RTSP and SCTP. Each category provides specialized features for different applications.

In this chapter, we propose a new approach for content delivery services by meshing together the best of grid computing and peer-to-peer computing. The result is a secure, reliable, and scalable system for efficient and fast distribution and delivery of content.

This chapter concentrates on bulk file transfer mechanisms and is organized as follows. In the next section, we present some of the current solutions. We then present the concepts and requirements for using grids for content delivery followed by the description of a particular implementation of those concepts, the IBM Download Grid. Finally, we present our conclusions.

Current Solutions

One of the main objectives of the ideas presented in this chapter is to increase download speed. This can be accomplished by circumventing some of the inherent restrictions of the Transmission Control Protocol (TCP) protocol (Transmission Control Protocol, 1981). It is possible to increase the download speed by opening multiple TCP streams with the same source. The TCP/IP protocol will distribute the network bandwidth between the opened streams and thus achieve a better utilization of the network for the end user. Many software packages (Download Accelerator, 2004) use this method to increase download speeds. Unfortunately, this approach is not network friendly and will have adverse effects on other applications using the network. It is true that individual users will experience faster delivery, but if everyone were to use this method, no one will see any benefit. This is not a scalable solution. Moreover, it does not solve the server side bottleneck problem.

Peer-to-peer (P2P) technologies (O'Reilly P2P, 2004; Barkai, 2002) have emerged as a potential solution for content delivery problems. The P2P approach solves the single-server bottleneck problem by downloading the different pieces of the content from multiple locations. The set of locations is often determined dynamically based on speed and response time considerations. Thus, the download is potentially as fast as possible.

The P2P approach introduces a number of issues. First and foremost is that of reliability. The discovery of a particular content is not always guaranteed. Even though the content exists within the P2P network, it might not be accessible by a particular requester because the search algorithm does not extend to the nodes that have the content, or because the node that has the content denies the request for some reason. Even when the content is accessible, there is no guarantee on the completion of the download or the speed at which it will be completed.

The integrity of the downloaded content is never guaranteed before the completion of the download. The authenticity and origin of the downloaded content is always questionable.

Security is a major concern of P2P networks. Privacy is currently not supported by P2P networks. All content is public and one has to use an alternate distribution channel to manage encryption keys or access control.

In this chapter, we present an alternative to existing solutions which builds upon both grid and P2P technologies to solve the issues described above.

Grid and P2P for Content Delivery

Grid Capabilities

Grid computing is defined as flexible, secure, coordinated resource sharing among a dynamic collection of individuals, institutions, and resources. The sharing of these resources must be tightly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. Therefore, grid computing presents unique authentication (Foster & Kesselman, 1999), authorization (Pearlman, Welch, Foster, Kesselman, & Tuecke, 2002; Thompson, Johnston, Mudumbai, Hoo, Jackson, & Essiari, 1999), resource access, resource discovery (Clarke, Sandberg, Wiley, & Hong, 2000), and resource management challenges.

Security is paramount in the grid environment because the shared resources belong potentially to different organizations and access to those resources must be secured and controlled. This is particularly important when new code, services, and content are to be distributed over the network. The resources must be able to trust the request, without having a priori knowledge of the requester.

This is usually accomplished through a public key infrastructure (PKI) (Benantar, 2001) and X509 certificate (Internet X.509, 1999). This infrastructure is usually augmented by proxy certificates. A proxy certificate is a certificate that is derived from, and signed by, a normal X.509 certificate or by another proxy certificate for the purpose of providing restricted impersonation within a PKI-based authentication system. A proxy certificate allows the definition of proxy policies that specify and restrict the rights and privileges of the proxy certificate holder in addition to providing an authentication mechanism.

P2P Capabilities

Grid capabilities ensure many advantages over P2P, as described above. Still, peer resources are something to consider for content sharing (Ding, Nutanong, & Rajkumar Buyya, 2003). In order to take advantage of the grid capabilities, one has to install the grid framework on the target machines first. For obvious reasons, this cannot happen on each and every machine of the end-user community. The grid framework is often heavyweight and more adapted for servers than end-user desktops/laptops.

In the case of content delivery network, each downloader could potentially act as a server for other requesters, making available his/her local files.

It makes sense to take advantage of this potentially large pool of “free” resources that downloaders offer for content delivery.

Grid Meets P2P for Content Delivery

When designing a system where grid and P2P coexist (Foster & Iamnitchi, 2003), it is important to keep in mind that P2P networks are peer driven and cannot be trusted. The trust issue appears on many points:

- Availability: The peers can go up and down any time, without any control over it.
- Control: We cannot let peers share any file and we must preserve the QoS control that grid offers.
- Security must not be jeopardized by the addition of peers.

Below, we present a number of solutions to these problems.

Availability

Since peers can go down anytime, therefore we always need a grid network to back up the P2P network. It is true that the existence of P2P resources can reduce the load on the server grid resources. However, we cannot deduce from this that we can dramatically reduce the number of grid servers without affecting the performance. First, it is very difficult to estimate the number of peers who will be able to serve content at a given time. It is difficult to know the availability of peers, and even more difficult to predict the usage patterns for the downloaded files. It could happen that downloaded files are deleted right away after delivery.

Next, there is a direct side effect of the reliability issue for peers. Since the P2P network is not controlled, it can go down anytime as a whole. For example, it could be that a virus is infecting all peer machines, removing the ability of peers to serve files. Worse, a malicious code could corrupt the peer-to-peer delivery mechanisms to deteriorate the overall network availability.

