

UMEÅ UNIVERSITY
Department of Computing Science
Master Thesis

**IMPROVING NETWORK
UTILIZATION FOR
PEER-TO-PEER SERVICES**

Marcus Bergner

19th December 2002

Abstract

Peer-to-peer networking is not a new concept, but has been resurrected by services such as Napster and file sharing applications using Gnutella. The network infrastructure of today's networks are based on the assumption that users are simple clients making small requests and receiving, possibly large, replies. This assumption does not hold for many peer-to-peer services and hence the network is often used inefficiently.

This report investigates the reasons behind this waste of resources and looks at various ways to deal with these issues. Focusing mainly on file sharing and the Gnutella protocol addressing and query optimization is investigated, both in theory and practice.

Looking at other peer-to-peer services conclusions regarding peer-to-peer in general can be made. The thesis ends by summarizing the solutions to some of the flaws in peer-to-peer services and provides guidelines on how peer-to-peer services can be constructed to use network resources more efficiently.

Contents

1	Requirements	1
1.1	Summary of requirements	1
1.2	Investigate existing services	1
1.3	Analyzing the Gnutella protocol	2
1.4	Constructing a file sharing system	2
1.5	Simulations and related research	3
1.6	Other applications and frameworks	3
2	Introduction	4
2.1	History	4
2.1.1	Early services	4
2.2	Peer-to-peer related services	5
2.2.1	Usenet	5
2.2.2	Domain Name System	5
2.2.3	Routing inside autonomous systems	6
2.2.4	Multicast	6
2.3	Common problems	6
2.3.1	Fire walls	6
2.3.2	Port abuse	6
2.3.3	Dynamic host addresses	6
2.3.4	Network Address Translation	7
2.4	Examples on different services	7
2.4.1	Client/server: browsing the web	7
2.4.2	Centralized peer-to-peer: Napster	8
2.4.3	Decentralized peer-to-peer: Gnutella	9
3	Peer-to-peer services	10
4	Gnutella	11
4.1	Introduction	11
4.2	Protocol details	11
4.2.1	Establishing connections	12
4.2.2	Message header	13
4.2.3	Ping message	14
4.2.4	Pong message	14

4.2.5	Query message	15
4.2.6	QueryHit message	15
4.2.7	Push message	16
4.2.8	Bye message, version 0.6	17
4.3	Ping and Pong optimizations	18
4.3.1	Pong caching	18
4.3.2	Ping multiplexing	18
4.4	Managing Query and QueryHit messages	18
4.5	Flow control	18
4.6	Ultrappeers and routing	18
4.7	Transferring files	18
4.8	Metadata	18
5	Analysis	19
6	Design	20
7	Implementation	21
8	Simulations	22
9	Conclusions	23
10	Other applications	24
11	Summary	25
	Bibliography	26

Chapter 1

Requirements

This chapter specifies the details concerning this thesis. All requirements are specified in sufficient detail.

1.1 Summary of requirements

The goal of this thesis is to present a thorough study of peer-to-peer services. The study should present the flaws in today's services and suggest improvements. File sharing, the most well known service, will be the service receiving the most attention. The requirements of the thesis can be summarized as follows.

- Investigate existing peer-to-peer services and analyze their performance
- Make a more in depth analysis of the Gnutella protocol
- Construct a more efficient system for distributed file sharing, which should consider
 - caching methods
 - hierarchical addressing
 - specialized directories
- Run simulations on the constructed system and relate the results to existing research
- Look at other applications and frameworks for peer-to-peer services

1.2 Investigate existing services

There are several services that use the peer-to-peer architecture. The most famous service is file sharing, but other services are also worth investigating. The following services will be investigated.

- Collaborative environments
- Distributed computing
- File sharing
- Instant messaging

These services all play important roles in the evolution of peer-to-peer and present interesting aspects of cooperation and computing. The services can briefly be described as follows.

Collaborative environments simplifies the process of several people working together. A well known collaborative environment is the *WikiWiki-Web*, where users can add, modify, restructure and modify content arbitrarily.

Distributed computing deals with the fact that many computers spend lots of their time being idle. The combined power of many workstations and PCs together form a cluster of immense performance. Distributed computing is the concept of using these wasted CPU-cycles for useful computations.

File sharing is, by far, the most commonly used peer-to-peer service. It lets people share files residing on their computers. File sharing applications allow people perform context based searches. Gnutella is a well known file sharing protocol implemented by several vendors. Some services are aimed at sharing disk space rather than files, but since the space eventually is occupied by files a similar result can be achieved. *Freenet* is an example of such a service.

