Chapter 1

1. INTRODUCTION

1.1. Scope

Consider the title of this dissertation: "Resource sharing in distributed peer-to-peer Internet applications". The title can be divided into the following concepts: Distributed systems, Internet applications, peer-to-peer and resource sharing. These concepts indicate the scope and interest of our research.

1.1.1. Distributed systems

Distributed systems refer to applications which are divided into multiple functional components, executing on multiple hosts. The application is therefore distributed across multiple hosts. The hosts depend on each other in order to provide a complete system.

1.1.2. Internet applications

The various distributed components of the application needs to communicate to each other through a communications network. This study is limited to distributed applications that utilize the Internet as communication medium. The Internet is the largest, and in many regards the ultimate, communications network that allows communication between peers across the globe.

1.1.3. Peer-to-peer

Peer-to-peer (P2P), for the purpose and scope of this dissertation, refers to an application model for distributed systems.

The conventional model is a client/server model where the distributed system is divided into client and server components. In such a model a large number of client components communicate with a small number of server components. The server provides all of the application services to the client.

There is no distinction between client and server roles in a peer-to-peer model. Each host performs the tasks of both client and server components. The hosts that participate in a P2P system has equal capabilities and are therefore referred to as peers. Peers communicate directly with each other and can provide all the required application services among themselves, without dependence on a central server.

1.1.4. Resource sharing

Many types of peer-to-peer applications exist and they can be categorized according to our classification system (discussed in chapter 4). Resource sharing refers to one of the functional categories of peer-to-peer applications. P2P resource sharing applications refers to applications that allow peers to share resources such as storage space, CPU cycles and content. Resource sharing, and more specifically content sharing, is discussed in much detail as an example of peer-to-peer application.

1.2. P2P and the TCP/IP reference model

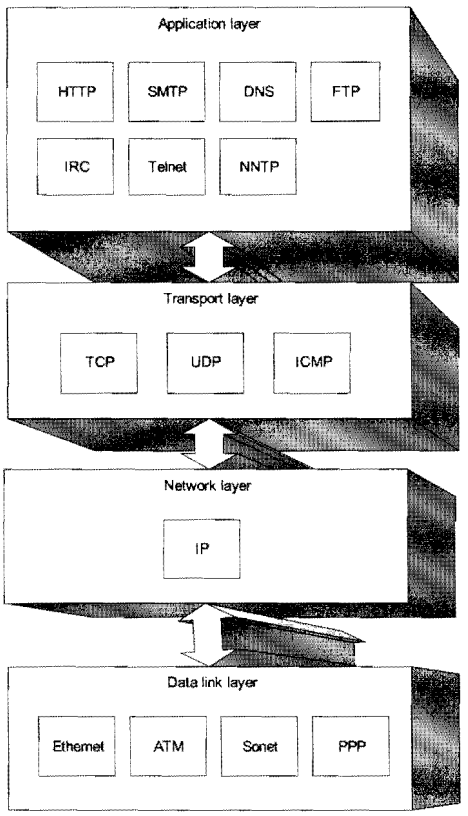


Figure 1: The TCP/IP reference model

Networked computing applications can be discussed on various levels or layers in terms of the well-known OSI and TCP/IP reference models, which are presented in detail by [Tanenbaum 1994]. The TCP/IP reference model [figure 1] is more relevant to distributed Internet applications.

The network layer (OSI model) or internet layer (TCP/IP model) facilitates peer-to-peer communications between hosts by ensuring that hosts can reach each other via the routing process. Internet traffic is broken up into packets. Each packet contains a source and destination address. Neither the transport layer, nor the network layer imposes any restrictions on the source or destination addresses of the packets. All nodes are treated as equal peers. And

there is no distinction between client and server nodes. The Internet is therefore inherently capable of peer-to-peer communications.

The peer-to-peer model, which is referred to in this dissertation, is however not the peer-to-peer communications which is inherent to the low-level network layers. The study is rather concerned with the models utilized in the high-level application layer.

1.3. Peer-to-peer purity

It is difficult to confine the exact scope of peer-to-peer applications. The purest form of P2P applications is a completely decentralized system. However, many centralized systems also expose some properties related to P2P applications. It is therefore helpful to refer to P2P applications as being *pure* or *less pure*. Pure P2P applications are also referred to as *true* P2P applications.

1.4. Resource sharing as an example of peer-to-peer applications

One possible (and currently popular) application of peer-to-peer networking is resource sharing. In chapter 6, 7 and 8 the author will present an in-depth study of three such systems: *Napster*, *Gnutella* and *Freenet*.

- Napster is a very controversial peer-to-peer application, which made many people realize the potential of peer-to-peer applications and sparked off a lot of debate, discussion and research.

- Gnutella is a very popular, open and widely used peer-to-peer protocol specification with many implementations.

- Freenet is a more complex but very promising peer-to-peer system with very unique properties such as anonymity and privacy.

The resources shared by the Napster, Gnutella and Freenet networks are computer files containing music, documents, images etc. These systems are therefore also referred to as content sharing or distributed file sharing applications.

1.6. Outline

This dissertation is divided into the following chapters:

- Chapter 1: Introduction – Defines the scope of our research and provides an introduction to peer-to-peer systems.
- Chapter 2: History – Discusses the history of Internet applications, as it led to the development of peer-to-peer applications.
- Chapter 3: Client/server vs. Peer-to-peer – Presents a comparison between the conventional client/server application model and the peer-to-peer model.
- Chapter 4: Classification – Presents a classification system for peer-to-peer systems. This allows applications with similar attributes and behavior to be grouped in order to study and refer to similar applications collectively.
- Chapter 5: Mechanisms – Presents five mechanisms which we have identified that can be used to describe the inner workings of peer-to-peer applications.
- Chapter 6: Napster – Presents an in-depth study of the controversial Napster peer-to-peer system.
- Chapter 7: Gnutella – Presents an in-depth study of the open Gnutella peer-to-peer system.

- Chapter 8: Freenet – Presents an in-depth study of the promising Freenet peer-to-peer system.

- Chapter 9: Comparison – Presents a comparison between the Napster, Gnutella and Freenet peer-to-peer systems. The chapter discusses differences and similarities in terms of important attributes other than the five identified mechanisms.

- Chapter 10: Other peer-to-peer initiatives – This chapter briefly mentions other peer-to-peer initiatives which the author is aware of.

- Chapter 11: Challenges – This chapter identifies remaining challenges and unresolved issues in peer-to-peer networks.

- Chapter 12: Implementing P2P on constraint mobile devices – Discusses various aspects unique to mobile P2P applications. Presents an implementation of a P2P phonebook application for constraint mobile devices.

- Chapter 13: Conclusion – This chapter concludes what the author has observed and learned through the study.

Chapter 2

2. A BRIEF HISTORY OF INTERNET APPLICATIONS

2.1. The importance of historical study

It is essential to study the origins of Internet applications in order to better understand the current state-of-the-art. It will allow us to identify future trends and to predict possible developments. We can only understand where Internet applications are headed if we know where they have been.

Modern Internet applications are influenced and constrained by their historical counterparts, due to the following factors:

- A common practice in software engineering is to design new software in such a way that it will be backwards compatible, supporting previous (historical) versions.
- New software is rarely designed from scratch. Most software is designed to leverage existing technology.
- Software frequently evolves by developing new layers of abstraction on top of existing components.

We therefore conclude that new software is frequently constrained by the functionality and mechanisms imposed by legacy systems. This is especially evident in Internet applications, which relies on the underlying protocols and infrastructure.

2.2. Evolution of the Internet

2.2.1. Past to present

The Internet started with the development of the ARPANET, whose original goal was to create a U.S. military command and control network that would be able to survive a nuclear war (this is considered to be a myth by some sources [InetDef]). The first four nodes of the ARPANET were connected in 1969. Institutions with ARPA (Advanced Research Projects Agency) research contracts were subsequently connected to the ARPANET. Other networks were also developed later.

The first ARPANET protocols (IMP-IMP and NCP) lacked the ability to connect different networks to each other. TCP/IP was specifically designed to handle communication between different interconnected networks. More and more networks started to use TCP/IP, including the ARPANET.

In 1984 the NSF (U.S. National Science Foundation) began developing a backbone network that would allow researchers from all universities to connect to it, regardless of DoD contracts. The NSFNET was the first network that used TCP/IP right from the start. It also had connections to the ARPANET and other networks. The NSFNET were operational in 1986.

The TCP/IP protocol allowed many networks all over the world to be connected to each other, eventually creating a global interconnected network, today known as the Internet.

2.2.2. The future

Current research efforts are underway to develop the next generation Internet, known as the Internet2 or NGI (Next Generation Internet). The Internet2 already connects networks in various countries (although not yet in South Africa).

One of the key benefits of the Internet2 is that it will provide a significant increase in bandwidth by utilizing fiber optic cables and optical routers. The Internet2 backbone operates at gigabit speeds (currently 2.5Gbps).

The current Internet uses the IP version 4 protocol (IPv4). IPv6, also known as IPng (IP next generation), became the official protocol of the Internet2 in 2000. The most notable enhancements of IPv6 are the following:

- Provides a much larger IP address space
- Provides Quality of Service (QoS) guarantees.
- Simpler automatic configuration of address.
- Expanded routing and addressing facilities.
- Improved efficiency.
- Improved security features.

2.3. Early Internet applications

Along with the development of hardware and infrastructure came the development of software applications to leverage these new technologies. In this section we discuss the origins of the most important Internet applications.

2.3.1. E-mail

Email can be seen as the lowest common denominator service on the Internet. Almost all persons with some form of Internet access will at least have an email facility. Email can be used on any Internet-enabled device, whether it is a full desktop computer or a portable wireless palmtop or cell phone.

In the early 1960s, before the ARPANET, computer scientists have devised ways of exchanging electronic messages within a time-sharing system. Ray Tomlinson invented an email program to send messages across the ARPANET in 1972. The mail program consisted of two parts: SNDMSG to send a message and READMAIL to read messages. The original intent of the program was only to handle mail locally on a time-sharing machine. Tomlinson later realized that he could use it in combination with his experimental file transfer protocol (called CPYNET) to carry a mail message from one machine and drop it into a file on another. In August 1972, Abhay Bhushan described in [RFC 385] how email could ride piggyback on the ARPANET file transfer protocol.

Tomlinson also became known for the @ sign used in email addresses. He needed a character that would not be found in any user's name, to separate the user name from the computer that the user was on. His selection of the @ sign later led to great debate. There was disagreement over what should go on the left hand side of the sign and what should go on the right. Tomlinson used a Tenex computer system. He did not realize that on the Multics computer system the @ character was used to send a "line kill" command.

An ARPA study showed that by the end of 1973 email messages accounted for 75% of the ARPANET traffic. In those days it was easy to send email, but reading and responding was difficult. When you read mail, all the messages you've ever received showed up and they were not separated. You could not respond to a message, you had to start from scratch, and the text editing tools were primitive. Larry Roberts subsequently wrote the first email management program RD (stands for READ), which was able to list, selectively read, file, forward and respond to messages.

Many variations and improvements to the first email programs followed. Different email programs started using different header formats. Messages sent from one mail program could not be handled by another, because the

headers could not be parsed. These compatibility issues led to some standardization efforts. A new list of standard headers was published in April 1975 in [RFC 680].

John Vittal developed MSG in 1975. MSG was by far the most popular all-inclusive email program for the ARPANET, providing replying, forwarding and filing capabilities. A new revised email specification [RFC 733] was published in 1977. The MSG program was incompatible with the [RFC 733] headers, even though Vittal had helped to write the RFC. The headers and protocols used by the popular MSG program eventually became the standard, not those specified in the RFC.

One of the first ARPANET mailing lists, MsgGroup, was created in 1975. Messages received from postings were manually re-mailed to everyone on the list. The process was later automated.

In 1980 there were about 400,000 electronic mailboxes and by 1990 the figure had risen to more than 12 million.

2.3.2. Telecommunication network (Telnet)

Telnet allows a user to control another computer remotely. The user is provided with a console, which reflects a console on a remote computer. Keystrokes are sent to the remote computer where the commands are executed.

Telnet was the first application demonstrated on the four node IMP (Interface Message Processor) ARPA network installed by the end of 1969 [Khare 1998b]. The first RFC related to Telnet was [RFC 97], "First cut at a proposed Telnet Protocol", and was published in 1971. The protocol was developed over the next 12 years and described in various RFCs. The Telnet protocol finally became the 8th official Internet standard in 1983. The Telnet standard suite is described in [RFC 854 to RFC 861]. Various other Telnet-

related RFCs (such as [RFC 1080] and [RFC 2066]) have been published after 1983, introducing extensions and clarifying existing issues.

The original spectrum of computers connected to the ARPANET varied significantly in aspects such as character sets, display sizes, line lengths, time-sharing mechanisms etc. Developing a protocol that would allow these disparate systems to interoperate was no easy task. The Telnet specification therefore land marked the design of the first sophisticated application-layer protocol.

In 1994 Telnet was the second most popular Internet application, based on Internet traffic.

Telnet is still used today, especially in Unix environments. Microsoft recently added a Telnet server to their Windows 2000 operating system. Telnet is great for remotely executing command-line utilities, but not GUI based applications. This led to the development of various remote desktop and remote control applications, such as X Server on Unix systems and Terminal Services on Windows systems.

2.3.3. File Transfer Protocol (FTP)

The File Transfer Protocol allows a user to copy files over the Internet. Copying files over the Internet might seem a trivial task when you consider that the bits of a file need to be transferred over the network from one host to another. That is in fact what the Trivial File Transfer Protocol (TFTP, Internet standard 33 discussed in [RFC 1350]) is all about. The complete FTP protocol however includes other functionalities such as authentication and file-system operations across various platforms [Khare 1998a].

When FTP first emerged it was the de facto transfer protocol. FTP was used to transfer printer spool files, email messages and public documents. Even though FTP connections were synchronous one-to-one connections, other

applications, such as the first email applications [RFC 385, RFC 751], were developed on top of FTP to simulate asynchronous many-to-many connections.

To date there is 44 RFCs published related to FTP. The first FTP RFCs were published in 1971 [RFC 163, RFC 172, and RFC 265]. The File Transfer Protocol was eventually approved as Internet standard 9 in 1985 [RFC 959].

Today FTP is losing ground to HTTP file transfers. The following aspects might indicate some of the reasons:

- Web browsers have become the preferred user interface for the Internet. Some web browsers support FTP downloads, but not FTP uploads.

- FTP does not maintain as much file meta-data as HTTP does. Information such as file creation date, original location and application data type is not available through FTP.

- FTP addressing structure is an opaque pathname at a host. HTTP uses URLs which has an internal path structure. This allows for easier mirroring and redirection mechanisms.

- FTP embarks a session-oriented state-full approach, which requires connection setup and management, while HTTP follows a much easier stateless request-response approach.

- The author also observed that in general file transfers are faster using HTTP than using FTP for the same file. However, we do not have any scientific theory or proof for this observation.

2.3.4. Usenet news

Usenet provides a convenient method for people with similar interests to exchange messages. Different subjects are grouped into hierarchies of related topics. The name of each group is supposed to give some indication of the topic of the newsgroup. Users can post messages to newsgroups or read existing messages from newsgroups which they are interested in.

The Unix User Network, or Usenet, was established in 1979. The first implementations of Usenet utilized the popular Unix-to-Unix Copy Protocol (UUCP). In 1986 the Network News Transfer Protocol (NNTP) was designed to enhance Usenet news performance over the Internet. A combination of interconnected UUCP and NNTP hosts was used to distribute news messages until the UUCP backbone eventually died in 1988 [Usenet].

Usenet started with just two newsgroup hierarchies; mod.* and net.*. The mod.* hierarchy contained groups that were moderated, while the net.* hierarchy contained all other groups. The fa.* (For ARPANET) hierarchy as well as other hierarchies with limited local distribution were also added in the early days. A complete archive of early newsgroup postings (1981) is available from "The Usenet OldNews Archive" [OldNews].

The Usenet backbone was created to dedicate resources towards the timely distribution of Usenet news. The "Backbone Cabal" was a group of backbone administrators who controlled the newsgroups. Since the Cabal owned the backbone, they decided on issues such as which groups get created and which groups get distributed.

A process known as "The Great Renaming" happened between July 1986 and March 1987. Seven main hierarchies were created (comp.*, misc.*, news.*, rec.*, soc.*, talk.*). All existing newsgroups were renamed to fit the new structure, according to the taste of the Cabal. This led to much unhappiness

and a subsequent flame war. The reasons for the renaming were the following:

- The increased number of groups made it difficult to organize the current structure.

- Controversial groups would all be added to talk.*, which would make it easy for administrators to drop them from distribution lists.

The Backbone Cabal refused to create groups such as rec.sex and rec.drugs on the backbone machines. This has led to the creation of the alt.* (alternative) hierarchy. These groups were distributed via alternative routes, which avoided the Usenet backbone and therefore could not be controlled by the Cabal. Brian Reid finally created the *alt.sex*, *alt.drugs* and *alt.rock-n-roll* group on April 3, 1988 [Hardy 1993].

Usenet news is still used today. Other news services have also been developed, most of which utilize a web interface.

2.3.5. Internet Relay Chat (IRC)

Internet Relay Chat allows users to talk to each other and have a conversation over the Internet. Multiple users can join the same chat room. Users can send messages, which are then instantly displayed to all other users.

The first computer-to-computer chat took place in 1972 at UCLA. The first IRC client and server were developed by Jarkko Oikarinen at the University of Oulu, Finland in 1988. The first IRC server was called "tolsun.oula.fi". Jarkko asked some of his friends at other universities to start hosting IRC servers when the number of users started increasing. By the middle of 1989 there were about 40 IRC servers worldwide.

The history of IRC is marked by various disagreements on how development of IRC should evolve, leading to the splitting of the IRC network (on various occasions) into many different networks, such as EFNet, ANet, Undernet, Dalnet, oz.org and IRCnet.

In 1991, during the gulf war, more than 300 simultaneous users were experienced for the first time in a channel called "live reports". The first IRC RFC was published in 1993 [RFC 1459]. Other enhancements to the protocol were later published, such as [RFC 2810, RFC 2811, RFC 2812 and RFC 2813].

Modern incarnations of IRC are still used extensively today. More advanced capabilities have been added. Numerous IRC networks exist, each developing its own customized version of the IRC protocol.

2.3.6. Gopher

Gopher is an information retrieval service, which can deliver text, graphics, audio and other content to a client. Gopher content is linked together in a hierarchical fashion. The hypertext links are kept in a tree structure, separate from the content.

Paul Lindner and Mark McCawhill from the University of Minnesota released Gopher in 1991, the same year in which the World Wide Web was released. Gopher was the first Internet application to provide point-and-click navigation and McCawhill therefore called it "the first Internet application my mom can use" [NetTimeline]. The software was freely distributed on the Internet.

Both Gopher (TCP port 70) and the WWW (TCP port 80) were designed to provide an easier way to use the Internet than the Unix shell experience. Original implementations did nothing FTP did not already offer. In 1994 there were 4,800 Gopher sites, 1,200 anonymous FTP sites and only 600 web

sites [RFC 1689]. Gopher was therefore more popular in the early days than the World Wide Web. Gopher maintained a master list of publicly accessible servers, which was indexed regularly by Veronica (Very Easy Rodent-Oriented Net-wide Index to Computerized Archives), much like the Archie FTP index.

HTTP headers are considered to be the main reason why the WWW became the killer application, which killed Gopher [Khare 1999b]. Adding new HTTP headers could easily extend the HTTP protocol functionality. State was hacked back into this (stateless) protocol by using mechanisms such as cookies.