Let us consider a content delivery network with only one grid server and thousands of peers. A new virus takes down the P2P network. The fix to the virus needs to be delivered, but the P2P delivery mechanism is down. Everyone will have to tap into the unique grid server in order to get the fix pack. This will probably result in a denial of service and temporary slowdown of enterprise activity.

Consequently, the decision to decrease the server grid infrastructure in favor of a client grid becomes a policy issue. It is a matter of assessing the reliability of the delivery mechanisms.

Ideally, if resources are not an issue, a server grid network to be able to provide all necessary delivery capabilities is preferred, and the P2P network (or client grid) would be a bonus.

With today's on-demand world and grid capabilities, this ideal scenario can be reached. The server grid network would be a pool of grid resources that can be used on demand. When the client grid is sufficient, cost is minimized. When the client grid does not provide necessary resources, the server grid is used at higher expense. In both cases, the quality of services is provided at minimum cost.

Control

Shared Content

In the usual P2P paradigm, it is up to the end users to decide what to share. In a trusted content delivery network, this is not acceptable. Thus, each file downloaded from a peer must be approved by grid means.

We can use grid security mechanisms to do this. In order to access content, a peer must present a valid proxy to another peer. The proxy must embed the file attributes so that only the file described in the proxy can be downloaded from the peer. The trusted authority does the deliverance of the proxy.

To find a file on the grid network (Balakrishnan, Kaaschoek, Karger, Morris, & Stoica, 2003; Czajkowski, Fitzgerald, Foster, & Kesselman, 2001), a requester can ask the grid information systems to locate the file. The grid information systems are semi-dynamic databases. Usually these are LDAP or SQL databases. For a peer-to-peer network, due to the highly dynamic nature of the peers, it is not scalable to store information of the peers' files and statuses in a database.

Many choices are offered. First is to use Gnutella flat topology (Gnutella Protocol Specification, 2000; Ding, Nutanong, & Rajkumar Buyya, 2003); this is clearly not acceptable when speed of data retrieval is crucial. Second choice is to use P2P supernodes infrastructure, like the fasttrack P2P network (Sharma, 2002; Ding, 2003), to publish this information, and then getting hold of it requires accessing those supernodes. Another solution is to have a centralized server, like Napster (Napster Protocol, 2004; Ding, Nutanong, & Rajkumar Buyya, 2003), which contains information about peers' files. Since our content delivery network must provide only a specific set of files to the end user, it is necessary that both the P2P information systems and the grid information systems match at some point. A file that is no longer on the grid network must not be available any longer on the P2P network. In order to keep both networks synchronized, it is much easier to opt for the Napster-like central repository. It also provides better control and better response time in terms of content update.

Upon startup, each peer publishes its list of files to the central repository. At any time, the overall status of the network is available. This gives greater control in terms of autonomic grid administration. Since we know which files are available on the peer network, the server grid agents can dynamically adjust the distribution of the files on the server grid network to continuously provide a certain quality of service in the overall content delivery network. As more peers serve the file, grid agents can delete copies of the file on the grid network, saving space and money.

Quality of Service

One of the primary concerns is the speed at which the content will be delivered. In a grid environment, we control the grid servers and can know through information systems which ones are available and dispatch the load accordingly.

From a peer perspective, it is up to the peer to decide which bandwidth is allocated to download/upload for other peers. We cannot force peers to deliver a certain QoS. Consequently, in order to maintain a specified QoS in our download grid, the client has to automatically balance the QoS between the grid network and the peer network. If the peer/client grid network cannot provide enough speed, then the server grid network is used as backup. This business case makes sense in an on-demand world. The cost of the server grid network is incurred only when needed. The peer network is virtually free. By tapping resources in the peer network, we reduce the overall cost. Still, we do not decrease our quality of service because the server grid network will always provide the necessary bandwidth, at an additional cost to the content owner.

In this design, we lower the content owner's cost as much as possible while keeping a certain quality of service.

Security

The content delivery network security has to remain unchanged with peers. This involves both integrity and privacy of the files.

In order to keep the integrity of the files, before a peer serves a file to another peer, it has to check for the file integrity. This check is performed at the time files are published on the client grid information repository.

A double check is still needed by the requester upon file reception to ensure end-to-end integrity.

One major issue with peers is that we cannot rely on them for access control and authentication. The grid security framework does not apply here. Hence, we need an alternate solution. In order to keep control of the P2P network status, we need a way to control who is downloading what and when. To achieve this, we can use the grid proxy certificates with peers. Before downloading content from a peer, one has to show valid credentials to the peer. The credentials are issued by the trusted root authority. A peer will not serve content without an official proxy with appropriate credentials. With this mechanism in place, we know exactly who is downloading what.

We still have two additional pieces we need to know: When is the download performed, and where is it performed? The time constraint gives more flexibility in terms of provisioning and usage. The location information gives greater flexibility in terms of load balancing. Both the time and information controls need to be secure.

To achieve the time constraint, we can use the lifetime of the proxy certificates. Each X509 proxy certificate has an expiration date. A peer will deny access to its files once this expiration date is over. At any time, we know exactly the potential maximum usage of the P2P network. In case this usage is too high, we can increase the server grid network capabilities accordingly.

To achieve the location constraint, we can embed basic identity information in the proxy. For example, we can indicate an IP address in the proxy. A peer accepting a proxy will check that the proxy's policy contains its IP address before granting access.

Unfortunately, IPs are not reliable for a number of reasons such as NAT and so forth. However, we can easily solve the problem by using some unique IDs that each peer would create and publish at the time of startup/registration with the P2P network information services.

With the mechanisms described above, we can achieve a high level of control on the P2P/client grid network.

For the server grid network, we can trust the machines and the underlying security framework. Only authorized parties can access the files. However, for the P2P network, we cannot trust the peers, owners of the machines.