Instant messaging is a service that allows users to send messages to others. The most well known service that uses instant messaging is perhaps ICQ.

1.3 Analyzing the Gnutella protocol

The Gnutella protocol has become the de-facto standard protocol for peer-to-peer based file sharing. Since this protocol is widely used an in depth analysis is in order. The flaws of Gnutella will be explored and possible solutions will be discussed.

1.4 Constructing a file sharing system

Based on the studies of the Gnutella protocol and suggested solutions to its' limitations a more efficient file sharing system should be constructed. The most important part of the constructed system is that it is able to share files in a similar fashion as Gnutella, but wastes less network resources.

If there is enough time the file sharing system can be converted into a complete application for file sharing. If this is the case, a well designed interface has to be created as well.

1.5 Simulations and related research

Other scientists have also studied the Gnutella protocol and an important part of this part of the thesis is presenting results from several sources. This makes it easier to draw conclusions concerning the behaviour of Gnutella and provides a more stable ground on which to interpret the simulation results.

The constructed file sharing system is tested and its performance is measured and, if possible, compared against Gnutella. Other attempts to optimize file sharing should also be investigated.

1.6 Other applications and frameworks

File sharing is the most commonly used application, but to make the thesis complete a look at other applications is important. Identifying if other services have similar problems as Gnutella and if they can be solved in a similar manner provides most of the material for this part of the thesis.

Some attempts to generalize peer-to-peer services have been attempted, for example the JXTA project by Sun Microsystems. These frameworks for peer-to-peer services should be investigated to see how well they solve the problems encountered. If flaws are detected in these frameworks suggestions on how to improve them should also be presented.

Chapter 2

Introduction

This chapter introduces the necessary terminology and historical ideas that provide the foundation for peer-to-peer networking. Common problems encountered by peer-to-peer services are identified. Some aspects on today's network architecture that contributes to these problems are also presented.

2.1 History

The history of the *Internet* began in the United States in the late 1960's. Originally named *ARPANET*, the Internet was primarily used to share computing resources between university campuses. These campuses were at the time already independent computing sites and the purpose of the ARPANET was to connect these sites using an architecture where all hosts were equals. With the terminology used today this network arrangement, where all hosts are considered to be equals, is called *peer-to-peer*. [Ora01]

In the early days the Internet was much more open than it is today. Security was not a big concern and connections could be made directly to any host connected to the Internet. The increasing security concerns has, besides better protection, led to difficulties for efficient use of certain services. For more details see section 2.3 on page 6.

2.1.1 Early services

Although the initial network architecture was peer-to-peer based the earliest applications used extensively, FTP (see [FTP85] for details) and *Telnet* (see [Tel83] for details), were based on the concepts of *client* and *server*. A server is a host providing a certain service and a client is a host using this service.

These applications were client/server applications, but the idea was to allow all hosts to work both as clients and servers. Any host accepted FTP and Telnet connections and were also able to connect to any other host. This made the Internet as a whole work as a peer-to-peer network.

Client/server protocols are usually based on the client sending a *request* to the server. The server manages the request and sends a *response* (sometimes called a *reply*) back to the client. Telnet, FTP and web browser clients work in this fashion. There are many more client/server based applications available and an exhaustive list would occupy most of this report.

2.2 Peer-to-peer related services

Most people feel that peer-to-peer networking is something new, but it is not. As mentioned earlier, the early Internet was a peer-to-peer network where all hosts were equals. There has also been several applications that have worked in a manner resembling today's peer-to-peer systems, although they have not been treated as such by most people.

These services are not as strictly distributed peer-to-peer systems as for example Gnutella, but instead they use a hierarchical architecture leading to better performance.

2.2.1 Usenet

News has always been one of the most important network services although other more popular services has arrived over the years. Since the late 1970's Usenet news has been an important part of the Internet and is referred to as "the grandfather of today's peer-to-peer applications". [Ora01]

Originally Usenet used UUCP (Unix-to-Unix copy protocol). This protocol allowed one Unix machine to connect to another, exchange files and disconnect. In this fashion Usenet used UUCP to exchange messages within a set of topics.

The growth of the networks, the number of topics and the extensive growth of TCP/IP has led to that the Usenet uses NNTP (Network News Transfer Protocol, see [NNT86] for details) instead of UUCP. This new protocol allows Usenet machines to discover new newsgroups and exchange messages in each group.