Gopher sites still exist today. Few people are aware of the fact that even modern browsers such as Internet Explorer and Netscape can be used to access these Gopher sites (by typing gopher://gopher.browser.org in the browser). There is even a movement, called the "Bring Back Gopher Campaign" [GopherManifest], which is attempting to bring Gopher back to life.

2.3.7. World Wide Web

The World Wide Web (WWW) is currently the most popular Internet application [Hobbes]. The WWW provides a global network of hypertext documents linked to each other. The Hyper Text Transfer Protocol (HTTP) specifies the rules for accessing and distributing these documents. The content and structure of the documents are described with the Hyper Text Markup Language (HTML).

Tim Berners-Lee and other scientists at CERN designed the World Wide Web in 1989. His original goal was to provide a system that would make it easier to retrieve research documents. He developed a browser program a year later and called it the World Wide Web. The program was subsequently

released in 1991 for free on an FTP site. He had no idea of the major success that his invention would have.

The first web browsers were very primitive. Only plain black on white text documents were supported and images were published separately. By the end of 1992 there were only 50 web sites in the world. The first significantly improved commercial browser was Mosaic X, launched in 1993. The browser was developed by Mark Anderson, who later founded Netscape. Mosaic was made available for free to the educational community. The browser became an instant hit and the number of web sites exploded. The ability to combine words, pictures, and sounds on Web pages excited many computer users who realized the potential for publishing information on the Internet. In 1995 the WWW surpassed FTP as the most popular Internet service.

The World Wide Web is still evolving today and is used for much more than its original intent. Modern web sites are marked by their dynamic content, as opposed to the static nature of web sites a decade ago. HTTP is used today for much more than what it was originally designed for and might just become the "Grand Unified Protocol" that is envisaged [Khare 1998a].

2.4. Internet growth

Internet growth figures are presented here to demonstrate the popularity of the Internet as well as its future potential. The Internet started growing at a phenomenal rate in the 1990s. The growth pattern of the Internet seems to be exponential, with no sign of a slowdown yet.

There is currently a trend to connect not only computers, but also other types of devices (cell phones, PDAs, automobiles, toasters etc.) to the Internet. This might cause yet another host growth explosion.

Figure 2 presents the number of hosts present on the Internet between the year 1970 and 2001. In 2001 there were more then 110 million hosts on the

Internet. The World Wide Web (WWW) is currently the most popular application on the Internet. Figure 3 presents the number of websites on the Internet between July 1993 and July 2001. In 2001 there was more then 30 million website present on the Internet. The graph of website growth seems to be logarithmic with evidence of slowdown. The values presented in the graphs were obtained from [Hobbes].

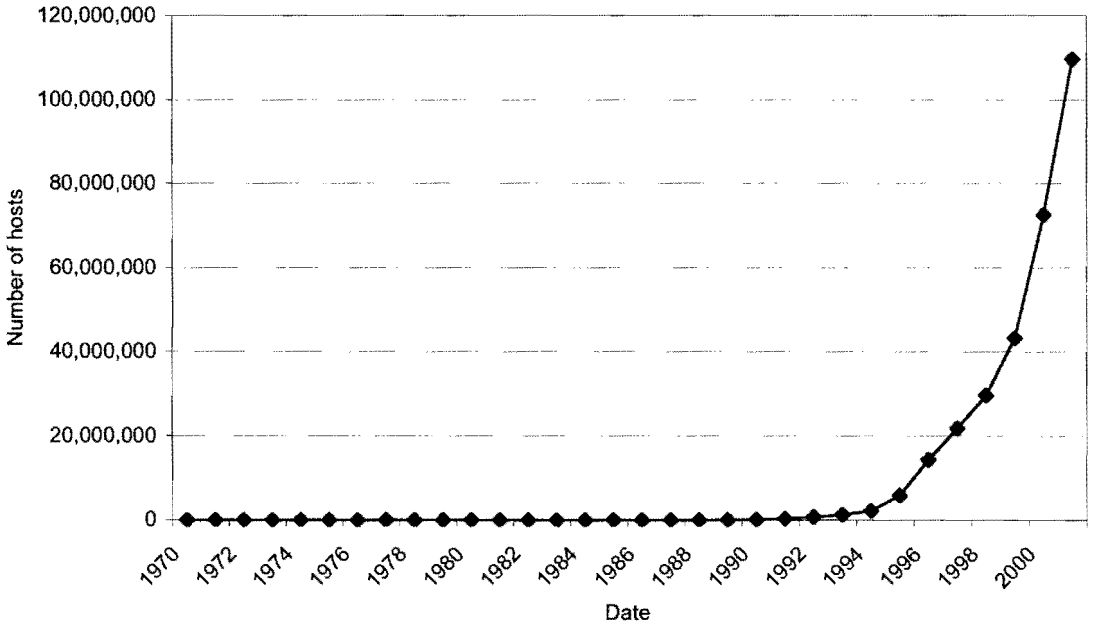


Figure 2: Internet growth

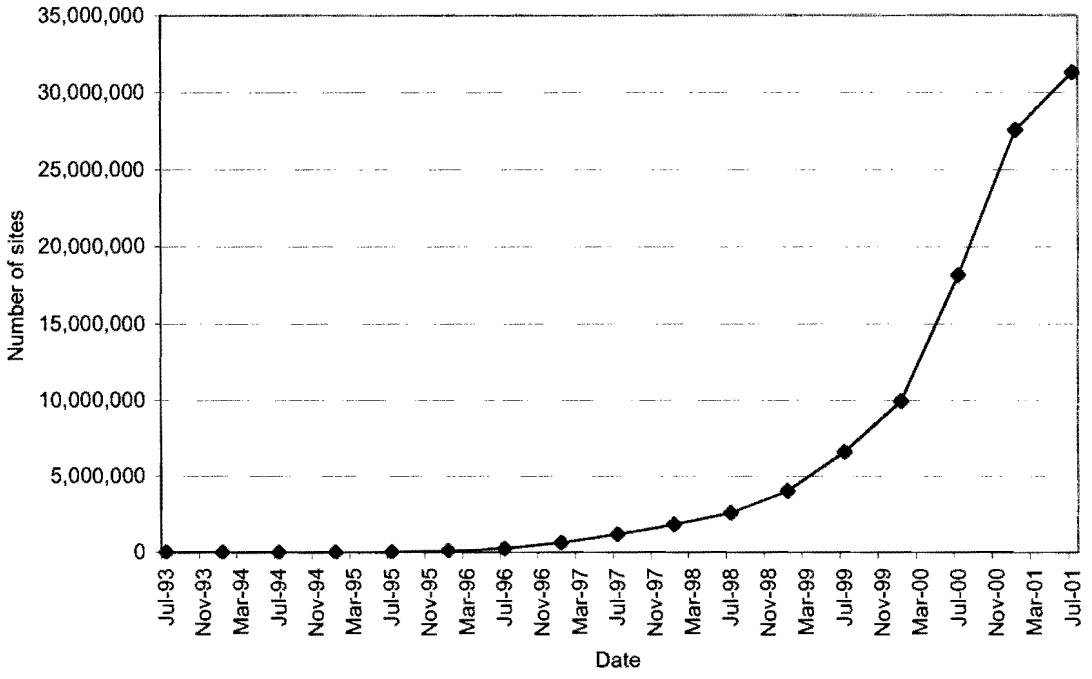


Figure 3: World Wide Web growth

It is becoming increasingly difficult to count the number of hosts on the Internet. Early attempts were based on the number of registered IP addresses. Modern techniques such as Network Address Translation (NAT) and Dynamic Host Configuration Protocol (DHCP) allow multiple hosts to share the same IP address space. Merely counting the number of registered IP addresses will therefore no longer yield the number of hosts on the Internet. Modern counting techniques are therefore based on statistics.

Chapter 3

3. CLIENT/SERVER VS. PEER-TO-PEER

3.1. Introduction

It is only recently that much research effort had gone into developing and improving peer-to-peer application models that could provide numerous benefits over the client/server model. In this section the conventional client/server application model is discussed and compared with an alternative peer-to-peer application model.

3.1. Client/server model

The client/server application model is the conventional and well-known model mostly utilized for developing Internet-based applications. The highly successful World Wide Web (WWW) service is based on the client/server model. Most Internet applications have traditionally been implemented with some form of a client/server model.

In the client/server model [figure 4] the application is divided into client and server components. The client components execute on a large number of end-user hosts, while the server components execute on a small number of dedicated centralized server hosts. The server components provide application services to the clients. The server is therefore a critical element which the application cannot function without.

Multi-tier architectures is an advanced form of the client/server model. The most common multi-tier architecture is the 3 tier architecture with a separate data, business and presentation tier. Each tier presents a logical part of the application, but the application is still distributed on hosts which perform

either a client or a server role. The data and business tier is usually distributed on server hosts, while the presentation tier is distributed on client hosts.

Communication between clients and servers is facilitated by routers which determine the communication path. Routing is however only performed on the low-level network layers and is completely transparent to the application level.

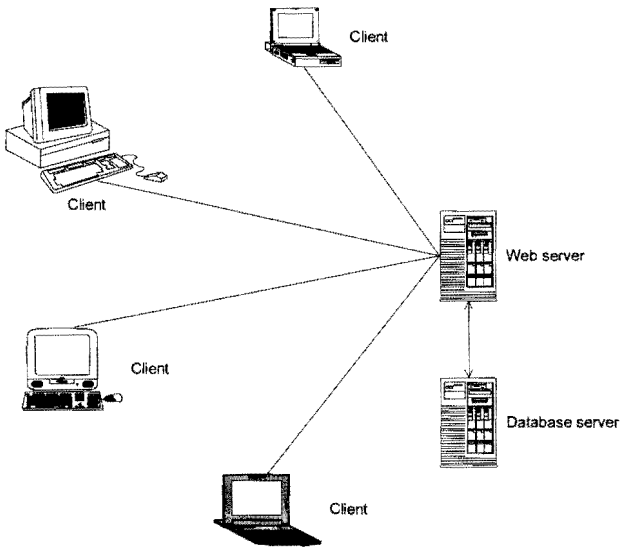


Figure 4: Client/Server model

3.2. Peer-to-peer model

The distinction between client, server and router becomes blurred with the peer-to-peer model [Lee 2003]. Each host performs the tasks of both a client and a server. Routing is still performed on the network layer, but on top of that, it is also performed independently on the application layer to form an overlay network.

In a peer-to-peer application there is no distinction between client and server hosts. Although the application may still be logically subdivided into client and server components, each host contains both components and performs

client, server and routing roles. Each peer has equal capabilities and the application does not rely on the existence of any specific host.

The peers are linked to each other to form a peer-to-peer network. The peer-to-peer network is sometimes referred to as an overlay network, because it is a logical network on top of the physical network (like the Internet). The overlay network has its own topology and routing algorithms, which is very different from the topology and algorithms of the physical network.

There is a moving trend towards developing applications that are based on a peer-to-peer model [figure 5] due to the possible advantages discussed in the next section. The World Wide Web, with its client/server model, took away users' attention from the peer-to-peer model for a while.

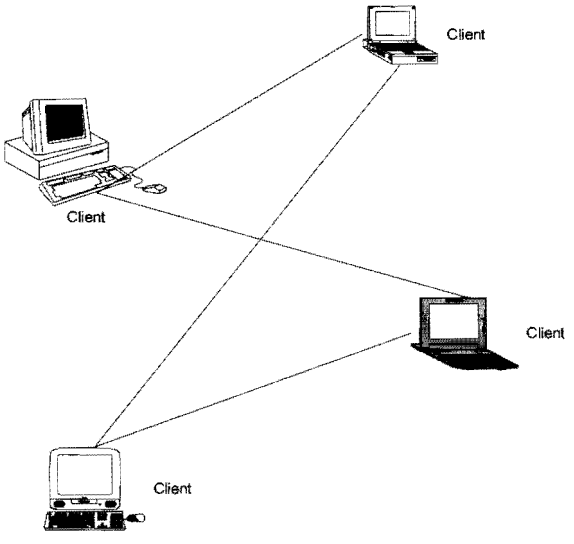


Figure 5: Peer-to-peer model

3.3. Potential of the peer-to-peer model

The peer-to-peer model has the potential to succeed in many areas where the client/server model failed. In this section some of these areas are briefly

pointed out. Chapter 11 points out the challenges that remain for the peer-to-peer model.

3.3.1. Dynamic operability

Peer-to-peer applications can keep operating transparently even though hosts join and leave the network frequently [Ripeanu et al 2002].

3.3.2. Scalability

The extent to which the application can scale is not limited by the capacity of a server. Performance of the overall system is not affected by adding additional hosts.

3.3.3. Network value

An increase in the amount of users also increases the value of the application to the users. For example, in a file-sharing P2P network more users mean a higher volume and availability of shared files and a higher probability of search matches.

3.3.4. Resource sharing

Clients in a peer-to-peer network can share resources such as files, storage space, bandwidth and processing power and therefore make more effective use of available resources. In a client/server model idle/unused resources on client hosts are wasted.

3.3.5. Collaboration

Applications such as instant messaging, Internet telephony and remote desktop control enables peers to interact and collaborate with each other.

3.3.6. Load balancing

Networks can employ mechanisms to distribute load or resource usage amongst peers dynamically or to relocate content to high demand areas of the network.

3.3.7. Redundancy and fault tolerance

Information can be replicated among many peers, causing a high degree of redundancy. This, in turn, increases fault-tolerance and availability: The failure (or attack) of a single node does not influence the availability of services or content, because it is available from other nodes. There is no single point of failure.

3.3.8. Content based addressing

In the WWW model content is located mainly based on their location (URL). In peer-to-peer networks addressing can be performed based on content, making the actual location of the content transparent to the user.

3.3.9. Dynamic indexing

Traditional web search engines keep an index of content that it discovered the last time the site was evaluated. This list is static because it does not reflect the current online status of the site. P2P networks can maintain a (possibly distributed) list of content that is currently available to the network.

3.3.10. Improved searching

Searches in a peer-to-peer network can be distributed and executed asynchronously and in parallel. Searches cannot only be used to discover resources, but also to discover users, allowing peer-to-peer collaboration independent of a service provider.

3.3.11. Anonymity

Peer-to-peer networks have the potential to protect the anonymity of people providing, holding and consuming sensitive information [Clarke et al 2002].

3.4. Contrasting characteristics

Both the client/server and peer-to-peer models have advantages and disadvantages. In this section we discuss the contrasting characteristics that were identified.

3.4.1. Fault tolerance

Server hosts are connected and available 24/7. The server hosts should be active in order to provide services required by the application. If the server host is down then the entire application cannot function even though a large number of client hosts might still be active. Client hosts leave and join the application at will.

In P2P applications a large numbers of peers join and leave the network frequently [Ripeanu 2002]. The application does not rely on the presence of any specific host in order to function. The application is therefore more fault-tolerant.

3.4.2. Host name and IP address

In the client/server model the server node usually has a static IP address with a friendly DNS name mapped to it. The client hosts usually have dynamic IP addresses assigned by a DHCP server and translated into public addresses on the network border via the NAT protocol. Client hosts also does not have permanent DNS host names registered.

In P2P applications peers have much the same characteristics as the client hosts in the client/server model. Peers have dynamic IP address and do not have DNS names assigned to them. Peers are located behind firewalls and proxies and rely on NAT to communicate on the Internet.

3.4.3. Service guarantees

Client/server applications tend to be deterministic. The services and resources made available by the application are known and can be easily measured. The server has a limited and known set of resources to offer to the clients.

Peer-to-peer applications tend to be more probabilistic. For example in resource-sharing applications there are no guarantee that a specific resource (like a specific music file) will be available at a specific time. The availability of

resources depends on the nodes that are currently connected to the network. Even though a node containing the required content is connected to the network, it might still be unavailable to the requesting node because of their distance (TTL can expire before node is reached). The challenge for peer-to-peer networks is therefore to provide a sufficiently high probability that a resource will be available, without giving any service guarantees.

3.4.4. Routing

In the client/server model routing of data is handled in packets by the physical network layers and is transparent to the application layer.

In the peer-to-peer model applications create their own overlay routing mechanism on top of the physical packet-routing network. The network topology of the application-level network is very different and unrelated to the physical-level network.

3.4.5. Scalability

In the client/server model all application services are provided by the server host. A large number of clients connect to the server in order to consume these services. An upgrade of server hardware and resources is usually required in order to support more clients. The server hardware therefore seems to be the limiting factor for scalability. Other techniques such as load balancing can also be utilized to improve scalability, but the challenge remains on the server-side.

P2P applications do not suffer from the central server bottleneck problem. With P2P applications the network bandwidth tends to be the main limiting factor for scalability. An increase in hosts leads to an increase in network and routing activity. The amount of hosts can only increase up to a point where the available bandwidth can no longer sustain the load. More research is required in this regard.

3.4.6. Host role

In the client/server model each host is either dedicated as a client or as a server. Client and server hosts perform different functions. The server hosts provides application services and the client hosts consume these services.

In the peer-to-peer model each host performs client, server and routing functionality [Ripeanu 2002]. Each host has equal capabilities.

3.4.7. Network location

In the client/server model the client hosts are located at the low bandwidth edges of the network and communicate with server hosts located within the high bandwidth core of the network.

In the peer-to-peer model the peers are located at the low bandwidth edges of the network and communicate with other peers also located at the edges. P2P communication is therefore sometimes referred to as edge-to-edge communication.

Chapter 4

4. CLASSIFICATION

4.1. Introduction

In this chapter we explore and devise methods to classify peer-to-peer applications. Our first attempt [Danzfuss and Bishop 1999] was to classify web-based Internet applications. The classification presented in this chapter focus more specifically on peer-to-peer Internet applications.

The purpose of classification is to group together applications with similar functionalities and computing models. This allows us to collectively study and refer to similar application types.

4.2. Functional categories

Peer-to-peer applications can be categorized according to their functional purpose. The author has identified the following generic functional categories:

4.2.1. Resource sharing

Resource sharing refers to applications that allow various computing resources to be shared among peers. In this dissertation the author focuses on the resource sharing functional category of P2P applications, as is suggested by the title: "Resource Sharing in Distributed Peer-to-Peer Internet Applications".

Resource sharing applications can be sub-classified according to the type of resource that is shared. P2P applications can share one or more of the following resources:

4.2.1.1. Computing cycles

The Internet has a large amount of connected hosts that are "idling" much of the time. These wasted CPU cycles can be combined to create a powerful "virtual supercomputer" [Lawton 2000]. The collective wisdom [Hayes 1998] of a huge amount of inexpensive computing devices results in computation power much greater than what can be provided by the world's best supercomputer.

4.2.1.2. Storage space

The storage space provided by a single computer is fairly limited. Peers in a P2P application can donate a percentage of available storage space to the P2P network for use by other peers. This results in an almost infinite amount of possible storage space. Storage sharing allows peers to store their content on storage space provided by another peer and vice versa.

4.2.1.3. Content

Content such as music files, videos, documents etc. can be shared by peers on the network. The Napster, Gnutella and Freenet networks encourage peers to share content on the network for free. This scheme works because each peer can access content from other peers for free.

4.2.1.4. Bandwidth

A single host has a fixed amount of network bandwidth. Each peer in the network benefits from combining the bandwidth of many peers participating in a P2P network. In theory, each new peer that joins the network effectively adds more bandwidth to the entire network. In practice however it seems to work the other way around and this point therefore needs more research and debate.

4.2.2. Collaboration

Collaborative applications are those that allow users to collaborate in various ways over a network, such as the Internet. This category includes applications

such as instant messaging, audio and visual communications, shared desktops, etc.