Since we cannot trust the peers, we need to have sensitive content encrypted not only over the wire but also in storage. Thus, any sensitive content on the P2P network has to be encrypted. Peers can get a copy of the encrypted content without jeopardizing the confidentiality of the content.

Only authorized people will be able to decrypt content. Unauthorized people will only be able to download content. Again, we can use the proxy certificates to implement this policy. When requesting a file, one has to first authenticate, if necessary. If access is granted, the issued proxy certificate will contain the file's encryption key to perform the decryption.

This key is specific to each file. In order to make the system secure, this key must either be sent over a secure connection or be itself encrypted by using public/private key pair mechanisms. The latter solution is preferred since the proxy is to be presented to every peer before getting access. It avoids having to handle secure connections between peers.

The IBM Download Grid

The IBM Download Grid (IDG) is an IBM internal prototype that is based on the concept introduced in the previous section. It is being used currently to distribute content to IBM employees.

Compared with existing P2P technology, the IDG builds upon the security framework and the distributed nature of a grid. IDG achieves a high download rate without losing control of network security and without adversely impacting the performance of other services provided by the grid resources.

IDG is a scalable, reliable, and secure content delivery network running on the IBM internal grid, which provides an efficient and adaptable download service. The features of IDG are as follows:

- Parallel downloads from multiple servers. The download client downloads portions of the same file from multiple servers concurrently. As the download of some chunks is completed, the client dynamically reassigns portions of remaining chunks to the download servers that have completed the download.
- Load balancing. The client downloads from a set of servers selected by the management center algorithms, which abides a pluggable policy. The policy aims at balancing the load among the servers.
- Quality of service. For each server, an associated bandwidth can be forced for the download, inside the download proxy. Thus, each client will have access to a defined quality of service.

- Security and privacy with centralized access control. Private files are encrypted and access control is done at the management center level. Access control policies can be changed directly at the management center level without having to access any IDG servers.

These features produce the following benefits:

- Faster downloads. The client experiences download rates that are on the average three to five times faster than regular downloads.
- Non-dedicated resources. IDG does not require dedicated resources. It uses shared resources that are available on the IBM internal grid.
- Better network load distribution. The download server selection algorithms shape the network traffic according to configurable policies.
- Reduced administration. IDG is an autonomic system. The grid is self-healing and self-optimizing.

In the following paragraphs, we present the details of the IBM Download Grid and explain how we achieve these benefits.

Architecture

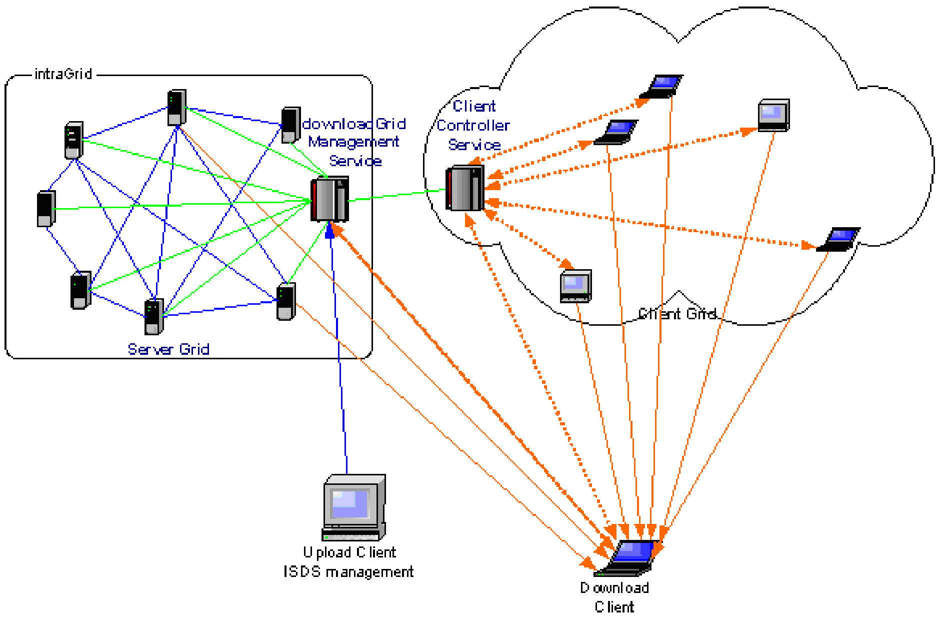
The IBM Download Grid system made of two complementary delivery mechanisms is shown in Figure 1. The server grid and the client grid form the two coupled components that interoperate to provide the download service. The basic operations of the download grid are provided by specialized download servers and specialized clients.

The download servers are implemented as high-performance non-blocking I/O servers. They enforce authentication and access control based on X509 proxy certificates. They can throttle the bandwidth of each session based on the proxy certificate's attributes.

There are two specialized clients. One is implemented as a Java applet and uses only the server grid; the second is implemented as a stand-alone Java application and uses both the server grid and the client grid.

Although there is no technical issue in using the client grid with the lightweight java applet, this mode is restricted to the application for the following two reasons:

Figure 1. The IBM Download Grid


intraGrid
downloadGrid Management Service
Server Grid
Client Controller Service
Client Grid
Upload Client ISDS management
Download Client

- The applet is a one-time function triggered from a browser and generally expects immediate action. Use of the client grid requires additional processing which would delay this immediate action.
- Due to the transient aspect of the applet, the applet cannot serve files on the client grid. Having a client grid consumer that is not a provider would result in unbalanced usage between the applet and the application.