2.2.2 Domain Name System

In the early days of the Internet all hosts stored a file named `hosts.txt` where all existing domain names were mapped to the corresponding IP addresses. A *domain name* is a more human friendly representation of an IP address, such as `www.ietf.org` representing the web server of the Internet Engineering Task Force (IETF) organization. In December 2002 this corresponded to the IP address 4.17.168.6, which most likely is harder to remember. Another benefit of using human friendly names is that the IP address corresponding to a certain name could be changed without affecting the users, as long as the human friendly name remains the same.

Having all computers on the internet store information about all other hosts was not feasible in the long run, and in 1983 the *Domain Name System*

(DNS) was created (see [DNS87] for all details). The DNS uses a hierarchical system of names, which has allowed it to withstand the immense growth of the Internet without seriously reduced performance.

The use of hierarchical distribution of knowledge turns out to be a clever way to maintain good performance despite constantly increasing demand. For this reason hierarchical structures are worth investigating for peer-to-peer services used today. More details on how DNS works will be presented later when hierarchical methods are discussed in chapter 5 on page 19.

2.2.3 Routing inside autonomous systems

... discard this one?

2.2.4 Multicast

... discard this one?

2.3 Common problems

With the original appearance of the Internet, where most hosts supported the same services, peer-to-peer applications could be developed fairly easy. At that time network performance was limited, basically eliminating the use of such applications.

2.3.1 Fire walls

When the networks grew larger administrators became more concerned with security and started to protect their networks. At this point fire walls started to appear at many places protecting a local area network from the surrounding Internet. This causes problems, since the hosts within a protected local area network are not equal to hosts that are not protected by fire walls. It is usually impossible to connect to a host behind a fire wall using some arbitrary port number. This problem is henceforth simply referred to as the *fire wall problem*.

2.3.2 Port abuse

Since many fire walls allow connections to hosts on the local area network if that connection is made to a certain port (usually port 80, the port used by HTTP) many peer-to-peer applications (and other network applications as well) use port 80 for many other things than it was originally intended to be. This is sometimes referred to as the *abusing port 80 problem*.

2.3.3 Dynamic host addresses

As the networks grew even further the address space of IPv4, (Internet Protocol version 4, see [IP81] for details) started to become exhausted. Since many

ISPs (Internet Service Providers) were assuming that their customers spent their time on the Internet downloading content, and not uploading, permanent IP addresses were not necessary. This caused the DHCP (Dynamic Host Configuration Protocol, see [DHC93] for details) to appear. This protocol automatically distributes IP addresses to the connected hosts. This allowed ISPs to reuse addresses more easily, and only maintain a pool of addresses instead of assigning a unique address to each host on the network. This problem is from now on referred to as the *dynamic IP address problem*.

2.3.4 Network Address Translation

Another invention to prevent the addresses in IPv4 to run out was NAT (Network Address Translation, see [NAT00] for details). NAT allows a router to act as a address translation proxy for an entire network. This means that the hosts inside the network could use addresses used by some other network internally, but when one of these hosts wants to access an external resource the NAT router replaces the internal address with an external address. When a response arrives at the NAT router it translates the external address back to the internal address and sends the response to the appropriate host on the network.

2.4 Examples on different services

To illustrate the differences between various services three different categories are presented. The client/server example shows how a web page containing embedded images is downloaded by a web browser. The centralized peer-to-peer example illustrates how Napster was used to locate and download an MP3-music file. Finally a decentralized system is used to illustrate how a file can be found and downloaded using the Gnutella protocol.

2.4.1 Client/server: browsing the web

Figure ?? on page ?? shows a typical session for downloading a web page containing some (in this case two) embedded images. Some details concerning the establishment of TCP connections are left out. Some details concerning HTTP (Hypertext Transfer Protocol, see [HTT99] for details) are mentioned. The following steps occur:

1. The client (in this case the web browser) tries to *connect* to the web server, usually using port number 80.
2. The server *accepts* the connection.
3. The client *requests* a web page (in this case `index.html`). This is performed by using the GET command of HTTP.

4. The server *responds* by sending the requested web page. The server also sends (as a part of the reply) an indication that the operation was successful.
5. The client parses the response and detects that there are some embedded images on this web page.
6. The client issues *requests* for these images to the server.
7. The server *responds* by sending the images to the client.
8. The client *closes* the connection to the server.

The older version of HTTP, version 1.0 (see [HTT96]), stated that the client closes the connection between each request and hence needed to reestablish the connection with the server to request the embedded images. This made the protocol very simple, but somewhat inefficient. The later version of HTTP, version 1.1 (see [HTT99]), allowed clients to maintain a session with the server, making it possible to request the embedded images using the same connection.