4.2.3. Distributed processing

Distributed processing applications allow massive amounts of computation to be distributed among multiple computer nodes on the network. This category includes applications such as cycle sharing and computation. Multiple hosts can concurrently execute a computational task.

4.3. Computing model categories

The computing model refers to the degree of decentralization of the peer-to-peer application. A completely decentralized P2P application is also referred to as a *pure* or a *true* P2P application. According to [Parameswaran et al 2001] peer-to-peer applications can be categorized into the following three computing models:

4.3.1. Centralized

Centralized P2P applications are characterized by a central server system facilitating inter-communication among peer clients. The P2P application depends on the services provided by the central server and cannot exist or survive without it.

4.3.2. Decentralized

Decentralized P2P applications do not depend on centralized server or services. All communications traverse through the peers, which collaborate to form an independent virtual routable network.

4.3.3. Hybrid

A combination of centralized and decentralized topology is called a hybrid network. This type of P2P application is characterized by normal peers and powerful super peers, which act as ad hoc central servers when necessary. The

P2P network can however exist and survive without these super peers. The super nodes only exist to enhance/improve the functionality of the system.

4.4. Combined classification system

It should now be clear that a combination of the computing model category and the functional category should be sufficient to classify P2P applications into distinctive groups. Examples of classifications are therefore a *centralized resource sharing* P2P application or a *decentralized collaborative* P2P application.

4.5. Applying the classification system

The following table demonstrates how the classification system can be applied by classifying various P2P applications which we have studied.

4.5.1. Napster

The function of the Napster application is to provide users with the ability to share music files. Napster is therefore a *resource sharing* application. The type of resources shared is content. Napster is therefore more specifically a *content sharing* P2P application.

The Napster system is facilitated by the use of central services. The Napster clients connect to a central server to perform tasks such as searches. Napster is therefore a *centralized* P2P application.

Napster can be classified as a *centralized content sharing* P2P application by combining the functional and computing model categories.

4.5.2. Gnutella

The function of the Gnutella network is to allow users to share any type of computer file. Gnutella is therefore also an example of a *resource sharing/content sharing* application.

The Gnutella system does not rely on the existence of central servers/services for its operation. Super nodes exist on the Gnutella network that can act as

servers to enhance aspects such as node discovery. The survival of the Gnutella network does not depend on the existence of super nodes. Super nodes is not part of the Gnutella protocol specification and is therefore not considered for classification purposes. The Gnutella network is therefore an example of a *decentralized* P2P application.

Gnutella can be classified as a *decentralized content sharing* P2P application by combining the functional and computing model categories.

4.5.3. Freenet

The function of the Freenet network is to allow users to share any type of computer file, while ensuring privacy. Freenet is therefore another example of a *resource sharing/content sharing* application.

The Freenet system does not rely on the existence of central servers/services for its operation. Freenet is however currently enhanced by central servers for services such as node discovery. The Freenet network is therefore an example of a *hybrid* P2P application.

Freenet can be classified as a *hybrid content sharing* P2P application by combining the functional and computing model categories.

4.5.4. SETI@Home

The SETI@Home network allows users to participate in the search for extra-terrestrial intelligence by allowing each peer to process a part of the vast amounts of radio frequency data captured by the project. SETI@Home is therefore an example of a *distributed processing* application.

The SETI@Home system is coordinated by central servers/services. The system is therefore an example of a *centralized* P2P application.

SETI@Home can be classified as a *centralized distributed processing* P2P application by combining the functional and computing model categories.

4.5.5. P2P phonebook

The function of the P2P Phonebook application (presented in chapter 12) is to allow mobile users to share contact entries in their phonebooks. The P2P Phonebook application is therefore an example of a *resource sharing/content sharing* application. The content in this instance is the Phonebook entries.

The P2P Phonebook system does not rely on the existence of central servers/services for its operation. The P2P Phonebook is therefore an example of a *decentralized* P2P application. In our simulation we made use of a central server to relay messages between peers, but this was only needed because server sockets is not yet supported on most implementations of the J2ME platform. The central relay service should not be required in the "real-world" scenario.

The P2P Phonebook can be classified as a *decentralized content sharing* P2P application by combining the functional and computing model categories.

4.5.6. Microsoft NetMeeting

Microsoft NetMeeting allows users to do the following:

- Communicate to each other by sending instant text messages.
- Communicate to each other by means of voice (requires a headset) and/or video (requires a camera).
- Allow users to work together on applications by sharing their application screens and desktops.
- Send and receive any type of computer file.

NetMeeting is therefore an example of an application that facilitates various forms of *collaboration* among peers.

NetMeeting relies on a central server for providing its services. Instant messages are relayed through the central server. Other collaborative tasks such as application sharing are coordinated and facilitated by a central server. The NetMeeting application is therefore an example of a *centralized* P2P application.

Microsoft NetMeeting can be classified as a *centralized collaborative* P2P application by combining the functional and computing model categories.

4.6. P2P distributed processing vs. Grid computing

One category that needs special mentioning is the distributed processing applications, also referred to as P2P cycle sharing applications. This field of study overlaps with the Grid computing study field. The distinction between Grid computing and P2P cycle sharing become blurred as these two disciplines evolve.

Grid computing is a broad and active research field and is discussed in much detail by [Foster and Kesselman 1998]. Grid computing is a distributed system where a number of data centers (powerful servers) collaborate to provide a large-scale virtual super computer. The goal of many of the current Grid computing efforts is to develop standards that would enable various smaller Grids to interoperate and form a single international Grid. *The Grid* is therefore analogous to *The Internet*, which was also formed by allowing various smaller networks to interoperate and form a single international network.

Grid computing grabbed commercial attention in late 2003 with the launch of *Oracle 10g Database Server*. The *g* in *Oracle 10g* stands for "Grid". Oracle claims that this release is the world's first "grid-enabled" database management system. The self-managing features of Oracle 10g allows the database to

dynamically distribute processing across all available blade and clustered servers.

Chapter 5

5. MECHANISMS

5.1. Introduction

The author has identified five main mechanisms which are utilized by all content-sharing P2P applications. All of the peer-to-peer systems we studied will be described in terms of their *node discovery*, *content discovery*, *content retrieval*, *content publishing* and *content storage* mechanisms: In this chapter we present each of these mechanisms and discuss why they are important for content-sharing P2P applications.

5.2. Node discovery

The node discovery mechanism is concerned with how each node knows the names/addresses of other nodes that it can connect to. This mechanism can be further divided into *initial node discovery*, *additional node discovery* and *node caching*.

5.2.1. Initial node discovery

The initial node discovery mechanism determines how nodes initially joins (or attach themselves to) the network. A node needs to know the address of at least one other node in order to join the network. The simplest technique is for the user to supply a known node address, but this is not a transparent process and requires manual user intervention. More complex techniques are automated and transparent to the user and can even take attributes such as the distance between nodes into account.

5.2.2. Additional node discovery

Node discovery is an ongoing process and does not stop once the node is connected to the network. Each node actively tries to discover and connect to more nodes for the following reasons:

- Fault tolerance – If a connection between nodes is broken or a node leaves the network then the P2P network will be able to restore itself because affected nodes can connect (or is already connected to) other nodes which they are aware of.

- Improve performance – Search queries and other can be sent to multiple nodes in parallel for simultaneous processing. Results can then be discovered much faster.

- Increase reach – The amount of content/services available to a node depends on the amount of other nodes that it can reach. If a node is only connected to one single node, then it can only access resource which is available through that node. More paths exist when connected to multiple nodes.

- Reduce distance – Nodes can actively try to discover other nodes which are closer or better then the nodes which they are currently connected to. Determining the distance between nodes is a difficult issue and is discussed in Chapter 11.

A node can usually be configured to allow only a limited number of connections to it. This is necessary to prevent unreasonable bandwidth demands on a node.

5.2.3. Node caching

Node caching refers to the techniques utilized to store or remember addresses of nodes. Some systems uses node cache servers to keep a list of known

connected nodes. These servers can then be used by new peers to aid them in the node discovery process.

5.3. Content discovery

The content discovery mechanism determines how the client queries/searches the network to discover the required content. A centralized or distributed search algorithm is used, depending on the computing model involved. The process usually involves the following two messages:

- *Search request* – The message contains details about the content that the user is looking for. The request can also include performance related constraints, such as instructions to only return results from a host with a certain minimum connection speed. This message is also referred to as a query message.

- *Search response* – The message contains details about content that has been discovered according to the criteria specified in the search request message. This message can contain details such as the host address where the content can be found, file size, connection speed etc. This message is also referred to as a query hit message.

Search algorithms utilized by current popular P2P networks (Napster and Gnutella) are very basic and inefficient. Much research is underway to improve content discovery mechanisms, especially in a true decentralized fashion. Interesting new search algorithms for P2P are discussed by [Aberer et al 2002] and [Balakrishnan et al 2003]. Project JXTA utilizes a hybrid centralized/decentralized approach with the Query Routing Protocol [Waterhouse et al 2002].

5.4. Content retrieval

The content retrieval mechanism determines how content is transferred from its source to its destination once it has been discovered. We have identified four ways in which this can occur.

5.4.1. Embedded

The results can be embedded within the search response message. This technique can only be used when the size of the content is small and the content is likely to be accepted by the user. The P2P Phonebook system which we have developed makes use of this technique.

5.4.2. Direct

Content can be retrieved directly from the host on which it resides. The search response message contains the address of the node containing the content. Napster and Gnutella is examples of P2P systems that uses this technique.

5.4.3. Parallel

Content can be retrieved in parallel from multiple hosts. This is possible when the same content is available from multiple hosts or when pieces of content are distributed (striped) across multiple hosts. Certain implementations of the Gnutella client allow parallel downloads.

5.4.4. Relayed

Content can be relayed (or routed) from the originating host to the destination host through the P2P network topology. With this technique peers do not know the address of the originating host. The content is relayed through the P2P network in the same fashion as the query messages. Freenet is an example of a P2P system that uses this technique.

5.5. Content publishing

The content publishing mechanism determines how new content is added to the network. New content can be published in the following two ways:

5.5.1. Sharing local store

New content can be published on the P2P network by adding it to the local store or making a folder on the local file system available to the network (sharing). This technique is used by Napster and Gnutella.

5.5.2. Publish to the network

Another method is to send the content with a content publish message into the network. The content can therefore be stored on other nodes, depending on the mechanism utilized. This technique is used by Freenet and OceanStore.

5.6. Content storage

The content storage mechanism determines how and where published content is stored in the network and when they are deleted or replaced. Content can be stored in the following locations:

5.6.1. Local store

Content can be stored in the local store on the node that has published it. The local store can be in the form of a file system, database, XML or some other specialized storage system provided by the P2P application.

5.6.2. Replicated

The published content can be replicated among multiple hosts to provide features such as improved availability and parallel downloads.

5.6.3. Striped

Content can be broken up into pieces (stripes) and these pieces can be stored on separate hosts. The pieces can also be replicated among hosts. OceanStore is an example of a system that utilizes this technique.

In some P2P systems, such as Freenet, the user is not even aware of the contents of his local store. Freenet encrypts the contents of the local store and does not provide the user with information about what is stored on his host.

Chapter 6

6. NAPSTER

6.1. Introduction

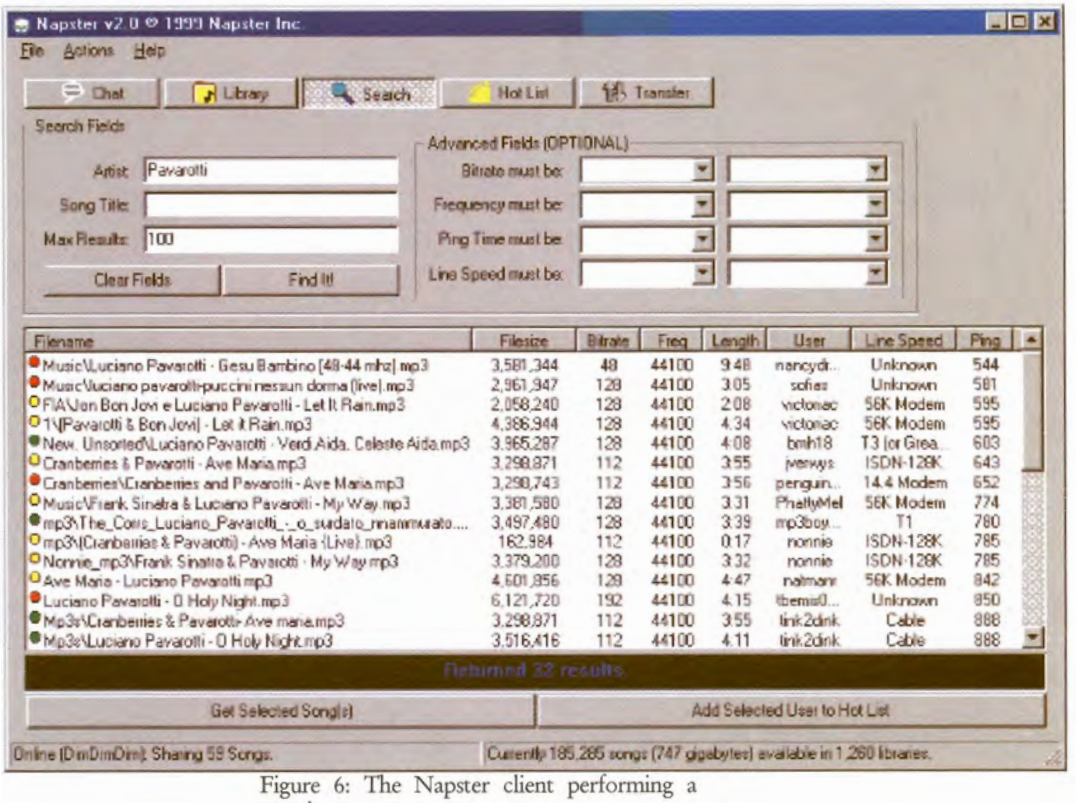


Figure 6: The Napster client performing a search

Any peer-to-peer discussion would be incomplete without mentioning Napster. Napster is (or rather was) a very popular, albeit controversial, MP3 *file sharing system*. Northeastern University undergraduate, Shawn Fanning, invented Napster's first incarnation in early 1999.

At its peak, Napster had about 70 million registered users of whom 1.57 million were accessing the service simultaneously [Lee 2003]. Another source [Stern 2000] reports that the Napster network was at some stage sharing approximately 10,000 music files per second and 100 users tried to connect per second.

MP3 files are audio (usually music) files that are compressed using the *MPEG-1 Audio Layer 3* lossy compression algorithm [Macedonia 2000]. This technique allows for high quality audio (44.1 KHz) to be encoded and stored in files that are compressed by a factor of 10, about 1MB per minute [Barkai 2001]. This provides a significant reduction in file size, compared to WAV files and CDs, which in turn make it feasible to distribute these over the Internet. With the increased availability of even faster Internet connections for home users (like DSL, cable and satellite) and other technological advances, the next big thing might be the distribution of high quality full-length movies (DVDs) over the Internet.

Napster caused quite a stir in the music industry (and otherwise), because it can and was used to distribute a substantial amount of copyrighted material. This led to a contributory copyright infringement lawsuit against Napster, which raised numerous legal issues. Sony was excused from a charge of contributory copyright infringement a few years before on its supplying VCRs to members of the public who videotaped copyright-protected TV broadcasts [Graham 2000].

The technology and mechanisms utilized by Napster are by no means revolutionary, but their popularity and controversy highlighted the power and potential of peer-to-peer networking. To prove that the file-sharing idea is nothing new, Napster is frequently compared with NFS (Network File System) developed by Sun as early as 1980 [Fox 2001].

Napster is not an Open Source project, which means that the source code, protocols etc. are not available for public study and scrutiny. This study is therefore limited to information that is made publicly available and to information from other studies. This study focuses on the first incarnation of Napster, since the new Napster service was not yet completely available at the time of study.

6.2. Node discovery

Napster nodes do not connect to each other in an overlay network structure like other true peer-to-peer systems do. The node discovery mechanism of Napster is a client/server mechanism. Napster peers connect to the network by connecting to a known central server when the client starts up. The client software knows the IP address or URL of the central server.

The central server is only a logical central server, because there is more than one physical server. Some sources [Graham 2000] record that there were in fact more than 150 physical servers. The large amount of physical servers was necessary to provide scalability and fault tolerance for the logical central server.

6.3. Content discovery

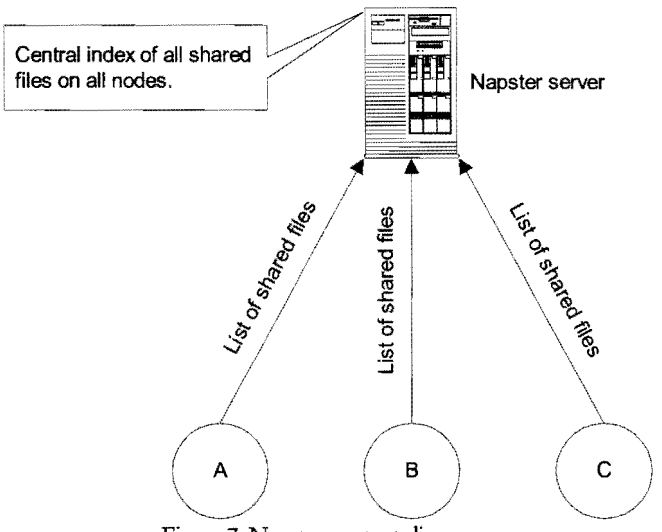


Figure 7: Napster content discovery mechanism

When run, the Napster client software connects to the central server and supplies the server with a list of MP3 files on the user's PC which the user is willing to share. The central server therefore has knowledge of all the available MP3 files on all the currently connected nodes. The user can then search for MP3 files by sending a detailed search request to the central server. The server will then respond with a list of matched entries as well as the nodes where each file resides. Content discovery is therefore also a centralized process.

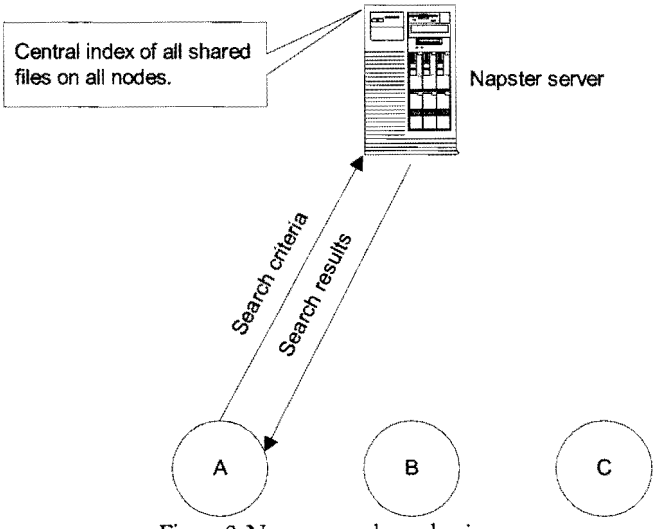
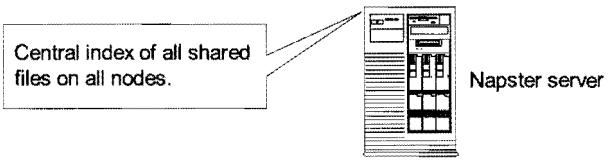


Figure 8: Napster search mechanism

Napster only supports sharing of MP3 files. Because the Napster system is only concerned with music files, it is able to use a highly efficient search mechanism. Each MP3 file contains meta-data (an ID3 tag), which describes information about the track, such as title, artist, album, production year, genre and comments. The ID3 tag is stored in the last 128 bytes of the MP3 file. The metadata descriptions of the content and the ability to centrally index and search the metadata make it easy for users to find exactly what they need. The added value of metadata for files lies at the heart of the Semantic Web, a vision of the W3C Web Consortium related to P2P [SymanticWeb].