The Download Grid client controller and the Download Grid management center cooperate to provide the download service. The Download Grid management center keeps track of all content available on the grid and the distribution of this content on the server grid. The client grid controller knows the overall available content on the client grid. It communicates with the management center through Web services. Depending on the available content on the client grid, the management center can automatically re-adjust the content distribution on the server grid to always provide a certain quality of service. The management center knows at any given time what the client grid can potentially deliver and can balance download requests between the server grid and the client grid.

We will describe the different components.

Server Grid

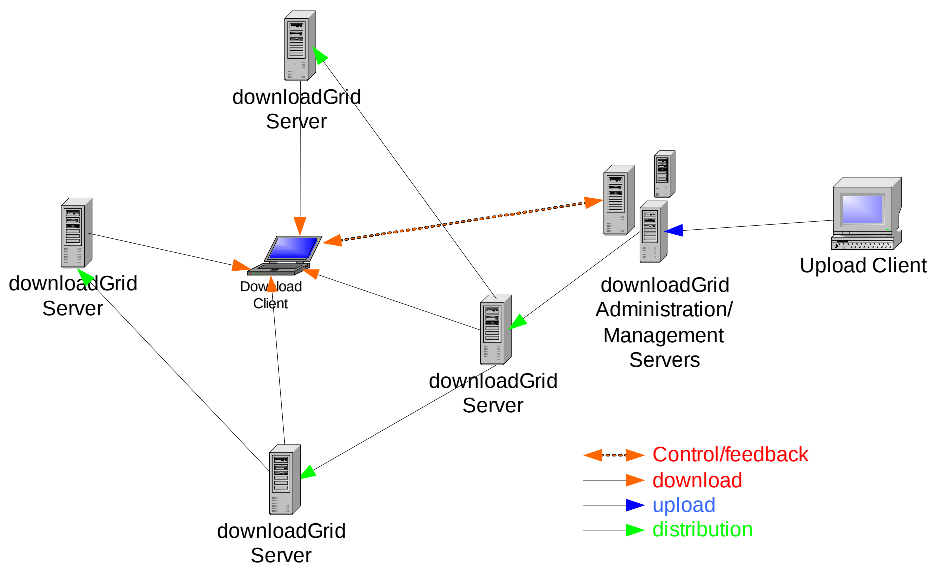
The server grid component of the IDG architecture is shown in Figure 2.

It consists of an Administration/Management service and a set of download servers distributed over the IBM internal grid.

The typical flow of actions using the server grid is as follows:

1. A content owner publishes content on the download grid by uploading a file using the upload client.
2. The management center distribution agents distribute the file to a specific set of grid servers, depending on servers load, publisher's requests and additional policies.
3. A client requests a file for download. The management center creates an optimized download plan for the client. The download plan lists a set of download servers that the client can use.
4. The client contacts the specified servers and downloads portions of the file using an algorithm described later.

Figure 2. The Server grid component of IDG



The management service provides the mechanisms for determining a download plan, to upload and distribute content to the servers, and to monitor the overall operation of the system. All the different components talk to each other through Web services, at the management center level.

From the content owner point of view, the download grid appears as a black box. The only exposed services are the management center Web services. The administration of the download servers is hidden.

The administration of files is similar to a regular FTP server from the publisher's point of view. The power and complexity of the download grid is hidden.

Client Grid

The client grid is managed by the download grid client controller. Upon startup, a download grid client registers with the download grid client controller. It publishes the list of files that are available. The client grid controller knows the list of allowed download grid files (through the management center) and intersects both lists.

At any time, the download grid client controller knows the map of available files on the client grid. When the download grid client unregisters, the map gets updated accordingly.

Time-out mechanisms ensure a necessary update of the map for disconnected clients.

Including the client grid, the download process described earlier is modified to the following:

1. The client requests a file from the management center. It gets an optimized download plan embedded in a proxy certificate.
2. Depending on the download plan attributes, the client will either request the same file from the client grid controller or start the download from the server grid.
    a. When the optimized plan allows the use of the client grid, the current client requests the file from the client grid controller. It gets a secondary peer download plan for the client grid and starts the download from the peers. If the speed does not reach a certain threshold, it automatically adds servers from the server grid according to the primary plan for speeding up the download process. The threshold is specified in the primary download plan. The threshold can force a download from the client grid only.

   b. Otherwise, the client starts the download from the grid servers according to primary plan.

3. As the download of some chunks is completed, the client reassigns the download of portions of remaining chunks to the download servers that have completed the download.
4. At the end of the download, the client checks the file integrity and reports to the management center about the download process and the download rates achieved.

IBM Download Grid Features

Security

The security mechanisms include access control and privacy. Access to content is performed in two steps. First, the user has to authenticate against the management center to get a proxy certificate. Then, the user needs to authenticate against the download grid servers or clients to be able to download a file. The second authentication is done through an SSL handshake which ensures that the user is authorized to use the presented proxy.

Since files are distributed on both the server and client grids, it is necessary to provide additional privacy mechanism. Files on the download grid are encrypted using AES 128-bit encryption as they are published on the download grid. The files are locally encrypted with a randomly generated key, and uploaded encrypted. The encryption key is securely sent over HTTPS to the management center.

When a user wants to download an encrypted file, once access is granted, the user is given the encryption key to decrypt the file. The following steps are necessary:

1. Authenticate against the management center.
2. The management center grants or denies access based on the file access control list/policy.
3. If access is granted, the file's encryption key is encrypted in the proxy certificate so that only the requester can decrypt it (with public/private key pair mechanisms).
4. The requester downloads the file encrypted from the grid servers.
5. At the end of the download, the requester decrypts the file locally.

Software Updates

Software updates are an important piece of the download grid. In a highly distributed environment, it is critical to have an update mechanism that can quickly update software on all nodes in no time and without disrupting the overall service.

