Institute for Security Technology Studies
Dartmouth College

File Sharing Protocols: A Tutorial on Gnutella[1]

Vincent Berk and George Cybenko

Institute for Security Technology Studies
Dartmouth College
Hanover NH 03755
http://www.ists.dartmouth.edu/IRIA

March 6, 2001

This report contains short descriptions of several popular file sharing protocols. Legal and law enforcement aspects of file sharing are briefly reviewed. File sharing is also known as peer-to-peer (P2P) networking. A general overview of P2P protocols is followed by a more detailed analysis of the GNUTELLA protocol, specifically from the point of view of scalability and security. GNUTELLA is designed to share files across the Internet, with each node being both client and server. A simple analysis shows that GNUTELLA provides neither reliability nor security for several reasons.

[1]Research partially supported by the National Institute of Justice, US Department of Justice and the Defense Advanced Research Projects Agenc. Any opinions expressed in this report are solely those of the authors.

1 Introduction

1.1 Overview

FIle sharing, also known as peer-to-peer (P2P) networking, has been called the biggest application breakthrough on the Internet since the introduction of web browsing software about 7 years ago. In recent months, the bulk of media attention has been focused on Napster. Napster is only one of many currently available file sharing protocols. However, the popularity and commercial goals of Napster have resulted in highly visible, complex legal proceedings concerning the redistribution of intellectual property. (Intellectual property crimes typically fall under Federal jurisdication. See http://www.cybercrime.gov/ipmanual.htm and http://www.cybercrime.gov/ipmanual/chart.htm specifically for more information).

The main goal of peer-to-peer networking is to allow users to share files without storing them on central servers. In principle, this could already be accomplished with standard web technology. Users could create a web site, load it with content they want to share, allow web search engines (such as Yahoo, Google, and others) to locate and index that content. Other users could then locate the content if they used appropriate keywords in a web search.

The two main reasons file sharing technology was developed in spite of the existence of traditional web technology are:

- Inexperienced end-users generally find it difficult to set up and maintain a web server on their personal machines; by contrast, installing a P2P application like Napster merely involves downloading and installing a simple application with minimal initial configuration or subsequent maintenance required;
- P2P users want to share specific types of content, mainly digital music in the MP3 format although images and other media types can be shared as well; existing web search engines do not specialize in these formats and return search results that are much broader than the content typically sought.

Advantages of P2P file sharing include:

- **Economy** - During daytime working hours, it is not uncommon for over 10,000 users to be connected to Napster and willing to share their music files; several terabytes (a 'tera' is one thousand 'giga' or 1,000,000,000,000) of storage are networked and available for sharing; by distributing the cost of disk space, an individual user carries a relatively small part of the overall cost of storage as compared with web hosting services; morevoer, a user does not even perceive that their cost (hundreds or even thousands of dollars for a consumer PC and disk storage) is being carried by them personally for such services, they focus on the "free" music to which they have access;
- **Perception of immunity from prosecution** - Because shared files are distributed over thousands of consumer-owned computers, it does not appear to be cost-effective to investigate so many individual, relatively small-scale users; Napster, by providing an index of shared files, is a single, tangible corporate entity and has been entangled in litigation over the legality of promoting illegal redistribution of copyrighted material; some other P2P protocols do not require a server and so are even more difficult to prosecute under copyright infringement statutes.

Web hosting services and Internet Service Providers (ISPs) do not enjoy these two advantages. Concentrating the cost of the massive storage required to store large quantities

of digital content is prohibitive for consumer-oriented services. Moreover, a corporate entity would be a much easier target for intellectual property owners and prosecutors to go after.

Additional background and tutorial material about P2P protocols and related technologies can be found at the following web sites:

- **http://www.zeropaid.com** - Zeropaid is an excellent resource with links to all major P2P sites, tutorials and how-to's;
- **http://www.wired.com/wired/archive/8.10/p2p_pages_pr.html** - *WIRED* magazine has compiled a list of P2P and digital media sites on this page.

1.2 Legal and Criminal Issues

Law enforcement issues surrounding P2P protocols fall into four broad categories:

- Abuse of intellectual property rights and licenses;
- Distribution of illegal content, such as child pornography;
- Covert communication channels supporting criminal activity;
- Increased vulnerability to malicious code and weakened computer security.

Web resources dealing with potential law enforcement aspects of P2P file sharing protocols are:

- **http://www.cybercrime.gov/ipmanual.htm** - This is a comprehensive US Department of Justice web site devoted to legal issues surrounding US intellectual property law;
- **http://www.cybercrime.gov/ipmanual/chart.htm** - The chart on this page provides details of successful and pending prosecutions of intellectual property use and licensing violations;
- **http://www.zeropaid.com/busted/** - Zeropaid set up a Gnutella server with bogus files, named to suggest child pornography material; for a period of time, they tracked downloads and posted the IP addresses of downloaders on their "Wall of Shame"; their brief and limited experiment demonstrates the existence of a large and active potential market within the P2P user community for illegal content;
- **http://freenet.sourceforge.net/index.php?page=faq#sec1.6** - This posting briefly describes the philosophy of the P2P community regarding use of the protocols for criminal and terrorist activities, claiming that channels supporting such activities already exist anyway and that unfettered distribution of information is an social imperative;
- **http://www.nwfusion.com/newsletters/sec/2000/0828sec1.html?nf** - This page is an overview of computer security issues related to P2P networking and a good place to start reading about the vulnerabilities introduced by those protocols;
- **http://www.wired.com/news/print/0,1294,40443,00.html** - This is another *WIRED* magazine posting about security issues raised by P2P networking;

- **http://www.ktsi.net/pdf_files/Security_Concerns_Peer-to-Peer_KTSI.pdf** - This is a PDF file with a comprehensive analysis of P2P vulnerabilities from the point of view of network and computer security, addressing corporate issues and the problems that open protocols create with respect to Trojans and other malicious code that could be introduced by shareware implementations.