6.4. Content retrieval



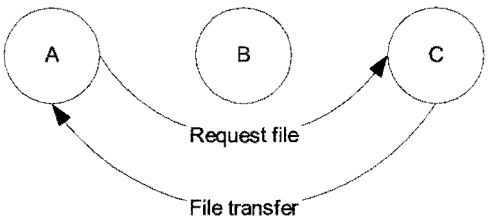


Figure 9: Napster content retrieval mechanism

The search results from the content discovery mechanism can contain multiple entries. Each entry contains the meta-data as well as the location (IP address) of the node where the content is located. Napster can also perform ping requests for each retrieved node in order to determine latency to each node. The user can then select the node from which he wants to retrieve the content. The file will then be downloaded directly from the node containing the content. Communication between the two nodes happens without server intervention and is therefore peer-to-peer. The server does not have any knowledge about the file exchange that is taking place.

6.5. Content publishing and storage

Users usually copy music from CDs (which is called ripping) and then encode them into compressed MP3 files. The files are then stored on the user's local hard drive in the same way that any other file would be stored. The user then

chooses to share MP3 files in certain directories on his PC with other Napster users.

When the user downloaded a file from another PC via the Napster network, it is by default saved in a folder that is shared with other users. This has the effect of content that gets replicated throughout the network. The user is however in control of this process and a node is not obliged to re-share content which he downloaded.

Napster does not host the MP3 files on its servers; it is rather hosted on the hard drives of the individual nodes. Napster only provides the ability for clients to locate the files they want easily. There is no special compression or encryption applied to these files. The user has the ability to specify which files or folders he wants to share. These files are indexed according to their metadata and only this index information is sent to the central server. In this regard Napster can be compared to a web search engine such as [Google], which maintains a central index of resources.

6.6. Discussion

Perhaps the biggest disadvantage of the Napster system is its reliance on a central server for operation. This is why it was possible for a court order to shut down the entire system.

Napster can be classified as a *centralized content sharing* peer-to-peer network. The Napster system survival relies on the availability of the central server to provide its clients with node and content discovery services. This makes the system vulnerable to many attacks such as legal attacks (closing down the Napster servers) and denial-of-service attacks.

6.7. The new Napster service

Since Napster was eventually forced to close its free music sharing service, it constructed a new service. The new service requires subscribers to pay a small

monthly fee. Napster is in turn buying licenses from record companies that will allow them to share copyrighted material legally on the network.

Napster invented a new secure file format (NAP) that allows them to control the usage of downloaded files. With this file format a user can replay downloaded music files only on a specific computer. The more recent Microsoft Windows Media Audio (WMA) file format has similar copy protection features.

[Lee 2003] estimates that after the downfall of the original (free) Napster service, more then 50 other systems has taken its place.

Chapter 7

7. GNUTELLA

7.1. Introduction

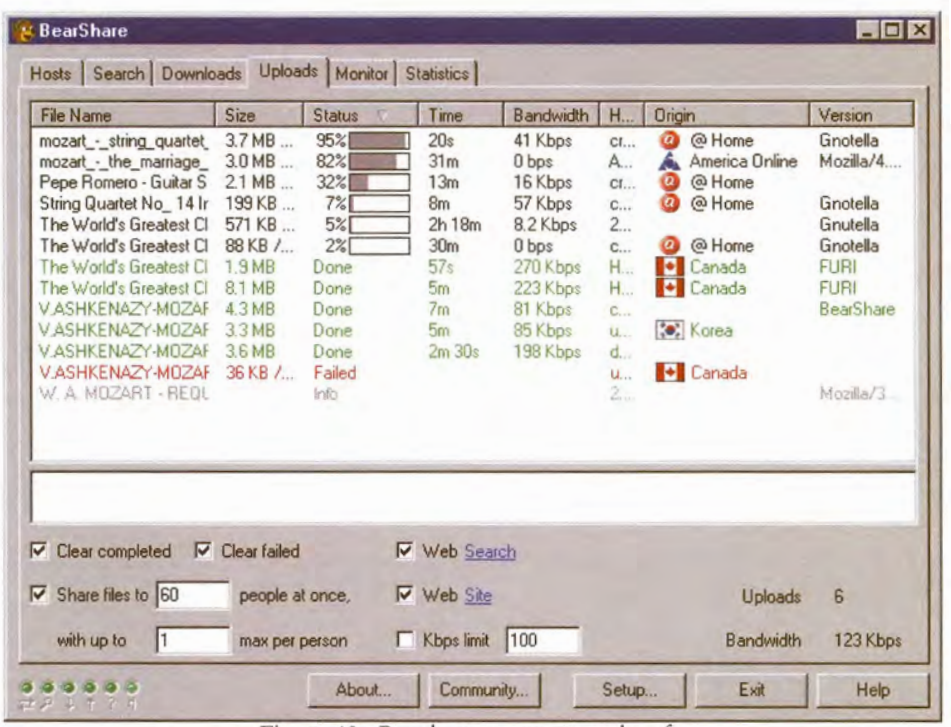


Figure 10: Bearshare as an example of a Gnutella client

The Gnutella system was first developed by Nullsoft, but later abolished due to its potential legal threats. Gnutella allows users to share copyrighted material illegally, just like Napster. The protocol leaked out to the public domain and was subsequently adopted by open source initiatives. Gnutella therefore became an open protocol specification, available for public scrutiny. Many individual developers and organizations are developing client software

that conforms to the Gnutella protocol [Gnutella]. Examples of existing Gnutella clients are [Bearshare], [LimeWire] and [JTella].

7.2. Node discovery

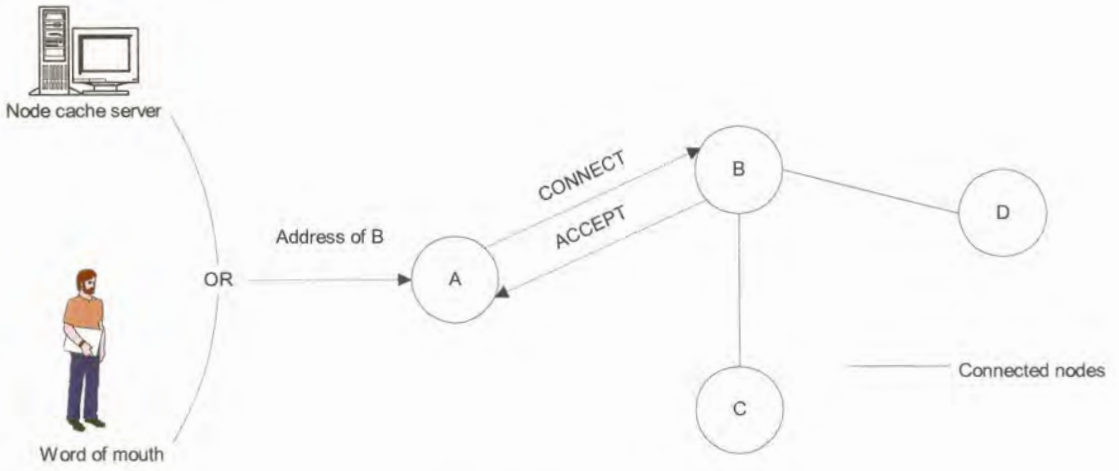


Figure 11: Gnutella initial node discovery mechanism

A Gnutella node connects itself to the network by establishing a connection with another node currently on the network. The acquisition of another node's address is not part of the protocol definition, so it is up to the client implementation to determine a mechanism.

Most client implementations obtain an address by contacting special nodes (cache nodes), which contain a list of currently attached nodes (like http://www.gnutella-hosts.com). This is a centralized approach, since the client application needs to be aware of the address of the (central) cache servers. If the cache servers fail (or are forced to close down by court order), new clients would not be able to automatically discover an initial node to attach to. Users can, however, manually connect to a known node address, which they discovered by some other means, such as word-of-mouth, newsgroups, emails, web sites etc.

It seems that there is currently no effective mechanism for automatically discovering the initial address of an existing node on a peer-to-peer network. This process is also known as the bootstrap mechanism. An effective bootstrap mechanism should not rely on the client having to know the address of a cache or other server. The mechanism should also not rely on a centralized approach. Further research in this regard is needed.

Once the address of another node on the network is known, a CONNECT request message is sent to the node. The node replies with an ACCEPT or REJECT message. The connection can be rejected for a variety of reasons such as: the maximum number of allowed connections is exceeded or the node uses a different, incompatible protocol version.

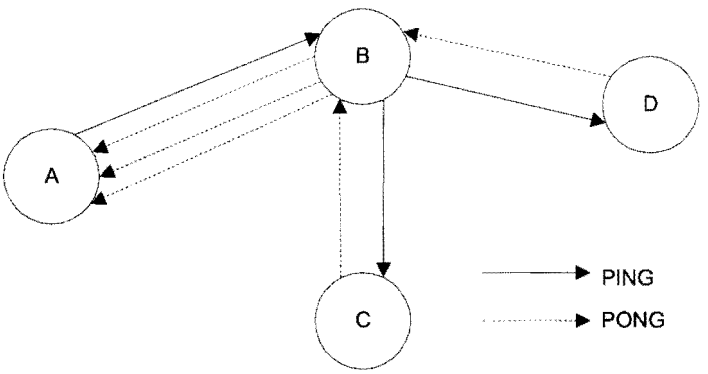


Figure 12: Gnutella automatic node discovery mechanism

Once the node successfully attached to another node it can continue the node discovery mechanism by sending PING messages to its attached nodes. Each node receiving the PING message will reply with a PONG message and forward the PING message to all of its attached nodes. The PING message (like all other messages) has a TTL (time to live), which is decremented at each node and therefore prevents endless loops.

A PONG message is sent in response to a PING message. The PONG message contains the address of a connected node as well as the amount of data shared by that node. Multiple PONG messages can be sent in response to a single PING message, which allows a node to send cached node address information.

7.3. Content discovery

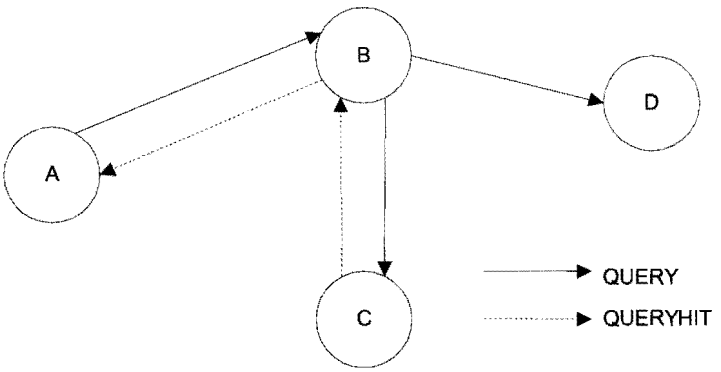


Figure 13: Gnutella content discovery mechanism

The Gnutella network can share any file type. Content is discovered by utilizing a distributed search algorithm. A node starts the search process by sending a QUERY message to its attached nodes. The QUERY message contains a *minimum speed* and a *search criteria* field. The query message propagates through the network in the same way as the PING messages, until the TTL field expires.

Upon receiving a QUERY message, each node searches its local store to determine whether it contains a matching file. The process by which the *search criteria* field is matched with files in the local store is not specified by the protocol. Most client implementations search for file names that closely match the search criteria fields, while others also search meta-data such as the ID3 tags of MP3 files. If a match is found, the node responds with a QUERYHIT message. A node should only respond with a QUERYHIT

message if its bandwidth is greater than the *minimum speed* field dictated in the QUERY message. QUERYHIT messages are routed along the same path that it was received by.

7.4. Content retrieval

The QUERYHIT message contains: the address of the node that hosts the content, a unique ID, the file size and the file name of the matching file. Files are downloaded via the HTTP protocol. The node sends an HTTP GET request directly to the node containing the file. Downloads therefore occur directly between the sending and receiving nodes and are not routed through the network as is the case with the other messages.

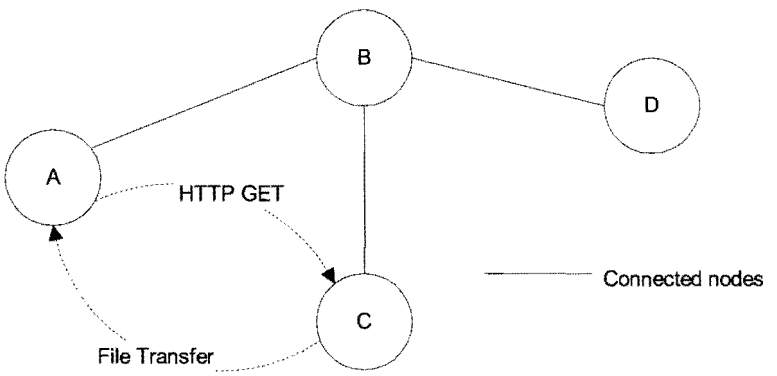


Figure 14: Gnutella content retrieval mechanism

The Gnutella protocol also entails a mechanism that allows for firewalled nodes to participate in file exchanges. Nodes behind a firewall cannot usually accept incoming HTTP connections (port 80) from outside; they can only initiate HTTP connections to the outside world. Gnutella nodes that want to retrieve files from a firewalled node can send a PUSH message to a firewalled node. The PUSH message includes the file ID and the address of the node that wants to "pull" data. The firewalled node can then initiate a direct HTTP connection with the sender of the PUSH message from within the firewall.

7.5. Content publishing and storage

Any user can publish content in the Gnutella network by placing it inside a directory that is shared for the network. Each user has control over the content on his drive and has full knowledge of what is stored on his node. Files are saved in the usual way in the file system. Files are not specially encrypted or compressed for the Gnutella network.

Because there is no inherent use of meta-data to ease content discovery, the Gnutella network relies on the use of descriptive filenames.

7.6. Discussion

Many people believe that peer-to-peer networks like Gnutella cannot be stopped, because they do not rely on a central server. The Gnutella network is however not very tolerant towards misbehaving nodes. A misbehaving node can be specifically crafted to disrupt the network. Misbehaving nodes can be introduced by organizations concerned with the distribution of spurious content, such as copyrighted material and otherwise objective content. Such an attack has been suggested by [MediaDefender], which also launched similar attacks against the Napster network.

Unlike Napster, the Gnutella network does not rely on a single server for its operation and survival. The initial node discovery mechanism is complemented by a centralized approach, but the survival of the network does not depend on it. Gnutella is an example of a true decentralized peer-to-peer network. This makes the network highly fault tolerant, scalable and survivable.

A virus called "W32/Gnuman Worm" (also known as Mandragore) has been reported [Edwards 2001] which specifically targeted users of the Gnutella network. This is the first malicious code known to the author to specifically target peer-to-peer networks. Fortunately the Gnuman worm did not include a destructive payload. The Gnuman worm points out the vulnerability of

existing peer-to-peer systems and the importance of security in peer-to-peer system design.

The Gnutella network is a very active and dynamic network. Studies [Ripeanu et al 2002] have shown that 40% of all nodes leave the network within four hours and only 25% of the nodes remain active for more than 24 hours. In a period of seven months the network has grown 25 fold.

The Gnutella network has been criticized for the large amounts of network traffic that it generates. Research done by [Ripeanu et al 2002] estimate that the Gnutella network generated 330 terra-bytes of traffic (excluding file transfers) in December 2000. This would account for 1.7% of all traffic over the U.S. Internet backbone during this period.

There is no security and anonymity built into the Gnutella network. Researchers [Ripeanu et al 2002] were easily able to extract topology and other information from the Gnutella network by developing a simple network crawler. This further demonstrated that the Gnutella network is extremely vulnerable to various attacks.

Chapter 8

8. FREENET

8.1. Introduction

Freenet is another example of a true decentralized peer-to-peer network and it is a very promising system. It provides a global distributed storage system with very unique properties, such as anonymity, authenticity and integrity. In practice, however, the current design and reference implementation does not perform very well and is not as useful as other systems, such as Napster and Gnutella.

Ian Clarke first invented Freenet in 1999, while still an undergraduate student at the University of Edinburgh. Freenet is further developed and maintained by the "Freenet Project Inc" [FreenetProject], an open source initiative.

Freenet is a work in progress that is still in its early beta stages of implementation. This study focuses on the version that was available and stable at the time of study, which is version 0.3.

The Freenet system was designed with the following goals in mind:

- The system should have no element of centralized control or administration.
- To protect the privacy and anonymity of information producers, consumers and holders.
- To provide global data storage that can survive many potential threats.

- Resistance to information censorship.
- High availability through decentralization and redundancy.
- To protect data integrity and reliability through the use of cryptography.
- To make efficient use of available storage by automatically replicating and deleting files in order to adapt to usage patterns.
- To allow any type of file to be published.

8.2. Node discovery

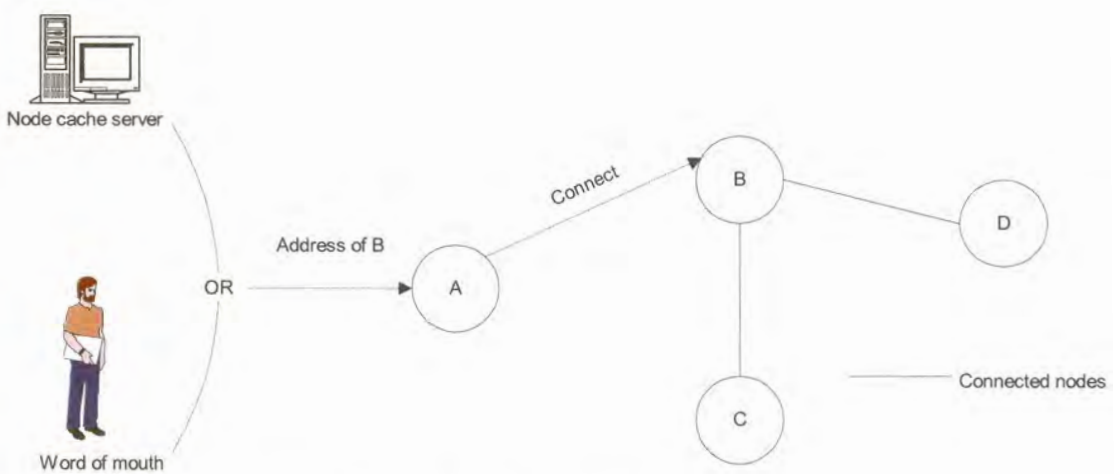


Figure 15: Freenet initial node discovery mechanism

In order for a peer to join the network (bootstrap), he needs to discover at least one other Freenet node. In the version 0.3 implementation there is no automated mechanism. The initial node discovery is a manual process. The user gets the details of another node by means such as word-of-mouth, email, web sites and news groups. Once a node is connected to the network it can

then discover additional nodes by sending node discovery requests to the connected node. The node can then reply with details of other nodes that were previously discovered.

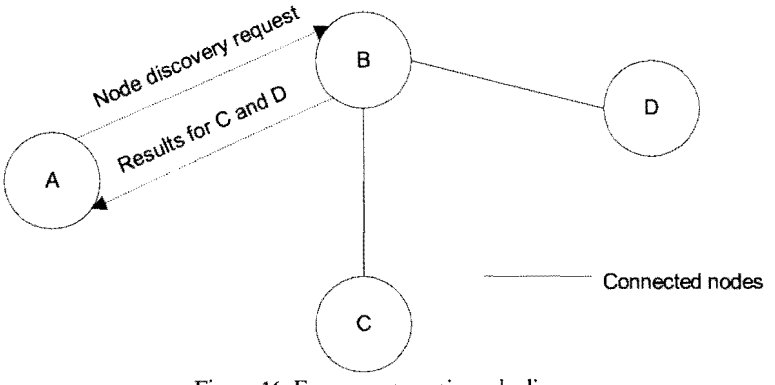


Figure 16: Freenet automatic node discovery mechanism

To ease the process of the initial node discovery, the version 0.3 implementation keeps a central list of available nodes. When a node starts up, it sends its details to the (known) server and in exchange receives the details of another node from the server.