Update concerns both the download grid servers and the download grid client application.

The download grid server code update is done through the grid framework. The server is remotely and securely updated using grid security infrastructure (GSI) (Generic Security Service Application Program Interface, 1997; Grid Security Infrastructure, 2004) and job submission. The update sequence consists of:

- stopping the server remotely
- transferring the new code
- starting the server remotely

The download grid client code update needs another mechanism, since the grid framework is not available on the clients. For this purpose, we use an existing update mechanism through Java Web Start. Since the download grid code is 100% Java, it makes sense to use the deployment and update mechanisms of Java Web Start. Each time a new update is necessary, Java Web Start will only send the difference in jars between the old and new version. This results in great savings in terms of update deployment. Moreover, at each client start, Java Web Start automatically checks for updates, the process being transparent to the end user.

Autonomic System

One concern of the download grid is the administration of the grid servers. The download grid is distributed worldwide on potentially thousands of servers, and we cannot afford duplicating administration costs. In order to keep the administrative costs low, we use autonomic mechanisms.

Self-Healing

The most basic issue is when a grid server goes down for some reason. Monitoring continuously the grid servers could result in a scalability issue.

Moreover, it would require an intelligent and distributed system of sensors. Indeed, monitoring the servers from a unique location would not be reliable because of the heterogeneous nature of the network. The link could be broken between sensor A and grid server B, but not between grid server B and the rest of the world.

In order to solve these issues, we use the download grid nodes (both clients and servers) as sensors. In the event of a connection failure, a downloader, which can be either a grid server during file distribution or a client, immediately alerts the management center. On the management center, the monitoring agents double-check the identified misbehaving grid server. If it appears that the grid server is down, the monitoring agent will try to restart the server automatically and securely through the grid framework. If it cannot restart it, it will flag it as "not active" in the database, so that further download requests will not use it. As soon as the grid server can be restarted, it is flagged as "active" in the database and joins back the pool of active grid delivery servers.

The system will automatically heal itself and readjust the distribution of the files to satisfy requirements. For example, if one of the servers goes down and it appears that the file is distributed on an insufficient number of grid servers to deliver enough bandwidth to end users, the distribution agents will automatically replicate the file on other grid servers.

Self-Optimizing

This automatic distribution can also be coupled with usage patterns. Since the management center receives feedback on all downloads, it knows the actual speed end users are getting when downloading a file. If this speed is below a certain threshold, it can automatically increase the distribution of the file on other grid servers, wherever and whenever necessary.

Implementation and Future Directions

All components of the download grid are written in Java. The management center uses a backend database to store information. The main components talks with each other through Web services. Some components that require very fast response such as distribution agents, use the database directly.

Though the download grid presented above presents an exciting number of improvements compared to the old-fashioned client/server HTTP/FTP model, it still relies on an old protocol: TCP/IP. TCP/IP was engineered decades ago and was not optimized for fast transfer of large amounts of data. Academic effort (PhatPackets research project, 2004) is developing new protocols based on a mix of UDP/TCP to reliably move large amounts of data more efficiently across the Internet. Future directions for content delivery network would be to investigate new protocols in order to take advantage of the new technologies, not only at the software and architecture level, but also at the transport level.

Conclusion

Creating a secure, reliable and scalable content delivery network presents many challenges. Grid and P2P communities provide key mechanisms for delivering content. We use the grid mechanisms to reach a high level of security. The scalability is greatly improved by using peer-to-peer capabilities. Finally, by meshing together the grid and peer-to-peer networks, we reach a high level of reliability and flexibility.

Thus, we can take advantage of both grid and peer-to-peer worlds to create an innovative solution for delivering content to end users.

References

Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., & Stoica, I. (2003). Looking up data in P2P systems. *Communications of the ACM, 46*(2).

Barkai, D. (2001). An introduction to peer-to-peer computing. From http://www.intel.com/update/departments/initech/it02012.pdf

Benantar, M. (2001). The Internet public key infrastructure. *IBM Systems Journal, 40*(3).

Clarke, I., Sandberg, O., Wiley, B., & Hong, T.W. *Freenet (2000): A distributed anonymous information storage and retrieval system*. International Workshop on Designing Privacy Enhancing Technologies, Berkeley, CA.

Czajkowski, K., Fitzgerald, S., Foster, I., & Kesselman, C. (2001). Grid information services for distributed resource sharing. *10th IEEE International Symposium on High Performance Distributed Computing, 2001*,181–184.

Ding, C. H., Nutanong, S., & Rajkumar Buyya, R. (2003). Peer-to-peer networks for content sharing. from http://www.gridbus.org/~raj/papers/P2PbasedContentSharing.pdf

Download Accelerator (2004). SpeedBit. From http://www.speedbit.com

Foster, I. (2002). The grid: A new infrastructure for 21st century science. *Physics Today, 55*(2), 42–47.

Foster, I., & Iamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. From http://iptps03.cs.berkeley.edu/final-papers/death_taxes.pdf

Foster, I., & Kesselman, C. (Eds.) (1999). *The grid: Blueprint for a new computing infrastructure.* Morgan Kaufmann.

Generic Security Service Application Program Interface, Version 2 (1997). IETF, RFC 2078, from http://www.ietf.org/rfc/rfc2078.txt

The Gnutella Protocol Specification v0.4 (2000) Document Revision 1.2. From http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf

Grid Security Infrastructure (2004). Globus Alliance. From http://www.globus.org/Security/

Internet X.509 Public Key Infrastructure Certificate and CRL (1999). IETF, RFC 2459, Profile. From http://www.ietf.org/rfc/rfc2459.txt

Napster protocol (2004). From http://opennap.sourceforge.net/napster.txt

O'Reilly P2P (2004). From http://www.openp2p.com

Pearlman, L., Welch, V., Foster, I., Kesselman, C., & Tuecke, S. (2002). *A community authorization service for group collaboration.* IEEE 3rd International Workshop on Policies for Distributed Systems and Networks.