2.4.2 Centralized peer-to-peer: Napster

Figure ?? on page ?? shows a Napster client locating and downloading a music file from another host on the Internet. The following steps occur:

1. The client tries to *connect* to the Napster directory server.
2. The directory server *accepts* the connection.
3. The client sends a *query* to the directory server describing the music file wanted. This could for example include the name of the song, the artist performing it or the name of the album where it resides.
4. The directory server processes the query and sends a *response*, containing the hosts that have music files matching the query, back to the client.
5. The client *disconnects* from the directory server.
6. The client *connects* to one or more of the hosts that have the music file.
7. Hopefully at least one of the hosts *accepts* the connection.
8. The client *requests* the music file.
9. The other host (or possibly hosts) *respond* by sending the music file.
10. When the client has downloaded the file it *disconnects*.

The most important part of the Napster system is the directory server. It contains a large database with available music files. A host connecting to the Napster directory server adds descriptions for all its' music files to the database. Queries to the directory server can then be processed efficiently. To make good use of bandwidth the actual download takes place directly between the client and the host storing the file.

2.4.3 Decentralized peer-to-peer: Gnutella

The performance of Napster is superior to that of Gnutella because a central directory server is available. Since legal matters have basically stopped the use of Napster another approach was needed. Both Napster and Gnutella have been used to distribute multimedia illegally over the Internet, but since there is no central point of connectivity in Gnutella it is much harder to shut down.

A host in a Gnutella network is commonly referred to as a *servent* (note the spelling). This name comes from combining the words server and client. Figure ?? on page ?? shows when a servent wishes to locate a file and download it. The following steps occur:

1. The servent *contacts its neighbours* in the Gnutella network and sends the *query* to these hosts.
2. The neighbours *process* the query to see if they have anything that matches the query. If they have, they send a *response*.
3. The neighbours send the query to their *other neighbours*.
4. These neighbours-of-neighbours repeat the steps performed by the neighbours.
5. After a certain number of steps the query stops propagating through the network, due to the expiration of a TTL (time to live) counter.
6. If the initial servent receives responses it can *connect* to the host storing the file and request the file.
7. The host storing the file *accepts* the connection and *responds* with the requested file.

Chapter 3

Peer-to-peer services

This chapter presents an overview of some of today's peer-to-peer services. Different service categories are presented and discussed. The chapter ends by concluding what properties these services have in common and if they are in need of improvement.

Not yet written

Chapter 4

Gnutella

Gnutella, the de-facto standard protocol for file sharing, is presented in this chapter. All the necessary details of the protocol is presented.

4.1 Introduction

Gnutella is a *file sharing protocol*. Using a Gnutella client files shared by a certain host can be located and downloaded by another Gnutella client. The protocol has a simple structure and uses broadcasts to locate files. The material in this chapter is based on the Gnutella Protocol Specification 0.4 ([Gnu00]) and the 0.6 draft ([Gnu02]) unless specified otherwise.

There are many applications available that use Gnutella. Two of the most widely used Gnutella clients are the following.

BearShare is a Gnutella client for Windows being updated frequently and containing the latest features of Gnutella clients. For more information see <http://www.bearshare.com>.

Limewire is a Java based, open source Gnutella client. Contains the latest features in the Gnutella field and runs on most platforms. For more information see <http://www.limewire.org>.

These and some other Gnutella clients can be found through the website <http://www.gnutelliums.com>, where clients for several platforms are listed.

4.2 Protocol details

The Gnutella protocol is based on maintaining TCP connections to a number of other Gnutella hosts. These hosts are from now on simply referred to as *neighbours*. In order to find new neighbours a Gnutella server broadcasts a Ping message. If a Gnutella server receives a Ping message it responds with a Pong message.

A server issues a query by sending a `Query` message to all its neighbours. The neighbours pass this message on to their other neighbours and so on, for a certain number of steps defined by the `TTL` (Time To Live) field in the message header.

A server receiving a query checks if it has something that matches this query. If it does, it responds with a `QueryHit` message, but still passes the `Query` message on to its other neighbours.

If a server receives a `QueryHit` message from a host that does not support incoming connections a `Push` message is sent to that host. This message causes the host holding the file in question to open the connection. This usually makes it possible to bypass fire walls.

The remainder of this chapter describes the components of the protocol more in detail. The descriptions are based on both the Gnutella protocol version 0.4 and the Gnutella protocol draft version 0.6. If no version number is given a certain description applies to both versions.