8.3. Content discovery

All content on the Freenet network is identified by unique keys. The user needs to know the key of the content he wishes to retrieve. The key must be found by means such as word-of-mouth, email, web sites and news groups. There is currently no content search mechanism built into the system, which means that the user cannot search for content keys.

There are currently two main types of keys defined.

- *Content Hash Key (CHK)* - Each document contains a CHK. The CHK is a low-level data storage key, which is generated by performing a hash function over the contents of the file. This mechanism has the

advantage that similar documents put into the network by different users will be automatically coalesced, since each user will calculate the same key. Another advantage is that the integrity of a file can be easily verified against its CHK. The user can therefore be certain that the file will contain the intended content and that it has not been tampered with. The hashing function used is SHA-1 secure hashes. SHA-1 collisions are considered nearly impossible. Therefore it can be assumed that each unique document will have a unique CHK key.

- *Signed Subspace Keys (SSK)* – These keys are used to set up a personal namespace that anyone can read but only the owner can write to. This is achieved by utilizing public/private key cryptography and user-readable names. The SSK is generated by first hashing the private key and friendly description independently. The results are then hashed together. SSKs are typically used to store indirect files, which contain pointers to CHKs, rather than storing the data itself. The advantage of SSKs is that they facilitate trust by guaranteeing that the same pseudonymous person created all files in the subspace.

8.4. Content retrieval

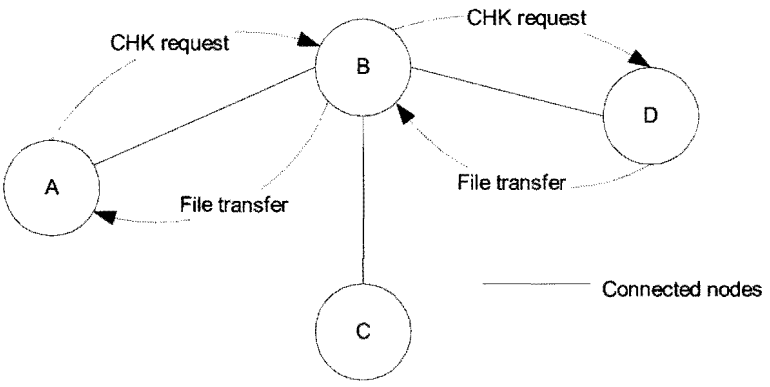


Figure 17: Freenet content retrieval mechanism

When the user wants to retrieve content, he first looks up the key in his local data store. Unlike Gnutella, a Freenet node does not blindly forward requests to all of its connected nodes. If the key does not exist locally, he sends a request to the neighboring peer who is most likely to have the content. This is determined by examining the nodes' routing table for a node with the closest matching key. Upon receiving the request each node in turn looks for the key in its own data store and forwards the request to its most likely neighbor if it does not exist in the local store, until the content is discovered at a node.

Each message has a unique ID (randomly generated by the initiating node) and each node keeps track of the message ID together with where the message originated from and whereto the message was sent. This information is used to prevent message loops. If a node discovers a loop (receives a message it already forwarded), it will initiate a backtrack message and subsequent nodes will try to forward the message to the second-best node that is likely to have the content.

Once the content is found at a node it is sent back via the same route that the original request came from. Each node sends the response back to the node where it received the request, updating its routing table with the ID of the content holder. When a node receives the response he might also save a copy of the data in its local data store. This allows for the data to be moved and replicated through the network.

An important property of this mechanism is that no node is aware of who the original initiator was. There is no way for a node to figure out if the requesting node is the originator or just merely relaying the request on behalf of another node. All communications between nodes are also encrypted using public/private key cryptography.

8.5. Content publishing

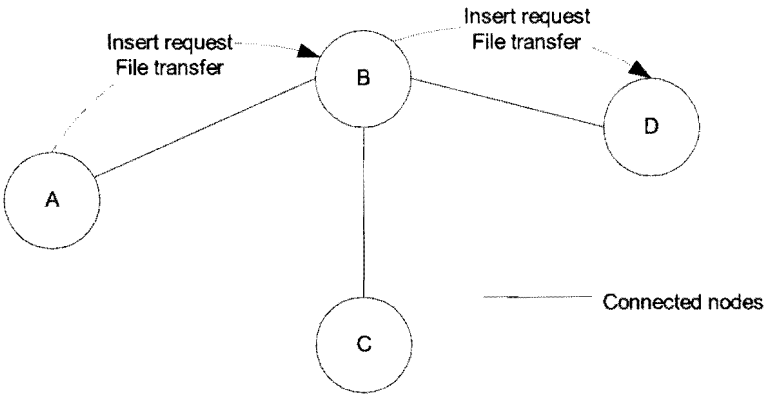


Figure 18: Freenet content publishing mechanismure

All nodes have the ability to add content to the network. The node first generates a hash key (CHK) of the content he would like to add to the network. Optionally a friendlier descriptive key (KSK or SSK) can be generated. An insert request with a small TTL value is then sent to neighboring nodes. The insert request propagates through the network the same way in which a retrieval request would, until the TTL expires. The data is then sent and stored in multiple nodes along this path. Content with near-similar keys will therefore be clustered at the same nodes and the routing algorithm can therefore determine where content are likely to be located.

There is no way for a node to figure out if the request was received from the originator or from a node that was merely relaying it on behalf of another node. This property gives the content publisher the same anonymity as the content consumer.

8.6. Content storage

There is no central storage location. The content is distributed among different nodes. Each node can decide how much disk space it would like to contribute to the network.

The storage is called a data store and is organized as a finite stack. New data is pushed on top of the stack. When the stack gets full, the least used data (determined by the number of requests) gets deleted from the stack to make space for the new data. Unlike other systems, such as OceanStore [Rhea et al 2001], Freenet does not explicitly try to guarantee permanent data storage. Because disk space is finite, a tradeoff exists between publishing new documents and preserving old ones. It is however anticipated [Clarke et al 2002] that Freenet will eventually attract sufficient resources from participants to preserve most files indefinitely.

This mechanism has the effect of tending to move data across the network to locations where there is a greater demand for it. Content in high demand will also be replicated more frequently across the network, causing an inherent load balancing mechanism.

Freenet makes it hard to discover exactly which node stores which files. A node does not have any knowledge of what is stored in its data store. This is achieved by encrypting the data store. The encryption keys can be distributed along with the CHK keys. The owner of the node could therefore not be held responsible for the content saved in his data store.

8.7. Discussion

The current initial node discovery mechanism needs further investigation. It would be helpful if there could be an automated or transparent way of discovering the initial node. The temporary centralized solution is undesirable, because it introduces a central point of control, failure, administration or attack.

Although systems like Napster and Gnutella do not explicitly replicate and move information throughout the network, the way they are used does have the same effect. When people download information from the network, they store it on their local hard drives and share it with other clients. Although

anonymity is not as good as in the Freenet network, Napster and Gnutella also has the property that makes it difficult to figure out who originally put the content onto the network.

The efficiency of current peer-to-peer networks, especially the Freenet network, depends on the network having sufficient permanent nodes. Transient nodes (nodes that join and leave the network frequently) are not very well supported by Freenet.

The first implementations of Freenet relied on a manual (human) process for discovering an initial node. The address of a node needs to be discovered by word-of-mouth or from web sites, mailing lists etc. Version 0.4 includes a node announcement mechanism that allows nodes to find each other through a list of seed nodes.

The network is non-deterministic. There is no guarantee that a user will be able to retrieve content that exists in the network. This will depend on the proximity of the data relative to the requesting node.

An interesting feature of the current reference implementation of Freenet is the existence of a web conduit called Fproxy. The Fproxy client provides the user with a familiar web-based user interface. The web interface is provided by a local web server (the conduit), which also implements the Freenet protocol and connects to the Freenet network. A similar conduit technique can possibly be used to extend Freenet to mobile devices in the future.

Many of the claimed benefits that the Freenet design might have are based on the assumption that connected nodes are close to each other. For instance, the phenomenon that data moves to high demand areas assumes that these nodes are close to each other. With the current node discovery mechanism this is not the case. A node to connect to is randomly selected from the

advertising server, irrespective of its proximity to the requesting client. The word-of-mouth method has the same problem.

The benefits of the distributed decentralized nature of the Freenet network surely came at a cost – bandwidth. The perceived performance of the current Freenet implementation is very poor for users connected via standard modems. This is due to the increased bandwidth requirements, which are imposed by the employed distributed algorithms. The increased availability of faster Internet connection like cable modems and DSL connections should partly solve this problem. The Internet2 project also promises to provide users with a significant bandwidth increase in the future.

Disadvantages and challenges:

- Each node has to process and relay the requests of other nodes, which means that bandwidth and processing power are used on each node for data that it did not request.

- Each node has to save data on its hard drive which other nodes requested and which it is not interested in.

- Messages have to propagate through different nodes (hops) in the network before reaching its destination. In a system like Napster the client knows which peer contains the required information and can directly connect to it. Simulations of the Freenet network done by [Clarke et al 2002] however suggested that on average, it would take only 2.7 hops to reach the destination node.

- There is currently no mechanism to control the amount of storage space consumed by the network on each node's hard drive.

- Freenet is not yet searchable. Content is only retrieved by means of a key.

- The current implementation of Freenet does not take the size of documents into account. A single large file might displace multiple smaller files in the case of a full data store stack. Current implementation relies on external indexing and searching mechanisms, much like the way in which the web works.

- Although SSK keys provide the potential for authors to update their files, it has not yet been implemented. The challenge still remains to devise a mechanism to ensure that all old copies get replaced

Chapter 9

9. A COMPARISON BETWEEN NAPSTER, GNUTELLA AND FREENET

9.1. Introduction

In the previous chapters the Napster, Gnutella and Freenet networks were discussed in terms of node discovery, content discovery, content retrieval, content publishing and content storage mechanisms. In this chapter these systems are compared by discussing some additional important differences.

9.2. Content search criteria

The search criteria mechanism makes it easy for the user to search for specific content that he is interested in. Napster is the only network with a good content search criteria mechanism. This is done through the metadata ID3 tags.

Gnutella only allows the user to search on file names. This is not a good mechanism, because filenames do not reveal anything about the actual content. It does not work too badly in practice though, because users have learned to give their filenames descriptive names and include extra search keywords in the filename.

Freenet, although superior to the other networks in many regards, does not (currently) even have a search mechanism.

9.3. Connection search criteria

The connection search criteria allow the user to filter the search results according to properties of the connection on which the content resides.

Napster allows the user to specify both connection speed/type and ping times as criteria. However, the connection speed/type is manually specified by each user when the Napster software is installed and can therefore be incorrect. For example: Many people with broadband connections (such as ADSL and cable modem) deliberately specify to the system that they only have a 33,6Kb dialup connection. This will reduce the amount of traffic to their nodes because no one wants to download files from slow connections. PING criteria are a better mechanism. PING messages are sent to the nodes listed in the search results and the user can take decisions depending on PING times.

The Gnutella network provides much the same connection criteria as Napster. The Freenet network does not have any knowledge of connection speed. All nodes are treated equally, regardless of connection speed.

9.4. Supported file types

The Napster service only allows for sharing of MP3 music files. Gnutella and Freenet allows for sharing any file type. None of the systems has any limit on the file sizes that can be shared (except for the usual OS file system limits).

9.5. Routing

There is no routing of messages in the Napster network. All communications are directly between a node and the server or between two nodes. There is no concept of forwarding or routing messages on behalf of other nodes.

Gnutella has a logical routing network. All messages are forwarded to all connected nodes until the TTL of the message expires. Response messages are routed back the same path as the request.

The routing mechanism of Freenet is more efficient than that of Gnutella, because messages are forwarded to the nodes that are most likely to satisfy the request. Response messages are routed back the same path as the request.

9.6. File download

With Napster, files are downloaded directly from the node containing the file, using a proprietary port and protocol. With Gnutella, files are also downloaded directly from the node containing the file, but the standard HTTP port and protocol are used instead. This makes it easier to communicate via proxy servers or firewalls.

Freenet operates very differently: Files to be downloaded are routed back through the network along the path it was discovered. Each passing node might choose to cache the file. A proprietary port and protocol are used.

9.7. Anonymity

The Napster and Gnutella networks cannot guarantee the anonymity for content publishers, subscribers and distributors. These networks do not even attempt to provide anonymity. With Freenet the identity of content producers, consumers and holders are kept anonymous. This is accomplished by cryptographic and (interesting) routing techniques.

9.8. Source code

Napster is a commercial effort owned by a single company and the source code is therefore not available. Napster only runs on the Windows and Macintosh platform.

Gnutella is an open protocol specification for which there are many implementations. Some implementations are open source and some are commercial and they are implemented in various languages and on various platforms. Detailed protocol specifications are therefore available as well as the source code for many different implementations.

Freenet is an open source project which is coordinated by a single organization. It is implemented in Java and therefore runs on any platform

with a Java Virtual Machine. The protocol specification and source code are therefore also available.

9.9. Computing model

Napster is a centralized system. The operation of the system depends entirely on the availability of the central services. Gnutella is a completely decentralized system. The network can survive and operate without any central services. Freenet is a hybrid network. It is essentially a decentralized system and can survive and operate without central services, but it is complemented by centralized services.

9.10. Summary

The following table summarizes the characteristics of the different peer-to-peer networks that were studied.

Characteristic	Napster	Gnutella	Freenet
Content search criteria	Detailed search criteria on ID3 metadata, including artist, title, recording bitrate and frequency.	File names	None. Only way to retrieve content is by its CHK, which can optionally be obtained from its SSK.
Connection search criteria	Connection speed and ping times	Connection speed	None.
Supported file types	MP3 music files only	Any file type	Any file type
Routing	None.	All messages are forwarded to all connected nodes until the TTL of the message expires. Response messages are routed back the same path as the request.	Messages are forwarded to the nodes that are most likely to satisfy the request. Response messages are routed back the same path as the request.
File download	Files are downloaded directly from the node containing the file, using a proprietary port and protocol.	Files are downloaded directly from the node containing the file, using the standard HTTP port and protocol	Files to be downloaded are routed back through network along the path it was discovered. Each passing node might choose to cache the file. A proprietary port and protocol

			are used.
Anonymity	None	None	Identity of content producers, consumers and holders are kept anonymous.
Source code	Commercial.	Open protocol specification. Many open source projects. Commercial clients also available.	Open protocol and source code developed by single organization.
Computing model	Centralized	Decentralized	Hybrid

Chapter 10

10. OTHER PEER-TO-PEER INITIATIVES

10.1. Introduction

This chapter briefly presents other peer-to-peer projects and initiatives known to the author. These projects are mentioned here to indicate the vast amount of research that is currently under way and the growing field of applications for peer-to-peer systems.

10.2. Project JXTA

The JXTA (Juxtapose) project is an Open Source initiative initiated and coordinated by Sun's chief scientist, Bill Joy. The aim is to create a framework that will ease the development of distributed service, which is interoperable and available to any peer on the expanded Web. The project is discussed in more detail by [Gong 2001].

Current peer-to-peer applications such as SETI, Napster, Gnutella and Freenet have the following shortcomings:

- Address a single function. Gnutella can only be used for file sharing and SETI can only be used to search for green men from other planets.

- Unable to share data between each other. There is no way that a Gnutella client can (seamlessly) integrate with a Freenet client, although both systems provide file-sharing services.

- Systems are developed to function only on specific platforms. Napster can only run on WinTel and Macs. There is no easy way for a Napster client to run on a 3G cell phone or a web-enabled toaster.

Project JXTA was born to address these and other issues.

The project is generalizing peer-to-peer functionality and focuses on creating basic mechanisms, while leaving the policy choices to application developers. JXTA leverage existing technologies such as Java and XML. The framework only specifies communication protocols (in XML) and is therefore language independent. Currently there is both a Java and C++ implementation.

The JXTA core provides the following mechanisms, which can be used to provide peer-to-peer services and applications:

- *Peer groups* - JXTA introduces the concept of peer groups, which allows users with similar interests to be grouped together. Mechanisms are defined to create, delete, advertise, find, join and leave peer groups.

- *Peer pipes* – Pipes provide communication channels among peers. Messages sent in pipes are structured with XML. A pipe is unidirectional, but two pipes can be used together to provide a bi-directional communication channel.

- *Peer monitoring* – Enables control of the behavior and activity of peers. These mechanisms can be used for management and maintenance functions such as access control, prioritization and load balancing.

Besides the core mechanisms, the framework also defines JXTA services that sit on top of the core. These include mechanisms for searching, indexing, sharing and caching.

Part of the JXTA project is the development of a command line shell. The JXTA shell functions much the same way as a Unix shell. The shell can be used to access and experiment with the functionality provided by the core and other services.

It should be possible to modify and implement current peer-to-peer networks like Napster, Gnutella and Freenet using the JXTA framework. The advantage would be greater inter-operability among these networks, easier development and deployment across a greater variety of devices.

Project JXTA also extends to wireless and mobile devices. JXME is a project aimed at providing JXTA functionality on J2ME devices, such as cell phones and PDA's (CLDC/MIDP). These devices typically have very limited capabilities. The JXTA project overcomes these limitations by introducing a JXTA relay agent. The relay peer is the one that is actually participating in the P2P network on behalf of one or more mobile devices.

10.3. OceanStore

OceanStore is a very interesting content publishing system, which is part of the Globe project [Rhea et al 2001]. Content is hashed and broken into small fragments, which are distributed among peer servers. Hashing gives it the ability to identify content uniquely and to reconstruct the original content from the fragments. Content are read-only once it is placed in the system. Subsequent updates are versioned. The idea is to have a persistent global archive of content.

10.4. SETI@home

SETI (Search for Extraterrestrial Intelligence) is a scientific effort seeking to determine whether there is intelligent life outside Earth. SETI processes large amounts of data collected from a radio telescope, by utilizing donated CPU cycles on thousands of computers connected to the Internet. The project homepage can be found at [Seti].

10.5. Entropia

Entropia is a commercial system that rents out processing power to its clients. The processing power is obtained through a peer-to-peer network of hosts that contributes otherwise wasted CPU cycles. Some of the time is donated to research projects. The Entropia business model is questionable [Delaney 2001] because there does not seem to be any real motivation for clients to donate CPU cycles to the network. The project homepage can be found at [Entropia].

10.6. Gnougat

Gnougat is an experimental decentralized system for demand-based distribution of static content [Faybishenko and Kan 2002]. The goal of the project is to improve on the Gnutella protocol by providing a content-centered approach, instead of a metadata centered approach. Gnougat improves performance by taking advantage of the uniqueness (using hashes) and fluidity (using chunks) properties of content.

Content in the Gnougat system is identified by a hash of the content. This enables the system to identify equal content (similar content will produce the same hash results). The Gnougat project tries to prove that the system can do much more to optimize itself if the uniqueness of content is easily discernable.

Unlike most other systems, Gnougat does not require the user to specify his connection speed or quality. Gnougat uses an application layer packetization mechanism to detect the quality of network connections. Lossy connections are dropped while lossless connections are maintained.