PhatPackets research project (2004). From http://www.phatpackets.com

Sharma, A. (2002, September). The FastTrack network. *PC Quest Magazine, India*. From http://www.pcquest.com/content/p2p/102091205.asp

Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., & Essiari, A. (1999). Certificate-based access Control for widely distributed resources. 8th Usenix Security Symposium.

Transmission control protocol (1981). IETF, RFC 793. From http://www.ietf.org/rfc/rfc793.txt

About the Editors

Ramesh Subramanian is the Gabriel Ferrucci professor of computer information Systems at the School of Business, Quinnipiac University in Hamden, Connecticut. Dr. Subramanian received his PhD in Computer Information Systems and MBA from Rutgers University, New Jersey. He also holds a Postgraduate Diploma in Management from XLRI–Center for Management Studies, Jamshedpur, India, and a Bachelor of Applied Sciences from Madras University, India. Dr. Subramanian's research interests include information systems strategy, information systems security, digital asset management, e-commerce, XML, XSL, Web content management, peer-to-peer networking and resource sharing, and IT education; he has published and presented several papers in these areas. Prior to his current appointment, Dr. Subramanian has held the following positions: Senior Software Engineer at IBM's Advanced Technology Lab (Southbury, CT); Associate Professor of MIS (tenured) at the College of Business and Public Policy–University of Alaska, Anchorage; Instructor of Computer Science at the Computer Science Department, Rutgers University, New Jersey; Member of the Technical Staff–Database Research District at Bell Communications Research, Morristown, New Jersey.

Brian D. Goodman is currently a certified IT architect for IBM's w3 On Demand Workplace, based out of New York, New York. His current assignment in the CIO's Office is to work closely with the Lotus organization to develop and communicate a client strategy and architecture for the w3 On Demand Workplace. Other recent areas of focus include expertise location, knowledge management, collaboration, and common profiling. Prior to his current position, Brian worked as a Software Engineer in WebAhead, one of IBM's advanced Internet technology labs, in Southbury, Connecticut, developing proof points with emerging technologies. Other professional experience includes commercial CBT development, multimedia production, graphic design, and print. His current interests include emerging data architectures, caching strategies, service-oriented architectures, message queuing, pub/sub, Web services, social networking systems, and peer-to-peer systems. Brian received a multidisciplinary BA degree in computer science, psychology, and graphic design from Hampshire College, Amherst, Massachusetts. His interest centered on early childhood human computer interface design.

About the Authors

Sridhar Asvathanarayanan is a data warehouse administrator in a multinational reinsurance company and is a graduate with a major in Information Systems from Quinnipiac University, Connecticut (USA). He has more than 15 years of experience in information technology, in areas such as development, training, and administering database systems such as Oracle, Sybase, and Microsoft SQL Server. He has considerable experience in securing database systems for warehouses, e-commerce applications and operational systems for large organizations. Over the years, he has had the opportunity to work with different operating systems and networks. He enjoys reading articles on database and network security and working on SQL query analysis and optimization.

Irwin Boutboul, born in France, holds the French *Diplôme d'Ingenieur* degree from ENSIMAG School. Distributed systems, networks, and security have always interested him. He has been programming since he was 10. He began his career with IBM at the Watson Research Center (USA) where he helped in the development of the IBM internal grid. He is an early adopter of grid technologies. He recently joined the IBM WebAhead technology think tank where he develops new applications based on emerging technologies.

Michael Bursell graduated from Cambridge University (1994) and spent two years with Cambridge University Press engaged in electronic publishing and Internet development. He joined Citrix Systems where he worked on mobile code research projects, developing an interest in security, particularly for mobile entities. Work with Convergys in telecommunications allowed him to combine this interest with peer-to-peer systems, designing and implementing several authentication-based protocols. His social sciences background led him to investigate more deeply the impact of social structures on P2P systems, and since joining Cryptomathic (UK) in 2004, he has continued research into the interaction between trust and security.

Rajkumar Buyya is director of the Grid Computing and Distributed Systems (GRIDS) Laboratory in the Department of Computer Science and Software Engineering at the University of Melbourne, Australia. He was awarded the Dharma Ratnakara Memorial Trust Gold Medal in 1992 for his academic excellence during his studies at Mysore University. He received leadership and service excellence awards from the IEEE/ACM International Conference on High Performance Computing in 2000 and 2003. He has received over one million dollars in national and international competitive research grants. Dr. Buyya is one of the creators of system software for PARAM supercomputers, developed by the Center for Development of Advanced Computing (C-DAC), India. He has pioneered economic paradigm for service-oriented grid computing and demonstrated its utility through his contribution to conceptualization, design, and development of grid technologies such as Nimrod-G, GridSim, and Gridbus to power-emerging e-science and e-business applications.

Giorgos (Georgios) Cheliotis has a PhD in Electrical Engineering and Computer Science from the National Technical University of Athens. He was a doctoral and later post-doctoral fellow at the IBM Zurich Research Laboratory, his research focusing primarily on the fields of telecommunications and IT economics. He studied the international telecommunications bandwidth market extensively and became a recognized expert in the field of bandwidth trading. He also codesigned and implemented part of a first-of-a-kind sales tool, which estimates the financial benefits of adopting grid technology in a commercial organization. He joined McKinsey & Company's Business Technology Office in Zurich, Switzerland in September 2003.