4.2.1 Establishing connections

When a server *C* has found another Gnutella server *S* and wishes to open a connection the following steps occur

Gnutella 0.4

1. *C* sends the string `GNUTELLA CONNECT/version\n\n` where *version* in this case is 0.4 and `\n` denotes the newline character (ASCII character 10)
2. *S* responds, if it chooses to accept the connection, with `GNUTELLA OK\n\n`
3. Connection is established and *C* and *S* can exchange messages

Gnutella 0.6

1. *C* sends the string `GNUTELLA CONNECT/version\r\n` where *version* in this case is 0.6 and `\r` denotes the carriage return character (ASCII character 13)
2. *C* sends capability headers (not including vendor specific headers), each terminated by `\r\n`, with an extra `\r\n` at the end
3. *S* responds with the string `GNUTELLA/0.6 200 string\r\n`. The *string* should be OK, but clients are advised to only check the 200 code
4. *S* sends all its headers in the same format as in step 2
5. *C* responds with the string `GNUTELLA/0.6 200 OK\r\n`, as in step 3 if the client, after parsing all headers sent by *S*, still wants to connect.

Otherwise a reply containing an error code is sent and the connection is closed

6. *C* sends any vendor specific headers to *S* in the same format as in step 2
7. Connection is established and *C* and *S* can exchange messages

4.2.2 Message header

Each Gnutella message has a header. This header contains information describing what kind of message it is, how much further it should be sent and a unique identifier for this message. All fields in the message header are in *network byte order* (big endian). Table 4.1 shows the structure of the message header.

Bytes	Description
0–15	Message ID (globally unique)
16	Payload type
17	TTL (Time To Live)
18	Hops
19–22	Payload length

Table 4.1: Gnutella message header fields

The fields in the Gnutella message header have the following meaning and semantics.

Message ID is a 16-byte string that is globally unique and identifies a message on the network. Gnutella version 0.6 states that byte 8 (if bytes are numbered 0–15) should be all 1’s to indicate that the id belongs to a modern server. Gnutella version 0.6 also states that byte 15 should be zero, since it is reserved for future use.

Payload type denotes the type of message. It should have one of the values shown in table 4.2.

Value	Message type
0x00	Ping message
0x01	Pong message
0x02	Bye message (only Gnutella 0.6)
0x40	Push message
0x80	Query message
0x81	QueryHit message

Table 4.2: Gnutella message header payload types

Time to live holds the number of times the message will be forwarded by Gnutella servents before it is removed from the network. Each servent decrements the TTL before sending the message to another servent. When the TTL reaches 0, the message will not be forwarded further.

Hops contains the number of hops the message has been forwarded before reaching the current servent. As a message is passed from servent to servent the TTL and Hops fields satisfy the following condition.

$$\text{TTL}_0 = \text{TTL}_n + \text{Hops}_n$$

In this equation TTL_0 denotes the initial TTL (which usually is 7) and TTL_n and Hops_n is the value of TTL and Hops after n hops.

Payload length contains the length of the message following the message header. No padding is used. Messages should not be larger than 4 kB.

4.2.3 Ping message

Ping messages do not have any payload. The message header contains a payload type of 0x00 and a payload length of 0x00000000. Gnutella version 0.6 allows Ping packets to have an extension block defined using GGEP (Gnutella Generic Extension Protocol). Servents are recommended to implement GGEP.

Ping messages are broadcasted to all reachable hosts. The TTL field limits the *horizon* of the broadcast. The horizon is the edge where messages are discarded due to expiration of the TTL. No TTL should exceed 7 in order to limit network overhead.

4.2.4 Pong message

Pong messages contain information about a Gnutella host and are sent as response to an incoming Ping message. The message payload type is 0x01 and the message has fields shown in table 4.3. All fields are in network byte order (big endian), except the IP address in Gnutella version 0.4 which is in little endian byte order.

Bytes	Description
0-1	Port number
2-5	IP address
6-9	Number of shared files
10-13	Number of kilobytes shared
14-	Optional GGEP extension block

Table 4.3: Gnutella Pong message fields

4.2.5 Query message

Query messages, with a payload type of 0x80, are broadcasted to all neighbour nodes in the Gnutella network. Messages larger than 256 bytes may be discarded to lower the load on the network. A Query message has the fields shown in table 4.4.

Bytes	Description
0-1	Minimum speed, kb/s
2-	Null terminated search criteria
Rest	Optional extensions

Table 4.4: Gnutella Query message fields

The fields in the Query message have the following meaning and semantics.

Minimum speed is the lower bound of connection speed a servent accepts.

A servent receiving a Query message should only respond to the querying servent with a QueryHit if it is able to communicate at least at this speed.