Content is also broken up into manageable pieces, called *chunks*. Descriptor files contain the metadata describing the original file (name, description, keywords etc.) as well as the list of hash keys identifying all the chunks that can be used to reconstruct the original file. The chunk sizes are fixed in order

to ensure that different hosts calculate the same list of hashes for the same content.

The routing and broadcasting in Gnougat are similar to Gnutella. Currently Gnougat does not have any bootstrap or auto discovery mechanism (such as PING in Gnutella).

There are two kinds of query messages: text query (to discover content descriptors) and hash query (to discover content chunks). The amount of text query messages is reduced by aggressively caching descriptors based on demand (text query hits). Network traffic due to hash searches is also reduced by caching frequently requested chunks. Performance is further enhanced by downloading file chunks in parallel from multiple chunk providers.

Chapter 11

11. CHALLENGES

11.1. Introduction

This chapter discusses the challenges that need to be overcome if peer-to-peer applications are to be widely adopted.

11.2. Minimizing the distance between connected nodes

The traditional client/server model usually involved communication between the low bandwidth edge of the Internet (where clients are located) and the high bandwidth core of the Internet (where servers are located). The peer-to-peer model usually involves communication from edge to edge, since end user peers are usually located at the edge of the Internet. This can enhance or decrease performance, depending on the proximity of the connected peers. A challenge in the peer-to-peer model is therefore to minimize the distance between connected peers.

Studies performed on the Gnutella network [Ripeanu et al 2002] have suggested that the current mechanisms that determine the Gnutella network topology are highly inefficient. The study demonstrated that the topology of the overlay Gnutella network does not match very well with the topology of the physical Internet.

A big challenge is how to determine the distance between two nodes. The following are some possibilities:

11.2.1. Geographic location

Geographic location indicates the physical distance (measured perhaps in miles) between the nodes. Nodes that are geographically close can prevent Internet backbone and cross-Atlantic communications. However, geographic location might not give any indication of the performance (latency and bandwidth) of the connection. Two geographically close nodes might be connected via a satellite connection, which incurs a large latency. A high bandwidth node (DSL connection) might be connected to a geographically close low bandwidth node (56K connection), effectively slowing down the high bandwidth node.

11.2.2. Hop count

The hop count indicates the number of hops (routers) which store and forward packets between the sending and receiving nodes. The hop count gives an indication of how many networks are between the two nodes. It might be intuitive to think that more networks will increase latency and decrease bandwidth and are therefore a good measure. This depends, however, on the infrastructure of the networks that is traversed. Networks in the core of the Internet are, for example, faster than networks at the edge. The same amount of hops in the core will be traversed with less latency and more bandwidth than the same amount of hops at the edge.

11.2.3. Latency

Latency is an indication of the delay (amount of milliseconds) that it takes for a packet to detour between two communicating nodes. The latency can be a good measure of distance. However, although the latency might be low between two nodes, the bandwidth might also be low and therefore not appropriate.

11.2.4. Bandwidth

This is the data transfer speed (throughput) between two communicating nodes, measured in Kbps. Although the bandwidth between two nodes might

be high, there might still be a large latency. The latency might not be appropriate for some applications such as Internet telephony.

11.2.5. User interest

If users with the same interests are connected together, it might improve the overall performance of the network. Search matches will be resolved faster and will have to traverse fewer hops.

A common problem with all of these measures is that none of them are constant and that they tend to change dynamically. The geographic location of a mobile Internet device can change while connected, for example, a WAP enabled cell phone. The networks through which packets are routed, and therefore the hop count, changes dynamically as routers adapt to changing network conditions. The latency and bandwidth depends on the ever-available network traffic along the routed path, which is characterized by a Poison distribution.

It seems that the ultimate method of determining the distance between two nodes would be to take all five of the mentioned factors into account. Furthermore, this process should be dynamic to adapt to the dynamic change in distance. Peer-to-peer networks should actively try to discover closer nodes, but this process should by itself not incur too much overhead.

11.3. Initial node discovery

The biggest challenge for P2P systems according to [Clark 2001] is to enable devices to find one another in a computing model that lacks a central server.

An ideal initial node discovery mechanism should be able to satisfy the following properties:

- Discover the initial node without any human intervention. The user should not be required to supply the system with the address of a node manually.

- The client should not rely on the existence of a specific server. The IP address of a node cache server should therefore not be hard coded into the system. The system should therefore be able to survive without reliance on any specific server.

Current systems can only satisfy one of the above criteria, but not both at the same time. It seems that the only way to satisfy both criteria would be to use some sort of broadcast discovery mechanism.

Broadcast protocols are currently not very practical for use on the Internet. The Internet is a very large-scale network and it is impractical to broadcast discovery requests to every node on the Internet. Current routing protocols do not forward broadcast requests to its connected networks. Broadcast requests are only confined to the network where they originated from. Internet multicast might be an alternative to explore, although it seems that multicast messages would require the node to know a specific multicast address, which defeats the purpose.

11.4. Sustainable participation

Another challenge is to motivate users to join the peer-to-peer network and use it over a long period of time. The usability of most P2P networks increases with the amount of active users/peers. It is therefore crucial that a P2P application attracts as many peers as possible and to sustain a large amount of users.

The following types of participation can be distinguished:

- Voluntary participation – The users of the P2P application decide if and when they want to join and leave the network. The sustainability of the P2P application depends on the willingness of the peers to participate in the network.

- Enforced participation – The user is forced to participate in a P2P network, or the user is unaware of the fact that he is participating in a network. This type of scenario might be found in future operating system services/daemons: background processes that perform tasks by utilizing a P2P network which is transparent to the user. Sustainability is therefore enforced by the system.

P2P applications require peers to donate some of their resources, such as bandwidth, content, storage capacity and computing cycles, towards the P2P network. Users are typically reluctant to share their precious resources, unless they receive some benefit or incentive for their effort. In content sharing applications the benefit is that a user has access to a much greater repository of content (usually for free) than what would be available to him without participating in the network.

11.5. High bandwidth demand

Decentralized P2P applications place a high demand on bandwidth resources due to the overlay routing architecture. Each host acts as a router and is required to relay/forward messages and content on behalf of other hosts. If we assume that there is a cost associated with bandwidth usage, then a user will be paying for a lot of bandwidth usage which is none of his concern. The bandwidth is therefore said to be "donated" to the P2P network.

Bandwidth is not the only resource donated to the network. Other resources such as storage space and CPU cycles are also donated as mentioned in the previous section. Bandwidth is, however, a special concern to any Internet-based application, because of its general shortage and cost.

Simple decentralized search algorithms, such as those used in the Gnutella network can be extremely bandwidth intensive. Search request messages are typically forwarded to all connected nodes at each peer. An interesting decentralized algorithm that resolves this problem is discussed by [Aberer et al 2002]. The algorithm is based on a P-Grid approach and is implemented in a Gnutella-style P2P system called Gridella. Simulations done on Gridella show that P2P performance can be significantly improved by the use of proper search methods.

11.6. Firewalls

A firewall is a hardware device or a software program that protects an organizations internal network from security threats on the Internet [Schneier 2000]. This task is usually performed by restricting packets that go in and out of the organization's network. Most firewalls today are configured by closing all ports by default and only open up ports for essential Internet services such as World Wide Web (HTTP protocol on TCP port 80) and email (SMTP protocol on TCP port 23).

Peer-to-peer applications are Internet based and therefore peers communicate to each other through a TCP port. Firewalls cause a problem because most organizations block all ports other than email and web traffic. New Internet services, such as peer-to-peer applications, require a new application-level communication protocol and therefore a new TCP or UDP port.

One way to solve this problem is to require firewall administrators to configure their firewalls to allow traffic on a new port. There is currently no standard peer-to-peer communications protocol. Each P2P application utilizes its own protocols and therefore requires its own ports. Project JXTA aims to alleviate this problem by defining a generic framework and protocols. But you still have to convince all firewall administrators to open the required port, which will not happen unless the P2P service becomes widely adopted.

Another solution is to tunnel the P2P communications over HTTP, which is (almost) never blocked. New technologies such as XML-based SOAP (Simple Object Access Protocol) provide a convenient API for distributed communication via HTTP and therefore through current firewall configurations.

11.7. Service guarantees

Peer-to-peer systems are not as deterministic as client/server systems. Peer-to-peer systems can rather be described as probabilistic. The availability and quantity of resources and services depends on the current state of the network. Usually the value of the network and the service it provides increases as the amount of connected peers increases. This is because each peer donates additional resources to the network. The availability of any specific resource can therefore not be guaranteed.

The solution could be to design P2P systems in such a way that they provide a statistical guarantee, in other words that the probability of a resource being present is high enough to be acceptable to the user. Research in this regard has been conducted for the OceanStore system [Kubiatowicz 2003].

11.8. Political and social concerns

The challenges faced by peer-to-peer systems are not always just technical issues. Various political and social concerns are pointed out by [Agre 2003]. P2P systems like Freenet aim to be immune against censorship, monopoly, regulation and other instances of centralized authority. Although these goals might be technically possible the question remains if it would be politically and socially acceptable. The use of technology for committing crime is always a big concern to authorities and they would try to prevent such use or at least ensure that they will be able to control it.

11.9. Forward compatibility

Backward compatibility is easy to achieve, because you just need to ensure that the new version of your software still works with the old data format. Forward compatibility is more challenging, because you have to ensure that your current software will work with the future data formats which are currently unknown.

One interesting aspect of the World Wide Web that was pointed out by [Connolly 1998] is that web browsers are fairly forward-compatible. New HTML web pages will still work with old browsers. You might not experience all the bells and whistles of the new pages, but the basics will still be viewable. It is for example even possible to view some modern XML-based web pages in an old Netscape 1.1 browser. The W3C specifically designed HTML to be backward and forward compatible because they do not have control over the browser versions that is used by the clients. You cannot instantaneously upgrade all the browsers on the Internet to the latest version when a website with new HTML features is released.

This compatibility problem also arises with differences in browser implementation and platforms (Netscape on Linux vs. Internet Explorer on Windows) even though they claim to support the same HTML standards. We proposed a few solutions to this problem in [Danzfuss and Bishop 1999].

The same principle is valid for large scale P2P systems. Users will participate in a P2P system with various versions of the P2P client software. P2P protocols should therefore be designed in such a way that they are expandable and that they will still work in the future.

Chapter 12

12. IMPLENTING P2P ON CONSTRAINT MOBILE DEVICES

12.1. Introduction

In this chapter the author will attempt to implement a typical peer-to-peer application on mobile devices with limited capabilities. The unique challenges, attributes and possibilities of P2P on mobile devices are explored.

Most current research efforts have focused on peer-to-peer applications for desktop computers, therefore eliminating server infrastructure. The author attempts to take P2P to the next level by also eliminating desktop computers. This is believed to be a natural evolution of P2P which is in line with market trends. In the near future there might be more mobile computing devices than desktop computers (most probably in the form of cell phones).

12.2. Motivation

The following points summarize the motivation for implementing a mobile P2P application:

- In the near future there might be more mobile devices than computing devices, due to the evolution and increased capabilities of devices such as cell phones.

- Mobile computing devices are less expensive than a full-blown desktop computer and are therefore accessible to a wider audience.

- Not much research has been done in the field of mobile peer-to-peer computing.

- A mobile device is more "personal" than a Personal Computer (PC), because the user can easily carry it and always have it with him. This is ideal for P2P applications that facilitate collaboration, because the P2P application is more readily available to the end user.

Research efforts by [Lienhart et al 2002] extend media service delivery to mobile PDA devices via a P2P network by utilizing intermediate desktop PCs to process the media stream into a format acceptable to the mobile device. Project JXTA utilizes the same intermediate technique to deliver P2P services to mobile devices. Our approach is to eliminate the desktop computers entirely so that the mobile device becomes a true independent peer.

12.3. Unique aspects of mobile P2P

This section discusses some of the unique challenges, attributes and possibilities of mobile P2P applications that were discovered:

12.3.1. Limited capabilities

The capabilities of mobile devices are much constrained. Aspects such as processing power, storage capacity, communication and I/O abilities are very limited.

12.3.2. Platform diversity

There are a great variety of mobile platforms and devices with various capabilities. This makes it difficult to develop P2P applications that will run on a great variety of devices.

12.3.3. Communication reliability

Mobile roaming devices join and leave the network more often than desktop devices. This is partly due to wireless communication technology which is less reliable and stable than wired communication.

12.3.4. Testing and debugging

Testing and debugging can be difficult. Applications for mobile devices are usually developed and compiled on desktop computers, because the mobile device itself does not have sufficient resources in order to develop on the device itself. This makes it difficult to debug and test, because you are dealing with a variety of devices.

12.3.5. Mobility

Minimizing the distance between connected nodes has already been identified in chapter 11 as a challenge for P2P applications. The mobility of mobile devices makes this aspect even more challenging, because the physical distance between connected nodes are constantly changing as devices move around. A possible solution to this problem could be the utilization of location-based services (LBS) to determine the approximate distance between nodes dynamically and to adjust connections between nodes according to location information.

12.3.6. Bandwidth

Connected mobile devices are constrained by low network bandwidth (slow connections). This is because mobile communication is performed by wireless radio frequency (RF) technology. High speed communication technologies, such as fiber optics, are only available for wired devices.

12.4. The P2P phonebook application

The application that the author decided to implement is a peer-to-peer phonebook application. Most mobile devices already contain some sort of phonebook. The phonebook allows the user to store the names and numbers of his contacts. The user can then easily call up the number of a contact when it is required.

The phonebook problem can be solved with a traditional client/server approach. The phonebook entries from all the users would be consolidated into a single centralized list. The distributed peer-to-peer phonebook application would however, have the following unique characteristics:

- The user can search for numbers that are not only stored in his own phonebook, but also for numbers stored in the phonebook of another user. The user can therefore perform a distributed search. All participating users benefit from this, because they now have a much larger set of phonebook information to their disposal.

- The actual location of the phonebook information is transparent to the user. The user can do a search for "Jimmy" and the system will return his telephone number, without disclosing where it got that information from.

- The phonebook is more fault-tolerant. Chances are that the same information is available from the phonebooks on multiple devices. If one of the devices fails, the information can still be discovered from another.

12.5. Routing through phonebook entries

Mobile devices such as cell phones usually contain a phonebook list. Cell phone numbers (MSISDN) stored in the phonebook can be used to identify and address nodes on the network. This presents a unique opportunity for our mobile P2P application: The phonebook entries can be used to represent the routing topology for the P2P network. There is no need for an additional node discovery mechanism, because the entries in the phonebook of each device already serve as a list of connected nodes. Routing is performed by forwarding messages to all the nodes listed in the phonebook.

In practice it is currently difficult to utilize the built-in phonebook for routing purposes because the phonebook entries are not available to Java clients. Phonebook entries can be stored in both the phone memory and the SIM card. The SIM Application Toolkit (SAT) interface can be used to get phonebook entries stored on the SIM card, but not entries stored in the phone memory. There is also no interface between SAT and Java clients.

For our simulation we just used the phonebook entries created via our P2PPhonebook application for routing purposes and assume that these entries will eventually be obtained from the built-in phonebook list.

12.6. Platform and technology

The author has decided to implement the P2P application on the Java 2 Micro Edition (J2ME) platform. The reasons are the following:

- The Java programming language is well-known and accepted, especially in the academic community.
- The J2ME platform is currently supported on a rapidly expanding variety of mobile devices, such as cell phones and PDAs. This allows the author to present the concept theoretically as well as experimentally by implementing and evaluating it on real-world devices.
- J2ME provides a platform for running applications on devices with very limited capabilities.
- The promise of "write-once-run-anywhere" is expanded into the mobile arena with J2ME. Applications can therefore run on a variety of mobile peers and can interoperate seamlessly.

We made use of the following development tools:

- Borland JBuilder 7 Personal Integrated Development Environment (IDE) – Assists with development, compiling, debugging and testing of Java code

- JBuilder MobileSet 3 – Extends the JBuilder environment to allow support for J2ME libraries.

- Nokia Developer Suite 1.1 for J2ME – Includes J2ME libraries, documentation and an emulator for Nokia phones.

- Sun Microsystems Wireless Toolkit 1.04 - Allows documentation compilation, packaging and deployment for J2ME development.

- PalmOS Emulator – Allows emulation of applications on PalmOS devices, such as the PalmV. The author installed the PalmOS 3.5 ROM image and the KVM.

12.7. Configuration and profile

Various different mobile devices are known to expose a great variety of incompatible capabilities and features. This makes it difficult to develop Java-style applications that will run on all of them. To alleviate this problem, the J2ME platform groups devices into *configurations* and *profiles*. Applications are therefore developed for a specific group of devices. The device *configuration* is a broad classification, containing the more detailed device *profile* classifications.

For the purpose of implementation the author has selected the following J2ME configuration and profile:

- Connected Limited Device Configuration (CLDC) - These are devices with very limited memory and processing capabilities. However, these devices do have some form of network connection capable of

participating in a TCP/IP network. This is the second least capable group of devices. The only group that is more limited is the Limited Device Configuration (LDC), which contains devices that do not even have networking capabilities. LDC is clearly not an option for peer-to-peer networks.

- Mobile Information Device Profile (MIDP) – This is the least capable profile in the CLDC class of devices. Devices such as cell phones would typically fit this profile.

12.8. Limitations of the CLDC / MIDP

CLDC/MIDP devices typically have the following critical constraints:

- *Application size* - Current cell phones have a total limit of about 123K for storing all MIDlets. Some phones currently limit each MIDlet to be no more than 50K.

- *Persistent storage* on cell phones can be as little as 8K which is shared by all the MIDlets.

- *Runtime heap* is of the order of 32K - 64K

- *Bandwidth* can be very limited and latency is high with connections such as CSD (Circuit Switched Data) and GPRS (General Packet Radio Service). These connections are also not very stable and can be dropped unexpectedly as reception and network load conditions vary.

- *CPU performance* is very limited - around 20MHz.

- *Limited libraries* which MIDP-1.0 has:

    - *No built-in XML* parser. An XML parser would be useful for communicating via standard protocols such as SOAP. XML

parsers designed specifically for MIDP are available, such as [Kxml], but they add significantly to the footprint of the application.

- *Limited Networking* Only outgoing HTTP is required to be implemented on MIDP 1.0 devices. Socket and datagram connections are optional. MIDP 1.0 devices can therefore only act as clients (initiating requests) and not as servers (listening for requests).

12.9. Hardware

The author tested the mobile P2P implementation on the following devices:

- Nokia 6310i cell phone – This is a typical J2ME capable cell phone with a monochrome screen.

- Nokia 6610 cell phone – This is a J2ME capable phone with a full color display.

- 3COM Palm V – This is a typical entry-level Personal Digital Assistant (PDA) with very limited capabilities. This device runs the PalmOS operating system and does not have a J2ME virtual machine installed by default. The author installed the KVM for PalmOS, provided by Sun Microsystems. This device does not have wireless network connectivity, so the author had to connect it to the network via its cradle. Other devices, such as the Palm VII, are equipped with wireless networking.

12.10. Testing, debugging and emulators

Testing and debugging applications for mobile devices can be troublesome. The following difficulties were encountered:

- The application is developed on a PC and then needs to be compiled and installed onto the mobile device. The mobile device has very different capabilities than the PC on which it was developed.

- Compiling and installing the application on the mobile device is a cumbersome process, especially if one just wants to test small code modifications quickly. To make matters even worse – each mobile device has its own installation and packaging procedure. In some cases (non-Java platforms, such as PocketPC) one even has to compile the application separately for each supporting target device.

- In many instances it is impractical to obtain all the various target mobile devices in order to test the application on each one of them. However, testing applications on all the supporting devices is vital for quality assurance due to the variety of operating systems and capabilities of these devices. Mobile devices are expensive and some of them might not be readily available.