Li-Te Cheng is a research scientist at IBM Research in the Collaborative User Experience Group in Cambridge, Massachusetts (USA). He is currently working on enabling collaborative capabilities for small teams of developers within their integrated development environments, and also has interests in shared workspaces, human computer interaction, mobile computing, and augmented reality. He holds a PhD in Electrical Engineering from Memorial University of Newfoundland, and a BASc and MASc in Systems Design Engineering at the University of Waterloo.

Choon Hoong Ding is a research assistant in the Grid Computing and Distributed Systems (GRIDS) Laboratory at The University of Melbourne, Australia. He completed a Bachelor of Information Systems from Multimedia University, Malaysia, and the Master of Software Systems Engineering from the University of Melbourne.

Stacey L. Dogan is associate professor of Law at Northeastern University School of Law in Boston (USA). She received her JD from Harvard University and her BS in Economics at Massachusetts Institute of Technology. Before joining the Northeastern faculty in 1998, Stacey practiced law in Washington, DC and California, and served as a law clerk on the United States Court of Appeals for the District of Columbia Circuit. Professor Dogan writes and teaches in the areas of intellectual property and antitrust law, with a particular focus on the unique challenges raised by computer technology and the Internet.

Kai Fischbach is a research associate at the University of Cologne (Department for Information Systems and Information Management), Germany. His current research interests include the economics of ubiquitous and peer-to-peer computing and the impacts and development of Web services. He is a coeditor of an edited volume on peer-to-peer (P2P) (published by Springer in 2002) and coauthor of an article about P2P prospects published in the *Communications of the ACM* (2/2003). He is also the co-track chair of the “Peer-to-Peer Paradigm” mini-track at HICSS 2004.

Aryya Gangopadhyay, PhD, is an associate professor in the Department of Information Systems, University of Maryland Baltimore County (USA), and he is also the director of graduate programs in the same department. Dr. Gangopadhyay is interested a wide range of topics including applied database application, electronic commerce, geographical information systems (GIS), and bioinformatics. Dr. Gangopadhyay obtained his PhD in Computer Information Systems at Rutgers University.

Werner Geyer is a research scientist at the IBM T.J. Watson Research Lab in Cambridge, Massachusetts (USA), in the Collaborative User Experience Group (CUE). His current research interests include ad hoc collaboration, virtual meetings, and messaging. His research focuses on the intersections of egocentric versus public, informal versus formal, unstructured versus structured types of collaboration. Before joining CUE, Werner was a Post Doc at IBM Research in New York where he worked on new Web-based team support technologies and on capture and access of distributed meetings. He holds a PhD in Computer Science from the University of Mannheim, Germany. He also earned an MS in Information Technology, which combines Computer Science and Business Administration, from the University of Mannheim.

Ross Lee Graham is a founder and the series coordinator for the IEEE International Conference on Peer-to-Peer Computing (annual since P2P2001) and coedits the proceedings. He has participated in research projects on peer-to-peer accountability, peering smart homes, peer-to-peer methods for data distribution, and is now working on security issues in dynamic overlay infrastructures. He has contributed expertise to Ericsson on their AmIgo Project. He is a senior member of the information technology group at Mid-Sweden University and is currently negotiating the founding of a journal that specializes in overlay networks. He has published *Time-Module Intensive Reading Program*, a speed-reading course that focuses on building speed in comprehension.

Michal Jacovi is a research staff member at the IBM Haifa Research Lab, Israel, and belongs to the collaboration technologies. Her research interests include human computer interaction, information visualization, and knowledge management. Michal received her MSc in Computer Science from the Technion, Haifa, Israel, in 1993. She joined IBM in 1993, and has worked on several projects involving visualization and awareness on the Web. Over the years, she moved from site mapping to awareness, to collaboration, e-Learning, and knowledge management. She has published a number of papers in international conferences and journals.

Chris Kenyon has been a research staff member at IBM Research, Zurich Research Laboratory (Switzerland) since 2000 and was previously at Schlumberger Austin Research and McGill University. He has an MCSE in Operations Research from the University of Texas at Austin (1997), a PhD in Applied Mathematics from Cambridge University (1989), and is a former fellow of Wolfson College, Cambridge (Computer Modeling). His current research interests include price design for outsourcing contracts, grid economics, mathematical finance, and real options. He has more than 30 publications in journals, conference proceedings, book chapters, and patents. He is a member of INFORMS and IEEE. He may be reached at chk@zurich.ibm.com.

Xin Li is currently pursuing his PhD in the Department of Information Systems, University of Maryland Baltimore County (UMBC) (USA), and his dissertation topic is on the application of object-relational database for microarray (gene expression) data. Mr. Li is also interested in supply-chain management (SCM), Web services, and business process management (BPM). Mr. Li obtained his master degree in Information Systems at UMBC in 2003, and before he attended UMBC, he obtained both his master's degree in Management Science and his bachelor's degree in Management Information Systems (MIS) in China.

Yoelle S. Maarek is the senior manager of the Search and Collaboration Department at the IBM Research Lab in Haifa, Israel. Her research interests include information retrieval, Web applications, and collaborative technologies. She graduated from the ENPC in Paris, and received her DEA (graduate degree) in Computer Science from Paris VI University, both in 1985. She received her PhD in Computer Science from the Technion, Israel, in 1989. Before joining the Haifa Lab, Dr. Maarek was a research staff member at the IBM T.J. Watson Research Center. She currently serves on the PCs of several international conferences and has published more than 40 papers. She is a member of the IBM Academy of Technology and an IBM Master Inventor.