Search criteria is a string of keywords. Servent should only respond with files matching *all* keywords. Matching should be case insensitive. GGEP extensions may be used to specify an alternate matching procedure, such as interpreting the search criteria as a regular expression, but servents can never be sure that other servents understand these GGEP directives.

A Query message with a TTL field of 1, a hops field of 0 and a search criteria of exactly *four spaces* is used to index all files a host is sharing. If necessary several QueryHit messages may be sent. Servents should reply with all its shared files, unless it considers it harmful for privacy or bandwidth.

Optional extensions is only available in Gnutella version 0.6. Allowed extension types are HUGE (Hash/URN Gnutella Extensions), XML or GGEP. For more information on these extensions see [Gnu02].

4.2.6 QueryHit message

The QueryHit message, with a payload type of 0x81, is sent in response to a Query message when its search criteria matches one or more files shared by the servent. The fields of a QueryHit message are shown in table 4.5 on the next page. All fields are in network byte order (big endian).

Each element in the result set of a QueryHit message has the fields shown in table 4.6 on the following page. It has quite an odd structure with fixed length fields at the beginning and end, with variable length fields in the middle.

Bytes	Description
0	Number of hits (elements in result set)
1-2	Port number of responding host
3-6	IP address of responding host
7-10	Speed of responding host, kb/s
11-	Result set

Table 4.5: Gnutella QueryHit message fields

Bytes	Description
0-3	File index (unique)
4-7	File size in bytes
8-	File name, null terminated
<i>x</i> -	Extension blocks (HUGE, GGEP, plain)
<i>y</i> -	Recommended EQHD block
<i>z</i> -	Private vendor specific data
Last 16	Responding servent identifier

Table 4.6: Gnutella QueryHit result set fields

The extension blocks, EQHD block and private vendor specific fields are part to the Gnutella 0.6 draft. The EQHD block contains some vendor information structured in the fields shown in table 4.7. For more details see [Gnu02].

Bytes	Description
0-3	Vendor code (four case insensitive characters)
4	Open data size
5-	Open data. Flags describing servent capabilities

Table 4.7: Gnutella QueryHit EQHD block

4.2.7 Push message

The Push message is used to make it possible to download data stored on a host residing behind a fire wall. Instead a servent receiving a QueryHit, from a servent that does not accept connections, issues a Push message. When a Push message is received a servent should act if and only if the servent identifier field contains the receiving servants identifier.

The fields in a Push message are shown in table 4.8 on the following page. For more detailed descriptions see either [Gnu00] or [Gnu02].

Bytes	Description
0–15	Servent identifier of host having file
16–19	Index of file to push
20–23	IP address of host wanting file
24–25	Port number to push to
26–	Optional GGEF extension block

Table 4.8: Gnutella Push message fields

4.2.8 Bye message, version 0.6

The `Bye` message was introduced in the Gnutella 0.6 draft and is used to inform the servent, or servents, a node is connected to that the node is closing the connection. Since implementing this message is optional a special header must be sent during the connection handshake.

`Bye-Packet: 0.1`

Servents must not send `Bye` messages to hosts that has not indicated that they support this message. The TTL field in a `Bye` message must be set to 1 to avoid accidental propagation.

Upon receiving a `Bye` message a servent closes the connection in question immediately. The servent sending the message must wait a few seconds for the remote servent to close the connection before closing it. No data may be sent after the `Bye` message. The `Bye` message has the fields shown in table 4.9.

Bytes	Description
0–1	Code (as classified by SMTP)
2–	Description string, null terminated

Table 4.9: Gnutella Bye message fields

The code stored in the first two bytes correspond to return codes specified by SMTP (Simple Mail Transfer Protocol, [SMT01]). For example 200 means that everything is ok and 502 means that the send queue became full. For details on the semantics of individual codes see [Gnu02].

- 4.3 Ping and Pong optimizations**
 - 4.3.1 Pong caching
 - 4.3.2 Ping multiplexing
- 4.4 Managing Query and QueryHit messages**
- 4.5 Flow control**
- 4.6 Ultrapeers and routing**
- 4.7 Transferring files**
- 4.8 Metadata**

Chapter 5

Analysis

This chapter analyzes the Gnutella protocol and provides the foundation for a design of a more efficient peer-to-peer protocol. Existing research on this topic is summarized and the chapter is concluded by establishing the necessary components and their interaction in an efficient protocol.

Not yet written, but some analysis has been performed

Chapter 6

Design

This chapter presents the design of an efficient peer-to-peer file sharing system. This design is based on the analysis of Gnutella and contains protocol discussions and considerations such as partial centralization.