- How does one debug applications in the conventional fashion (with break points, watches, etc.) when the application cannot even run on the development machine?

Clearly there must be a better way. The solution is to make use of emulators. An emulator allows one to execute the mobile application code on the development PC. The emulator is capable of closely resembling the characteristics of the target mobile device. Luckily emulators exist for most mobile devices and they are designed with software developers in mind. Most emulators include debugging functionality, which allows the developer to hook up with the running code (on the device) from an IDE (on the PC) and to debug the code in the conventional fashion.

It is, however, still advisable to test the final product on the real device before releasing it. Emulators are not perfect. Sometimes the emulators themselves contain bugs. The application might also perform better in the emulator than on the real device.

Usability should be assessed on the target device itself, because usability depends on the input/output characteristics of the device. The actual mobile device might use handwriting recognition instead of a keyboard, a pen and touchpad instead of a mouse and a 3" monochrome screen instead of a full-color 15" CRT screen.

12.11. Source code

This section describes the functions of the various classes and utility files that entail the phonebook application, and includes the full source code listing. Note that some comments were removed to reduce the size of the code listing. The source code is also available electronically on request.

12.11.1. Phonebook.jad

```
MIDlet-Name: P2P
MIDlet-Version: 0.0.1
MIDlet-Vendor: MyCompany
MIDlet-Jar-URL: Phonebook.jar
MIDlet-Jar-Size: 7075
MIDlet-1: Phonebook, , p2p.phonebook.PhonebookMIDlet
App-LocalNumber = "27823219751";
App-RelayURL = "http://localhost/p2p/relay.asp";
App-ListenInterval = 10000;
App-TTL = 5;
```

The Java Application Descriptor (JAD) file is required by each J2ME application. The classes are compiled and then packed into a Java Archive (JAR) file. The JAD file describes the location of the JAR file as well as some other information relevant to the application. It can also include user defined settings specific to a distribution or installation of the application. In this case the author includes the following settings:

- App-LocalNumber – The MSISDN (cell phone) number for the device on which the application is installed.

- App-RelayURL – The URL of the web application that serves as a message relay. The relay is required due to communication restrictions in current MIDP devices.

- App-ListenInterval – The interval (in milliseconds) at which the MIDlet should check for new messages with the relay.

- App-TTL – The time to live period for messages routed via the peer-to-peer network. The TTL value determines how many times a message can be forwarded through the network before it gets discarded. This prevents endless loops and puts a limit on the network reach from each node.

12.11.2. SearchResultsList.java

```
package p2p.phonebook;

import javax.microedition.lcdui.*;
import java.util.*;

public class SearchResultsList extends List implements CommandListener {

  private static Vector results = null;
  private static SearchResultsList instance = null;

  /**Construct the displayable*/
  public SearchResultsList() {
    super("Search Results", List.IMPLICIT);
    instance = this;
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  /**Component initialization*/
  private void jbInit() throws Exception {
    // set up this Displayable to listen to command events
    setCommandListener(this);
    // add the Exit command
    addCommand(new Command("Save", Command.SCREEN, 1));
    addCommand(new Command("Clear", Command.SCREEN, 1));
    addCommand(new Command("Close", Command.BACK, 1));
    updateList();
  }
```

```
  public void updateList() {
    //first clear the current list
    for(int c=0;c<=this.size();c++) {
      this.delete(c);
    }

    //now add all the entries
    for(int c=0;c<=results.size();c++) {
      String[] entry = (String[])results.elementAt(c);
      String entryLine = entry[0] + ": " + entry[1];
      append(entryLine, null);
    }
  }

  public void clearList() {
    results.removeAllElements();
    updateList();
  }

  public static void addResult(String name, String number) {
    //check if entry already exist
    boolean exist = false;
    for(int c=0;c<=results.size();c++) {
      String[] entry = (String[])results.elementAt(c);
      if((entry[0] == name) && (entry[1] == number)) exist = true;
    }

    //add entry
    if(!exist) {
      String[] entry = {name, number};
      results.addElement(entry);
    }

    //update displayed list
    if(instance != null) {
      instance.updateList();
    }
  }

  public static SearchResultsList getInstance() {
    return(instance);
  }

  /**Handle command events*/
  public void commandAction(Command command, Displayable displayable) {
    Display disp = Display.getDisplay(PhonebookMIDlet.getInstance());
    String cmdLabel = command.getLabel();
    if(cmdLabel == "Save") {
      String name =
((String[])results.elementAt(this.getSelectedIndex()))[0];
      String number =
((String[])results.elementAt(this.getSelectedIndex()))[1];
      ContentStore.saveEntry(name, number);
      Alert alert = new Alert("Saved", name + " has been saved.", null,
AlertType.INFO);
      disp.setCurrent(alert, this);
    }
    else if(cmdLabel == "Clear") {
      clearList();
    }
    else if(cmdLabel == "Close") {
      disp.setCurrent(PBEntriesForm.getInstance());
    }
  }

}
```

This class represents a visual display of the search results. The list displays the names and numbers that were found as a result of a peer-to-peer search. The results are updated as they come in, meaning that the list can grow over time as more results are discovered.

12.11.3. SearchForm.java

```
package p2p.phonebook;

import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
import java.io.*;

public class SearchForm extends Form implements CommandListener {
  private TextField txtName;
  /**Construct the displayable*/
  public SearchForm() {
    super("Search");
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  /**Component initialization*/
  private void jbInit() throws Exception {
    // set up this Displayable to listen to command events
    txtName = new TextField("", "", 15,TextField.ANY);
    txtName.setLabel("Name");
    setCommandListener(this);

    // add commands
    addCommand(new Command("OK", Command.OK, 1));
    addCommand(new Command("Cancel", Command.CANCEL, 1));

    this.append(txtName);
  }

  /**Handle command events*/
  public void commandAction(Command command, Displayable displayable) {
    String cmdLabel = command.getLabel();
    if(cmdLabel == "OK") {
      search(PhonebookMIDlet.TTL, PhonebookMIDlet.LOCAL_NUMBER,
PhonebookMIDlet.getNextMsgID(), txtName.getString());
      Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
      dpl.setCurrent(SearchResultsList.getInstance());
    }
    else if(cmdLabel == "Cancel") {
      Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
      PBEntriesForm entries = PBEntriesForm.getInstance();
      dpl.setCurrent(entries);
    }
  }


  public static void search(int ttl, String from, long msgId, String name)
{

    //first search the local store
    ContentItem[] entries = ContentStore.getEntries();
```

```
    for(int c=0;c<=entries.length;c++) {
      if(entries[c].NAME == name) {
        //send response
        String command = "SEARCH_RESPONSE|" + ttl + "|" + from + "|" +
msgId + "|" + name + "|" + entries[c].NUMBER;
        MsgListener.send(from, command);
      }
    }

    //now forward request to all attached nodes
    ContentItem[] nodes = ContentStore.getEntries();
    for(int c=0;c<=nodes.length;c++) {
      String command = "SEARCH_REQUEST|" + ttl + "|" + from + "|" + msgId
+ "|" + name;
      MsgListener.send(nodes[c].NUMBER, command);
    }

  }

}
```

This class provides an input form for specifying search criteria. In this case the user only needs to supply the name of the user whose number he wants to search for. Clicking on the search button will kick off the peer-to-peer search process.

12.11.4. RouteTable.java

```
package p2p.phonebook;

import java.util.*;

public class RouteTable {

  private static Vector items = null;

  public static void add(RouteItem item) {
    items.addElement(item);
  }

  public static boolean alreadyReceived(String from, long id) {
    boolean match = false;
    for(int c = 0;c<=items.size();c++) {
      RouteItem item = (RouteItem)items.elementAt(c);
      if((item.FROM == from) && (item.MSGID == id)) {
        match = true;
      }
    }
    return(match);
  }

}
```

This class keeps a list of messages received and forwarded by this node. The list is needed to prevent the same message from being processed twice. The list contains a collection of RouteItem objects.

12.11.5. RouteItem.java

```
package p2p.phonebook;

public class RouteItem {

  public static String FROM = null;
  public static long MSGID = 0;
  //public static String TO = null;

  public RouteItem() {
    //provide empty constructor
  }

  public RouteItem(String from, long msgid) {
    FROM = from;
    MSGID = msgid;
  }

}
```

This class defines the routing entries that are stored in the RoutingTable object. Each entry contains the following fields:

- From – the MSISDN (cell phone) number of the node from which the message was received.
- MsgID – A number which uniquely identifies the message sent by the originating node. Each message sent from an originating node is given a sequential ID unique to that node. Therefore the combination of *MsgID* and *From* address is always unique.

12.11.6. PBNewEntryForm.java

```
package p2p.phonebook;

import java.io.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;

public class PBNewEntryForm extends Form implements CommandListener {

  public PBNewEntryForm() {
    super("New entry");
    TextField fldName = new TextField("Name", null, 30, TextField.ANY);
    TextField fldNumber = new TextField("Number", null, 30,
TextField.PHONENUMBER);
    append(fldName);
    append(fldNumber);

    Command cmdAdd = new Command("OK", Command.OK, 1);
    Command cmdCancel = new Command("Cancel", Command.CANCEL, 1);
    addCommand(cmdAdd);
```

```
    addCommand(cmdCancel);
    setCommandListener(this);
  }


  public void commandAction(Command cmd, Displayable disp) {
    String cmdLabel = cmd.getLabel();
    if(cmdLabel == "OK") {
        TextField nameField = (TextField)this.get(0);
        TextField numberField = (TextField)this.get(1);
        String name = nameField.getString();
        String number = numberField.getString();
        ContentStore.saveEntry(name,number);
        Alert alert = new Alert("Saved", name + ": " + number,null,null);
        alert.setTimeout(Alert.FOREVER);
        Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
        dpl.setCurrent(alert, disp);
    }
    //alwyas go back to EntriesList screen
    PBEntriesForm entries = PBEntriesForm.getInstance();
    entries = new PBEntriesForm();  //this will clear the list
    //entries.loadEntries();
    Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
    dpl.setCurrent(entries);

  }

}
```

This class represents an input form for the user to allow him to add a new entry to the local phonebook store.

12.11.7. PBEntryDetailForm.java

```
package p2p.phonebook;

import javax.microedition.lcdui.*;
import javax.microedition.rms.*;
import java.io.*;

public class PBEntryDetailForm extends Form implements CommandListener,
RecordFilter {

  String filterName = null;

  /**Construct the displayable*/
  public PBEntryDetailForm(String name) {
    super(name);
    filterName = name;
    addCommand(new Command("Back", Command.BACK, 1));
    addCommand(new Command("Delete", Command.SCREEN, 1));
    setCommandListener(this);

    showDetail();
  }

  public boolean matches(byte[] candidate) {
    if(filterName == null) {
      return(true);
    }
    else {
      String name = null;
      String number = null;
      ByteArrayInputStream bais = new ByteArrayInputStream(candidate);
      DataInputStream dis = new DataInputStream(bais);
```

```
        try {
          name = dis.readUTF();
          number = dis.readUTF();
        }
        catch(Exception e) {
          e.printStackTrace();
        }
        //String sTest = name + ": " + number;
        return(this.filterName.equals(name));
      }

    }

    /**Handle command events*/
    public void commandAction(Command command, Displayable displayable) {
      String cmdLabel = command.getLabel();
      if(cmdLabel == "Back") {
        Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
        PBEntriesForm entries = PBEntriesForm.getInstance();
        dpl.setCurrent(entries);
      }
      else if(cmdLabel == "Delete") {
        deleteEntry();
        Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
        PBEntriesForm entries = PBEntriesForm.getInstance();
        entries = new PBEntriesForm();
        dpl.setCurrent(entries);
      }
    }


    private void deleteEntry() {
      try {
        RecordStore rs = RecordStore.openRecordStore("phonebook", false);
        RecordEnumeration re = rs.enumerateRecords(this,null,false);
        rs.deleteRecord(re.nextRecordId());
      }
      catch(Exception e) {
        e.printStackTrace();
      }
    }


    private void showDetail() {
      try {
        RecordStore rs = RecordStore.openRecordStore("phonebook", false);
        RecordEnumeration re = rs.enumerateRecords(this,null,false);

        ByteArrayInputStream bais = new
ByteArrayInputStream(re.nextRecord());
        DataInputStream dis = new DataInputStream(bais);

        String name = dis.readUTF();
        String number = dis.readUTF();

        StringItem nameItem = new StringItem("Name: ", name);
        StringItem numberItem = new StringItem("Number: ", number);

        this.append(nameItem);
        this.append(numberItem);
      }
      catch(Exception e) {
        e.printStackTrace();
      }
    }

}
```

This class provides a visual representation for displaying the details of a phonebook entry.

12.11.8. PBEntriesForm.java

```
package p2p.phonebook;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class PBEntriesForm extends List implements CommandListener {

  private static PBEntriesForm instance = null;

  public PBEntriesForm() {

    super("Entries", List.EXCLUSIVE);
    instance = this;

    ContentItem[] entries = ContentStore.getEntries();
    for(int c=0;c<=entries.length;c++) {
      String entry = entries[c].NAME + ": " + entries[c].NUMBER;
      append(entry, null);
    }

    addCommand(new Command("Detail", Command.SCREEN, 1));
    addCommand(new Command("New", Command.SCREEN, 1));
    addCommand(new Command("Search", Command.SCREEN, 1));
    addCommand(new Command("Exit", Command.EXIT, 1));
    setCommandListener(this);
  }

  public void commandAction(Command cmd, Displayable disp) {

    Display dpl = Display.getDisplay(PhonebookMIDlet.getInstance());
    String cmdLabel = cmd.getLabel();

    if(cmdLabel == "New") {
        PBNewEntryForm form = new PBNewEntryForm();
        dpl.setCurrent(form);
    }
    else if(cmdLabel == "Detail") {
      if(getSelectedIndex() == -1) {
        Alert alert = new Alert("Error", "No entry selected!", null,
AlertType.ERROR);
        dpl.setCurrent(alert);
      }
      else {
        String strSelected = getString(getSelectedIndex());
        PBEntryDetailForm detailForm = new PBEntryDetailForm(strSelected);
        dpl.setCurrent(detailForm);
      }
    }
    else if(cmdLabel == "Search") {
      SearchForm searchForm = new SearchForm();
      dpl.setCurrent(searchForm);
    }
    else if(cmdLabel == "Exit") {
      PhonebookMIDlet.quitApp();
    }
  }

  public static PBEntriesForm getInstance() {
    return(instance);
  }
```

```
}
```

This form displays a list of all the phonebook entries in the local store.

12.11.9. MsgListener.java

```
package p2p.phonebook;

import javax.microedition.io.*;
import java.io.*;

public class MsgListener implements Runnable {


  public static boolean running = true;
  private int count = 0;

  public MsgListener() {
  }

  public void run() {
    Thread instance = PhonebookMIDlet.getMsgListener();

    while(running) {
      receiveMsg();
      try {
        instance.sleep(PhonebookMIDlet.LISTEN_INTERVAL);
      }
      catch(InterruptedException e) {
        //do nothing
      }
    }
  }

  private void receiveMsg() {
    HttpConnection con = null;
    OutputStream os = null;
    InputStream is = null;
    int ch;
    int pos;
    String line = null;
    String url = null;
    count += 1;
    System.out.println("Receive attempt " + count);
    url = PhonebookMIDlet.RELAY_URL + "?command=receive&number=" +
PhonebookMIDlet.LOCAL_NUMBER;
    try {
      StringBuffer buffer = new StringBuffer();
      con = (HttpConnection)Connector.open(url);
      os = con.openOutputStream();

      os.flush();

      is = con.openDataInputStream();
      while((ch = is.read()) != -1) buffer.append((char)(ch));
      String resultString = buffer.toString();

      while((pos = resultString.indexOf("\n")) != -1) {
        line = resultString.substring(0, pos);
        if(line.trim() != "") process(line);
        resultString = resultString.substring(pos+1,
resultString.length());
      }

      if(is != null) is.close();
      if(os != null) os.close();
```

```
      if(con != null) con.close();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    finally {
    }
    System.out.println();
  }

  private void process(String message) {

    MessageItem msg = parseMessage(message);

    //decrement TTL. Discard when we reach 0
    msg.TTL -= 1;
    if(msg.TTL>0) {

      if(msg.COMMAND == "SEARCH_REQUEST") {
        SearchForm.search(msg.TTL, msg.FROM, msg.MSGID, msg.PARAMS);
        RouteTable.add(new RouteItem(msg.FROM,msg.MSGID));
      }
      else if(msg.COMMAND == "SEARCH_RESPONSE") {

      }

    }

  }



  private MessageItem parseMessage(String message) {

    MessageItem result = new MessageItem();

    int pos = 0;
    pos = message.indexOf("|");
    result.COMMAND = message.substring(0, pos);
    message = message.substring(pos +1);

    pos = message.indexOf("|");
    result.TTL = Integer.parseInt(message.substring(0, pos));
    message = message.substring(pos +1);

    pos = message.indexOf("|");
    result.FROM = message.substring(0, pos);
    message = message.substring(pos +1);

    pos = message.indexOf("|");
    result.MSGID = Long.parseLong(message.substring(0, pos));
    message = message.substring(pos +1);

    pos = message.indexOf("|");
    result.PARAMS = message.substring(0, pos);
    message = message.substring(pos +1);

    return(result);
  }


  public static void send(String to, String message) {

    String url = PhonebookMIDlet.RELAY_URL + "?command=send&number=" + to
+ "&message=" + message;
    try {
      HttpConnection con = (HttpConnection)Connector.open(url);
      OutputStream os = con.openOutputStream();
      os.flush();
```

```
      os.close();
      con.close();
    }
    catch(Exception e) {
      e.printStackTrace();
    }

  }
}
```

This class handles all communications between nodes. The class is a thread that implements a listener that listens for incoming messages. The messages are then parsed and appropriate action is taken.

12.11.10. MessageItem.java

```
package p2p.phonebook;

public class MessageItem {
  public String COMMAND = null;
  public int TTL = 0;
  public String FROM = null;
  public long MSGID = 0;
  public String PARAMS = null;
}
```

Instances of this class represent the messages that are routed through the peer-to-peer network.

12.10.11. ContentStore.java

```
package p2p.phonebook;

import java.io.*;
import javax.microedition.rms.*;

public class ContentStore {

  public ContentStore() {
  }


  public static ContentItem[] getEntries() {

    ContentItem[] entries = null;

    try {
      RecordStore rs = RecordStore.openRecordStore("phonebook", true);
      RecordEnumeration re = rs.enumerateRecords(null, null, false);

      int iRows = re.numRecords();
      entries = new ContentItem[iRows];
      int count = 0;

      while(re.hasNextElement()) {
        ByteArrayInputStream bin = new
ByteArrayInputStream(re.nextRecord());
```

```
            DataInputStream din = new DataInputStream(bin);
            String name = din.readUTF();
            String number = din.readUTF();
            entries[count] = new ContentItem();
            entries[count].NAME = name;
            entries[count].NUMBER = number;
            count++;
          }
        }
        catch(Exception e) {
          e.printStackTrace();
        }
        return(entries);
      }


      public static void saveEntry(String name, String number) {
        try {
          RecordStore rs = RecordStore.openRecordStore("phonebook", true);
          ByteArrayOutputStream bout = new ByteArrayOutputStream();
          DataOutputStream dout = new DataOutputStream(bout);
          dout.writeUTF(name);
          dout.writeUTF(number);
          byte[] b = bout.toByteArray();
          rs.addRecord(b, 0, b.length);
        }
        catch(Exception e) {
          e.printStackTrace();
        }
      }

      public static String searchForNumber(String name) {
        String number = null;
        ContentItem[] entries = getEntries();
        for(int c=0;c<=entries.length;c++) {
          if(entries[c].NAME == name) {
            number = entries[c].NUMBER;
          }
        }
        return(number);
      }

}
```

This class handles all content storage functionality and represents the local store. Content storage in MIDP devices is handled through the special Record Management System (RMS). The RMS is the only way to provide persistent storage on MIDP devices. The RMS is something between a lightweight DBMS and a raw binary file and is a unique concept to MIDP.