Dikran S. Meliksetian is an IBM senior technical staff member leading the design and development of the IBM internal grid based on industry standards. He is a member of the WebAhead team of the IBM Internet Technology Division and is engaged in a number of leading technology projects. He has designed and implemented advanced content management systems. Previously, he has served on the faculty of the EECS Department at Syracuse University and the ECE Department at the South Dakota School of Mines and Technology.

Michael J. Muller works as a research scientist/design researcher at IBM Research in the Collaborative User Experience Group. His current work is to support the community of designers at Lotus Software and IBM. Previous work in the company includes developing architectures for communities of practice and designing for accessibility. Before coming to IBM, he worked in research and practice in usability, user-centered design, and work analysis at Microsoft, U.S. West Advanced Technologies, Bellcore, and IBM. He holds a PhD in Cognitive Psychology from Rutgers University. In addition, he finds his informal experience in conflict resolution a useful tool for helping the groups he works with reach consensus.

Sarana Nutanong earned a Bachelor of Engineering (Computer) (Hons), and Master in Software System Engineering from The University of Melbourne, Australia. He is currently working as a research assistant in the peer-to-peer networks and applications research team. His active research areas include parallel algorithms, distributed spatial database systems, and load balancing on P2P systems.

Manish Parashar is an associate professor in the Department of Electrical and Computer Engineering at Rutgers University (USA), where he is also the director of the Applied Software Systems Laboratory. His research interests include parallel and distributed computing, scientific computing, and software engineering. Manish received a BE in Electronics and Telecommunications from Bombay University, India, and MS and PhD degrees in Computer Engineering from Syracuse University.

John Patrick is president of Attitude LLC and former vice president of Internet technology at IBM, where he worked for 35 years. During his IBM career, John helped start IBM's leasing business at IBM Credit Corporation and was senior marketing executive for the launch of the IBM ThinkPad brand. Starting in the early 1990s, John dedicated his time to fostering Internet technologies. One of the leading Internet visionaries, John is quoted frequently in the global media and speaks at dozens of conferences around the world. *Business 2.0* named him as one of the industry's most intriguing minds, *Industry Week* named him one of the top-30 people who drive innovation and provide the initial spark to economic growth, and *Network World* called him one of the 25 most powerful people in networking. John was a founding member of the World Wide Web Consortium at MIT in 1994, a founding member and now the chairman of the Global Internet Project, a senior member of the Institute of Electrical and Electronics Engineers, and a member of the Internet Society, the Association for Computing Machinery, and the Working Group on Authentication at the Center for Strategic and International Studies. He is a member of the board of directors of Opera Software ASA, Jupitermedia Corporation, Knovel Corporation, and Danbury Health Systems, Inc., and has been an adviser to IntraLinks, Neoteny, Space.com, and ThirdAge Media. He is also president of The Ridgefield Symphony Orchestra Foundation. His book, *Net Attitude*, was released in November 2001.

Cristina Schmidt is a PhD student in the Department of Electrical and Computer Engineering at Rutgers University (USA). Her research interests include distributed systems and networking, in particular peer-to-peer systems and self-organizing overlays. She received BS and MS degrees in Computer Science from Babe-Bolyai University, Romania.

Christian Schmitt is a research associate at the University of Cologne (Department for Information Systems and Information Management), Germany. His current research interests include the enhancement of knowledge management using new technology such as ubiquitous and peer-to-peer computing.

Detlef Schoder is a professor at the University of Cologne, Germany, where he is presiding over the Department for Information Systems and Information Management. He is the author of more than 140 reviewed publications including journal articles in leading international and German outlets, for example, *Communications of the ACM, International Journal of Technology Management, Information Economics and Policy, Journal of Electronic Commerce Research, Zeitschrift für Betriebswirtschaft* (*ZfB*), and *Wirtschaftsinformatik*. Professor Schoder is on various editorial boards including *Information Systems and eBusiness Management, International Journal of Electronic Business, International Journal of Internet*, and *Enterprise Management*. He is a coeditor of an edited volume on peer-to-peer (published by Springer in 2002), guest editor of a special issue about P2P for *Wirtschaftsinformatik*, and coauthor of an article about P2P prospects which is published in the *Communications of the ACM* (2/2003). He is also the co-track chair of the "Peer-to-Peer Paradigm" mini-track at HICSS 2003 and 2004.

Vladimir Soroka is the manager of the Collaboration Technologies Group at the IBM Research Lab in Haifa, Israel. He holds a BSc in Computer Science from The Technion, Israeli Institute of Technology, and is currently finishing his Master studies in Communication at Haifa University. He is also a coauthor of several papers published in international journals and presented at international conferences. Prior to joining IBM, Mr. Soroka has been working on different Internet-based technologies, including Internet-based fax system. His interests include computer-supported cooperative work, virtual communities, social network analysis, and knowledge management.

Dinesh C. Verma manages the Policy & Networking Group at IBM T.J. Watson Research Center, New York (USA). He received his doctorate from the University of California, Berkeley (1992), and has worked at Philips Research and IBM Research laboratories. He has worked in the areas of managing performance and quality of service in computer networks, and development of content distribution networks. His current interests include policy-based network management, and various applications of peer-to-peer communication systems.

Juergen Vogel is a research associate at the University of Mannheim, Germany. In summer 2002, he was an intern at IBM Research in the collaborative user experience group in Cambridge, Massachusetts. He is interested in different aspects of peer-to-peer applications including consistency control, support for late-joining peers, transport protocols, and application-level multicast. At the University of Mannheim, he teaches courses on computer networks. He is currently working on his PhD in Computer Science and holds an MS in Information Technology from the University of Mannheim.

Index