Not yet written, but some design ideas have been sketched

Chapter 7

Implementation

This chapter presents the internal details of an efficient peer-to-peer file sharing system. Focusing on components and their interaction this chapter gives an understanding of how a system for distributed file sharing could be implemented.

Not yet written

Chapter 8

Simulations

This chapter presents several simulation scenarios where the implemented system is tested. Several aspects of performance is measured and, if possible, the result is compared against the Gnutella protocol.

Not yet written

Chapter 9

Conclusions

This chapter analyzes the results obtained from simulating the peer-to-peer implementation. Conclusions regarding various aspects of performance and efficiency is presented.

Not yet written

Chapter 10

Other applications

There is more to peer-to-peer than sharing files. This chapter discusses other applications of peer-to-peer and how they would benefit from the earlier work performed in this thesis.

Not yet written

Chapter 11

Summary

This chapter summarizes the entire thesis. It also presents the most important results and conclusions. The chapter ends with a discussion on what could be done in the future to follow up this work.

Not yet written

Bibliography

- [ACMD⁺02] KARL ABERER, PHILIPPE CUDRÉ-MAUROUX, ANWITAMAN DATTA, ZORAN DESPOTOVIC, MANFRED HAUSWIRTH, MAGDALENA PUNCEVA, ROMAN SCHMIDT, AND JIE WU. *Peer-to-Peer Computing*. Technical Report 73, Distributed Information Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2002.
- [APHS02] KARL ABERER, MAGDALENA PUNCEVA, MANFRED HAUSWIRTH, AND ROMAN SCHMIDT. *Improving Data Access in P2P Systems*. *IEEE Internet Computing*, 6(1):58–67, January 2002.
- [BWDD02] PETER BACKX, TIM WAUTERS, BART DHOEDT, AND PIET DEMEESTER. *A comparison of peer-to-peer architectures*. In *EURESCOM Summit*, Heidelberg, Germany, October 2002.
- [Chi01] ANDREW A. CHIEN. *Security Requirements Document*. Technical report, Peer-to-Peer Working Group, November 2001.
- [CMH⁺02] IAN CLARKE, SCOTT G. MILLER, THEODORE W. HONG, OSKAR SANDBERG, AND BRANDON WILEY. *Protecting Free Expression Online with Freenet*. *IEEE Internet Computing*, 6(1):40–49, January 2002.
- [DHC93] Internet Engineering Task Force. *Dynamic Host Configuration Protocol*, October 1993. RFC 1541.
- [DNS87] Internet Engineering Task Force. *Domain Names*, November 1987. RFC 1034 (Concepts and Facilities), 1035 (Implementation and Specification).
- [FTP85] Internet Engineering Task Force. *File Transfer Protocol*, October 1985. RFC 959.
- [GHI⁺01] STEVEN GRIBBLE, ALON HALEVY, ZACHARY IVES, MAYA RODRIG, AND DAN SUCIU. *What Can Databases Do for Peer-to-Peer?* In *Proceedings of the Fourth International Workshop on the Web and Databases*, 2001.

- [Gnu00] Clip2 Distributed Search Services. *The Gnutella Protocol Specification v0.4*, 2000.
- [Gnu02] *Gnutella 0.6 Protocol Draft*, June 2002.
- [Gon02] LI GONG. *Peer-to-Peer Networks in Action*. *IEEE Internet Computing*, 6(1):37–39, January 2002.
- [HHH⁺02] MATTHEW HARREN, JOSEPH M. HELLERSTEIN, RYAN HUEBSCH, BOON THAU LOO, SCOTT SHENKER, AND ION STOICA. *Complex Queries in DHT-based Peer-to-Peer Networks*. In *International Workshop on Peer-to-Peer Systems*, March 2002.
- [HTT96] Internet Engineering Task Force. *Hypertext Transfer Protocol — HTTP/1.0*, May 1996. RFC 1945.
- [HTT99] Internet Engineering Task Force. *Hypertext Transfer Protocol — HTTP/1.1*, June 1999. RFC 2616.
- [IP81] Internet Engineering Task Force. *Internet Protocol*, September 1981. RFC 791.
- [IP98] Internet Engineering Task Force. *Internet Protocol, Version 6 (IPv6)*, December 1998. RFC 2460.
- [JBBS01] MATTHEW B. JONES, CHAD BERKELEY, JIVKA BOJILOVA, AND MARK SCHILDHAUER. *Managing Scientific Metadata*. *IEEE Internet Computing*, 5(5):59–68, September 2001.
- [Jos02] SAM JOSEPH. *NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks*. In *Web Engineering and Peer-to-Peer Computing, NETWORKING 2002 Workshops*, Pisa, Italy, May 2002.
- [LHC⁺02] RAINER LIENHART, MATTHEW HOLLIMAN, YEN-KUANG CHEN, IGOR KOZINTSEV, AND MINERVA YEUNG. *Improving Media Services on P2P Networks*. *IEEE Internet Computing*, 6(1):73–77, January 2002.
- [MAJ01] KENNETH A. BERMAN MIHAJLO A. JOVANOVIĆ, FRED S. ANNEXSTEIN. *Modeling Peer-to-Peer Network Topologies through “Small-World” Models and Power Laws*. In *Telecommunications Forum*, Belgrade, Yugoslavia, November 2001.
- [MKL⁺02a] DEJAN S. MILOJICIC, VANA KALOGERAKI, RAJAN LUKOSE, KIRAN NAGARAJA, JIM PRUYNE, BRUNO RICHARD, SAMI ROLLINS, AND ZHICHEN XU. *Peer-to-Peer Computing*. Technical Report 57, Hewlett-Packard Laboratories, Palo Alto, California, March 2002.