12.11.12. ContentItem.java

```
package p2p.phonebook;

public class ContentItem {

  public String NAME = null;
  public String NUMBER = null;
```

```
  public ContentItem() {
  }

}
```

This class represents a single phonebook entry that is stored in the RecordStore.

12.11.13. PhonebookMIDlet.java

```
package p2p.phonebook;

import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class PhonebookMIDlet extends MIDlet {

  //default values for system settings. Overwrite this in JAD file
  public static String LOCAL_NUMBER = "27823219751";
  public static String RELAY_URL = "http://localhost/p2p/relay.asp";
  public static long LISTEN_INTERVAL = 10000;
  public static int TTL = 5;

  private static PhonebookMIDlet instance = null;
  private static Thread msgListener = null;
  private static long msgID = 0;

  public PhonebookMIDlet() {
    instance = this;
  }

  protected void startApp() {
    getSettings();
    msgListener = new Thread(new MsgListener());
    msgListener.start();
    //initialize search results list
    new SearchResultsList();
    PBEntriesForm entries = new PBEntriesForm();
    //Ticker ticker = new Ticker("Select one of the phonebook entries
below. Select the appropriate command button.");
    //entries.setTicker(ticker);
    Display.getDisplay(this).setCurrent(entries);
  }

  private void getSettings() {
    String temp = null;
    temp = getAppProperty("App-LocalNumber");
    if(temp!=null) {
      LOCAL_NUMBER = temp;
    }

    temp = getAppProperty("App-RelayURL");
    if(temp!=null) {
      RELAY_URL = temp;
    }

    temp = getAppProperty("App-ListenInterval");
    if(temp!=null) {
      LISTEN_INTERVAL = Long.parseLong(temp);
    }

    temp = getAppProperty("App-TTL");
    if(temp!=null) {
      TTL = Integer.parseInt(temp);
    }
```

```
  }

  public static long getNextMsgID() {
    msgID +=1;
    return(msgID);
  }

  protected void pauseApp() {

  }

  protected void destroyApp(boolean unconditional) {

  }

  protected static MIDlet getInstance() {
    return(instance);
  }

  protected static Thread getMsgListener() {
    return(msgListener);
  }

  /** Quit the MIDlet */
  public static void quitApp() {
    MsgListener.running = false;
    msgListener.yield();
    msgListener = null;
    instance.destroyApp(true);
    instance.notifyDestroyed();
    instance = null;
  }

}
```

This is the main class from where program execution takes place. This class extends the MIDlet class, which is the parent class of all J2ME applications. This class reads the system settings from the JAD file and starts up the initial screen.

12.11.14. Relay.asp

```
<% option explicit %>

<%
          dim sCommand, sMessage, sNumber
          dim arr
          dim c

          sCommand = Request("command")
          sMessage = Request("message")
          sNumber = Request("number")


          select case sCommand
                    case "send"
                              arr = Application(sNumber)
                              if isArray(arr) then
                                        c = (ubound(arr) + 1)
                                        redim preserve arr(c)
```

```
                                        arr(c) = sMessage
                                else
                                        c = 0
                                        redim arr(c)
                                        arr(c) = sMessage
                                end if
                                Application(sNumber) = arr
                        case "receive"
                                arr = Application(sNumber)
                                if isArray(arr) then
                                        for c = lbound(arr)to ubound(arr)
                                                sMessage = arr(c)
                                                Response.Write sMessage &
vbcrlf
                                        next
                                        Application(sNumber) = Empty
                                end if
                        case else
                                Response.Write "Invalid command!"
                end select
%>
```

This is the ASP page which is used for relaying messages between the various peers. The relaying mechanism is used to simulate peer-to-peer UDP connections. This is needed because the current J2ME specification only allows for client sockets and not server sockets. True peer-to-peer communication is therefore not possible without polling and relaying mechanisms such as the one that we used. It is however anticipated that server sockets will be available to MIDP devices in the near future, therefore eliminating the need for a relay agent.

12.12. P2P Phonebook in action

This section presents some screen shots to demonstrate the P2P Phonebook application in action. The demonstration shows the same Java application running in the emulators for two different devices: the Motorola i85s cell phone and the 3Com Palm V device. These two devices have different input/output capabilities, such as different screen sizes and a tap screen (Palm) versus soft keys (cell phone) for input. This section demonstrates how the exact same Java application can run on both devices.

12.12.1. Launching the application

Figure 19: Launching the P2PPhonebook

The above figures demonstrate how the P2PPhonebook application is launched. On the Motorola cell phone the application is selected from a list of installed MIDlets. On the Palm device the application is selected by tapping its icon from the main menu.

12.12.2. List of entries

Figure 20: Rendering phonebook entries

This form displays a list of all current entries in the phonebook. On the Motorola device the user selects an entry by scrolling up and down with the cursor keys. With the Palm device the user selects an entry by simply tapping on the entry. There are four commands that can be performed on this screen: *Detail*, *New*, *Search* and *Exit*. The Palm device provides three of the options as command buttons. The *Exit* command is provided as part of the application dropdown menu, thereby conforming to the look-and-feel of other Palm OS applications. The Motorola device can only display two command options at a time. If more than two options are available, the rest of the options are presented in a menu which is activated by pressing the menu button (shown in the next section).

12.12.3. Menu options

Figure 21: Rendering additional commands

As mentioned in the previous section, the Motorola devices (like most other cell phones) can only display two command options at a time. In the "list of entries" form it only displays the *Exit* and *Detail* commands. The other commands are accessed by pressing the menu button. The above figure shows how the *New* and *Search* commands are displayed in a separate menu.

12.12.4. Viewing entry details

Figure 22: Rendering entry details

A detailed screen is displayed when the user selects the *Detail* command. Note that the Palm device renders the *name* and *number* labels in bold face, while the Motorola device does not have the capability to display various font styles. The Palm device also right-align the labels and left-align the values, while the screen size of the Motorola device does not have enough space for alignment. In our simple application only the name and number are displayed, but a real-world application might include more detail such as address and birth date.

12.12.5. Adding a new entry

Figure 23: Adding new entries

This form is displayed after the user selected the *New* command. The form contains input fields that allow the user to enter details for the new phonebook entry. The Motorola user is limited to the numeric keypad for entering information. Entering names is therefore a tedious task and requires lots of keystrokes. The Palm user can use the graffiti pane for handwriting recognition or use the popup keyboard to tap the letters on the screen. Both accomplish the same task, but in very different ways.

12.12.6. Searching

Figure 24: Initiating distributed search

The search screen allows the user to enter the name for which he wants to search. The distributed search process is starts when the user selected the *OK* command. Search request messages is then send to all nodes listed in the phonebook. Each node in turn investigates its local store for query hits and forwards the request to all the nodes it knows about. Routing is controlled by the routing table and the TTL of each message.

12.12.7. Search results

Figure 25: Waiting for search results

The search results screen shows the results of the peer-to-peer search as it occurs. The longer this screen is open, the more results might be discovered and displayed. Once an entry is discovered, the user can select it and activate the *Save* command in order to add the newly discovered entry to his local phonebook. This way the content gets replicated onto multiple nodes and provides better availability.

Chapter 13

13. CONCLUSIONS

This chapter summarizes the lessons learned from our research.

13.1. Historical impact

New applications can only be studied in context of its historical origin. Peer-to-peer applications are a new breed of Internet applications, but they are constraint by the Internet infrastructure which was designed decades ago. This has a significant impact on the way these new applications are designed and implemented.

13.2. Application model

The peer-to-peer application model is very different from the conventional client/server model. With a peer-to-peer model each host performs both the tasks of a conventional client and server host. The peer-to-peer model also introduces an application-level routing infrastructure.

13.3. Classification

Peer-to-peer applications can be grouped according to the functionality that they provide and according to the computing model (degree of centralization). A combination of these two criteria is sufficient to provide a classification system. The classification system can be applied to describe and study P2P applications with similar attributes and behavior.

13.4. Mechanisms

The author has identified five main mechanisms that can be used to describe a distributed content-sharing P2P application. These mechanisms are node

discovery, content discovery, content retrieval, content publishing and content storage. The author performed a case study of the Napster, Gnutella and Freenet systems and demonstrated that these mechanisms can be applied to describe the inner workings of P2P systems.

13.5. Napster, Gnutella and Freenet

We presented a case study of the Napster, Gnutella and Freenet P2P systems. All three of these systems have the same goal: to provide distributed content-sharing abilities, but each of them achieves this goal very differently.

13.6. Research efforts

We indicated that various research efforts and projects are underway to try to further study and improve the peer-to-peer application model. Projects such as JXTA aim at providing a generic P2P framework and protocols.

13.7. Challenges

We have identified many challenges that need to be overcome if peer-to-peer applications are to become widely adopted. These challenges are to minimize the distance between connected nodes, invent better mechanisms for initial node discovery, establish sustainable participation, reduce the bandwidth demand, alleviate firewall problems, provide service guarantees, address political and social concerns and provide forward compatibility. We provided some hints on how some of these issues might be resolved. These challenges also indicate direction for future research efforts.

13.8. Mobile P2P

It might be possible to implement true peer-to-peer applications on mobile devices with very limited capabilities, such as cell phones. We implemented and simulated a P2P phonebook application for mobile devices. We realized that there is still a lot of challenges to implement true mobile P2P with current technologies presented some ideas on how this might be overcome in the near future.

Mobile P2P eliminates the necessity for server and desktop infrastructure all together and allows the end-user to enjoy the benefits of peer-to-peer applications from an inexpensive mobile device and from a greater variety of locations.

13.9. Future work

The section indicates some direction for future research work:

- Perform tests on the various categories of peer-to-peer networks to compare attributes such as performance, bandwidth consumption, scalability and security.

- Devise new peer-to-peer protocols and mechanisms that would solve issues such as initial node discovery and high bandwidth consumption.

- Keep track of new peer-to-peer initiatives as they evolve. Determine the unique characteristics of each network and critically evaluate them.

- More research is required to better understand the behavior and design constraints of peer-to-peer on mobile devices.

BIBLIOGRAPHY

Articles and books

1. Aberer K, Punceva M, Hauswirth M, Schmidt R, "Improving Data Access in P2P Systems", IEEE Internet Computing, **6** (1) 58-67, Jan/Feb 2002
2. Agre P E, "P2P and the promise of internet equality", Communications of the ACM, **46** (2) 39-42, February 2003
3. Balakrishnan H, Kaashoek M F, Karger D, Morris R, Stoica I, "Looking up data in P2P systems", Communications of the ACM, **46** (2) 43-48, February 2003
4. Barkai D, "Technologies for Sharing and Collaborating on the Net", Peer-to-Peer Computing 2001, 13-28
5. Clark D, "Face-to-Face with Peer-to-Peer Networking", IEEE Computer, **34** (1) 18-21, Jan 2001
6. Clarke I, Miller S G, Hong T W, Sandberg O, Wiley B, "Protecting Free Expression Online with Freenet", IEEE Internet Computing, **6** (1) 40-49, Jan/Feb 2002
7. Connolly D, "On the Architecture of the Web: Let a Thousand Flowers Bloom (Interview).", IEEE Internet Computing, **2** (2) 22-31, March/April 1998
8. Danzfuss F, Bishop J, "Lessons Learned from Building a Web-Based Spatial Data Discovery Facility", FTDCS 1999, 267-272
9. Delaney B, "The Power of P2P", IEEE MultiMedia, **8** (2) 100-103, April 2001
10. Edwards J, "Next-Generation Viruses Present New Challenges", IEEE Computer, **34** (5) 16-18, May 2001
11. Faybishenko Y, Kan G, "Introduction to Gnougat", Peer-to-Peer Computing 2001, 4-12
12. Foster I, Kesselman C, **The GRID: Blueprint for a new computing infrastructure**, Morgan Kaufmann Publishers, San Francisco, 1998
13. Fox G, "Peer-to-Peer Networks", IEEE Computing in Science & Engineering, **3** (3) 75-77, May/June 2001
14. Gong L, "JXTA: A Network Programming Environment", IEEE Internet Computing, **5** (3) 88-95, May/June 2001
15. Graham L, "The Legal Tortoise and the Technology Hare", IEEE Software, **17** (5) 18-19, Sept/Oct 2000

16. Hayes B, "Collective Wisdom", American Scientist, **86** (2) 118-122, March/April 1998

17. Khare R, "I Want My FTP: Bits on Demand", IEEE Internet Computing, **2** (4) 88-91, July/August 1998a

18. Khare R, "Telnet: The Mother of All (Application) Protocols", IEEE Internet Computing, **2** (3) 88-91, May/June 1998b

19. Khare R, "Building the Perfect Beast: Dreams of a Grand Unified Protocol", IEEE Internet Computing, **3** (2) 89-93, March/April 1999a

20. Khare R, "Who Killed Gopher? An Extensible Murder Mystery", IEEE Internet Computing, **3** (1) 81-84, Jan/Feb 1999b

21. Kubiatowicz J, "Extracting guarantees from chaos", Communications of the ACM, **46** (2) 33-38, February 2003

22. Lawton G, "Distributed Net Applications Create Virtual Supercomputers", IEEE Computer, **33** (6) 16-20, June 2000

23. Lee J, "An end-user perspective on file-sharing systems", Communications of the ACM, **46** (2) 49-53, February 2003

24. Lienhart R, Holliman M J, Chen Y, Kozintsev I, Yeung M M: "Improving Media Services on P2P Networks", IEEE Internet Computing, **6** (1) 73-77, Jan/Feb 2002

25. Macedonia M R, "Distributed File Sharing: Barbarians at the Gates?", IEEE Computer, **33** (8) 99-101, August 2000

26. Parameswaran M, Susarla A, Whinston A B, "P2P Networking: An Information-Sharing Alternative", IEEE Computer, **34** (7) 31-38, July 2001

27. Rhea S, Wells C, Eaton P R, Geels D, Zhao B Y, Weatherspoon H, Kubiatowicz J, "Maintenance-Free Global Data Storage", IEEE Internet Computing, **5** (5) 40-49, Sept/Oct 2001

28. Ripeanu M, Iamnitchi A, Foster I T, "Mapping the Gnutella Network", IEEE Internet Computing, **6** (1) 50-57, Jan/Feb 2002

29. Schneier B, **Secrets & Lies. Digital Security in a Networked World**, Wiley Computer Publishing, New York, 2000

30. Stern R, "Napster: A Walking Copyright Infringement?", IEEE Micro, **20** (6) 4-5, Nov/Dec 2000

31. Tanenbaum A, **Computer Networks**, Prentice Hall, New Jersey, 1994

32. Waterhouse S R, Doolin D M, Kan G, Faybishenko Y, "Distributed Search in P2P Networks", IEEE Internet Computing, **6** (1) 68-72, Jan/Feb 2002

Web references

33. Bearshare: "Bearshare Gnutella client", http://www.bearshare.com, 2002

34. Entropia: "Entropia", http://www.entropia.com, 2002

35. FreenetProject: "The Freenet Project", www.freenetproject.org

36. Gnutella: "The Gnutella Protocol Specification v0.4", http://www.clip2.com

37. Google: "Google", http://www.google.com, 2003

38. GopherManifest: Karger B, "The Bring Back Gopher Campaign", Nov 2000, http://www.scn.org/~bkarger/gopher-manifesto

39. Hobbes: Zakon R, "Hobbes' Internet Timeline v5.4", http://www.zakon.org/robert/internet/timeline, 2001

40. InetDef: "Internet definition", http://www.faqs.org/docs/jargon/I/Internet.html, 2003

41. JTella: "JTell, Java API for Gnutella", http://jtella.sourceforge.net, 2002

42. Kxml: "XML parser for the KVM", http://www.kxml.org, 2002

43. LimeWire: "LimeWire Gnutella client", http://www.limewire.com, 2002

44. MediaDefender: "The Media Defender", www.mediadefender.com

45. NetTimeline: "Life of the Internet: Net Timeline", http://www.pbs.org/internet/timeline.

46. OldNew, "The Usenet OldNews Archive", gopher://gopher.quux.org/1/Archives/usenet-a-news, 2002

47. Seti: "Search for Extraterrestrial Intelligence", http://setiathome.ssl.berkeley.edu, 2001

48. SymanticWeb: Fox, "The Symantic Web", 2001, http://www.w3.org/2001/sw

49. Usenet: "History of Usenet", http://www.vrx.net/usenet/history, 2002

Request for comments (RFCs)

0097. "First Cut at a Proposed Telnet Protocol". J.T. Melvin, R.W.Watson. Feb-15-1971.

0163. "Data transfer protocols". V.G. Cerf. May-19-1971.

0172. "The File Transfer Protocol". A. Bhushan, B. Braden, W. Crowther, E. Harslem, J. Heafner, A. McKenzie, J. Melvin, B. Sundberg, D. Watson, J. White. Jun-23-1971.

0265. "The File Transfer Protocol". A. Bhushan, B. Braden, W. Crowther, E. Harslem, J. Heafner, A. McKenzie, J. Melvin, B. Sundberg, D. Watson, J. White. Nov-17-1971.

0385. "Comments on the File Transfer Protocol". A.K. Bhushan. Aug-18-1972.

0680. "Message Transmission Protocol". T.H. Myer, D.A. Henderson. Apr-30-1975.

0733. "Standard for the format of ARPA network text messages". D.Crocker, J. Vittal, K.T. Pogran, D.A. Henderson. Nov-21-1977.

0751. "Survey of FTP mail and MLFL". P.D. Lebling. Dec-10-1978.

0854. "Telnet Protocol Specification". J. Postel, J.K. Reynolds. May-01-1983.

0855. "Telnet Option Specifications". J. Postel, J.K. Reynolds. May-01-1983.

0856. "Telnet Binary Transmission". J. Postel, J.K. Reynolds. May-01-1983.

0857. "Telnet Echo Option". J. Postel, J.K. Reynolds. May-01-1983.

0858. "Telnet Suppress Go Ahead Option". J. Postel, J.K. Reynolds. May-01-1983.

0859. "Telnet Status Option". J. Postel, J.K. Reynolds. May-01-1983.

0860. "Telnet Timing Mark Option". J. Postel, J.K. Reynolds. May-01-1983.

0861. "Telnet Extended Options: List Option". J. Postel, J.K. Reynolds. May-01-1983.

0959. "File Transfer Protocol". J. Postel, J.K. Reynolds. Oct-01-1985.

1080. "Telnet remote flow control option". C.L. Hedrick. Nov-01-1988.

1350. "The TFTP Protocol (Revision 2)". K. Sollins. July 1992.

1459. "Internet Relay Chat Protocol". J. Oikarinen, D. Reed. May 1993.

1689. "A Status Report on Networked Information Retrieval: Tools and Groups". J. Foster, Ed.. August 1994.

2066. "TELNET CHARSET Option". R. Gellens. January 1997.

2810. "Internet Relay Chat: Architecture". C. Kalt. April 2000.

2811. "Internet Relay Chat: Channel Management". C. Kalt. April 2000.

2812. "Internet Relay Chat: Client Protocol". C. Kalt. April 2000.

2813. "Internet Relay Chat: Server Protocol". C. Kalt. April 2000.