- [MKL⁺02b] DEJAN S. MILOJICIC, VANA KALOGERAKI, RAJAN LUKOSE, KIRAN NAGARAJA, JIM PRUYNE, BRUNO RICHARD, SAMI ROLLINS, AND ZHICHEN XU. *Peer-to-Peer Computing*. Technical Report 198, Hewlett-Packard Laboratories, Palo Alto, California, July 2002.
- [NAT00] Internet Engineering Task Force. *Network Address Translation*, February 2000. RFC 2766.
- [NNT86] Internet Engineering Task Force. *Network News Transfer Protocol*, February 1986. RFC 977.
- [Ora01] ANDY ORAM, editor. *Peer-to-Peer — Harnessing the Power of Disruptive Technologies*. O’Reilly & Associates, Sebastopol, California, first edition, March 2001.
- [PBB⁺01] JAMES S. PLANK, ALESSANDRO BASSI, MICAH BECK, TERENCE MOORE, D. MARTIN SWANY, AND RICH WOLSKI. *Managing Data Storage in the Network*. *IEEE Internet Computing*, 5(5):50–58, September 2001.
- [PSW01] MANOJ PARAMESWARAN, ANJANA SUSARLA, AND ANDREW B. WHINSTON. *P2P Networking: An Information-Sharing Alternative*. *IEEE Computer*, 34(1):31–38, July 2001.
- [RIF02] MATEI RIPEANU, ADRIANA IAMNITCHI, AND IAN FOSTER. *Mapping the Gnutella Network*. *IEEE Internet Computing*, 6(1):50–57, January 2002.
- [Rip01] MATEI RIPEANU. *Peer-to-Peer Architecture Case Study: Gnutella Network*. Technical Report 26, University of Chicago, July 2001.
- [RWE⁺01] SEAN RHEA, CHRIS WELLS, PATRICK EATON, DENNIS GEELS, BEN ZHAO, HAKIM WEATHERSPOON, AND JOHN KUBIATOWICZ. *Maintenance-Free Global Data Storage*. *IEEE Internet Computing*, 5(5):40–49, September 2001.
- [SGG02] STEFAN SAROIU, P. KRISHNA GUMMADI, AND STEVEN D. GRIBBLE. *A Measurement Study of Peer-to-Peer File Sharing Systems*. In *Proceedings of Multimedia Computing and Networking*, San Jose, California, January 2002.
- [SLS01] VINCENT SCARLATA, BRIAN NEIL LEVINE, AND CLAY SHIELDS. *Responder Anonymity and Anonymous Peer-to-Peer File Sharing*. In *IEEE International Conference on Network Protocols*, pages 272–280, November 2001.
- [SMT01] Internet Engineering Task Force. *Simple Mail Transfer Protocol*, April 2001. RFC 2821.

- [SY01] SUMEET SOBTI AND PETER N. YIANILOS. *The Evolving Field of Distributed Storage*. *IEEE Internet Computing*, 5(5):35–39, September 2001.
- [TCP81] Internet Engineering Task Force. *Transfer Control Protocol*, September 1981. RFC 793.
- [Tel83] Internet Engineering Task Force. *Telnet Protocol Specification*, May 1983. RFC 854.
- [WDKF02] STEVE WATERHOUSE, DAVID M. DOOLIN, GENE KAN, AND YAROSLAV FAYBISHENKO. *Distributed Search in P2P Networks*. *IEEE Internet Computing*, 6(1):68–72, January 2002.