At the present time, http://www.zeropaid.com is the most comprehensive and up-to-date site dealing solely with most aspects of P2P networking and the reader is encouraged to monitor this site for new protocols and developments in the technology.

2 Major File Sharing Protocols

This section contains a brief description of the major P2P protocols at the time of this writing, March 2001. Links to web sites with more detailed information are provided at the end of each description.

2.1 NAPSTER, OPENNAP and RAPSTER

NAPSTER (and its open source counterpart OPENNAP) was originally designed to share MP3 music files across the Internet. It is based on a client-server architecture. The clients connect to a central server and report what files they provide for sharing. The server adds these files to its search list. Clients issue search requests to the server, which then responds with the search results, consisting of filenames and clients providing those files. File transfers are peer-to-peer, meaning that the server isn't involved in storing or transfering the actual MP3 files. It only tells clients where to get what they're looking for, which is located on another client. The protocol is well-defined and there are numerous implementations of both the client and the server available. Napster does not provide encryption of any kind, files are shared and transferred in the open. RAPSTER is a MacOS implementation of the NAPSTER protocol.

Websites:
NAPSTER: www.napster.com
OPENNAP: opennap.sourceforge.net
RAPSTER: www.macnews.com.br/overcaster/products/rapster.html

2.2 GNUTELLA

GNUTELLA is a general file-sharing protocol designed to elimitate the need for a central server. Each client can provide files for sharing with the community. Search requests get propagated from client to client and responses follow the same path back. The network is based on peer-groups. Each client builds up a (constantly changing) peer-group of other clients. Whenever a client receives a request it forwards it to its peer-group. The cummulative effect is that in very short time a large number of clients receive and process the request. The downside of this is, of course, the generated network traffic. A small number of requests can add up and clog a network completely. Various implementations of clients are available in the open source community. More information about GNUTELLA is contained in the following sections.

Website:
gnutella.wego.com

2.3 FREENET

Like GNUTELLA, FREENET is a general file-sharing protocol. Also, it does not require a central server, however message-passing is done in a less demanding manner. Instead of forwarding requests to all known peers, a FREENET node forwards a request to the node that is most likely to have a match. If a match is found the request chain terminates and an answer is returned. File transfers are done immediately between two nodes. The FREENET protocol is also freely available and various clients have already been written. FREENET supports encrypted file transfer but clients are still hard to configure.

Website:
freenet.sourceforge.net

2.4 GROOVE

GROOVE is very much unlike the protocols described above. GROOVE is in early release but is ultimately positioned to be a commercial product for worksgroup collaboration. Designed mainly for use within a local network the GROOVE is more like a shared workspace. There is no central server and all clients are peers and of equal importance. All connected users see the same window, in which files can be shared, pictures can be drawn or the Web can be browsed. All communications are encrypted but the GROOVE protocol is not open so reverse engineering will be difficult and time consuming. GROOVE uses a considerable amount of bandwidth. No Unix clients are available.

Website:
www.groove.net

2.5 HOTLINE

HOTLINE is client-server based, much like the traditional BBS systems. Everyone can create a server on which they provide files and a chatbox on a certain topic. Clients locate servers through "trackers," that keep updated lists of active servers and their topics. Clients can upload and download files and engage in a conversation. HOTLINE features picking up on discontinued downloads but does not support encryption. HOTLINE provides both client and server software for Windows and Mac OS's. Since the protocol is not publicly available most Unix and open source clients are not fully functional.

Website:
www.bigredh.com/hotline3

2.6 PUBLIUS

PUBLIUS is build mainly with "Censorship Resistance" in mind. A file gets encoded using a key, K, which is in turn split up into n shares, such that any k of them can regenerate the original key K. Next, the encrypted file is stored on multiple servers, together with a share of the key. In order to retrieve the file, at least k different key shares have to be collected from the various servers. As long as there are at least k different keyshares distributed among the available servers, the file is retrievable. Each server has a complete copy of the encrypted file and one keyshare. The protocol is fully described in a paper and both client (proxy) and server software is available.

Website:
cs1.cs.nyu.edu/waldman/publius

2.7 Comparison of Described P2P Protocols

Protocol	*NAPSTER/OPENAP*	*GNUTELLA*	*FREENET*	*GROOVE*	*HOTLINE*	*PUBLIUS*
Central Server	Y	N	N	N	Y (partial)	Y (multiple)
Open Protocol	Y	Y	Y	N	N	Y
Sharing Type	MP3	All files	All files	All files	All files	All files
Encryption	N	N	Y	Y	N	Y

Table 1: Overview of Different P2P Protocol Features

3 Gnutella Details

The first time a GNUTELLA node (node A) connects to another GNUTELLA node (node B), which already has a peer-group, it does this by sending a "GNUTELLA CONNECT" request. Node B replies by returning a "GNUTELLA OK". At this point node A has a connection with B and can start expanding its own peer-group (which currently only consists of node B). In order to do so, node A sends a "PING" to B.

The most important feature of GNUTELLA is that all nodes forward all incoming messages to all members of their peer-group. Each GNUTELLA message contains a unique identifier number in its header. (Actually this number is chosen randomly or generated using the Microsoft Globally Unique Identifier function. There seems to be no guarantee that this really is a unique number.) When a node receives a message, it stores this message ID in a table before sending it on to its peer-group. If the message ID is already in the table, it discards the message because it has already been passed on. When the message gets passed on, the Time To Live (TTL) is decremented. As soon as the TTL is zero, the message also gets discarded.

Node A sends a "PING" request to B, which forwards it to its peer-group and replies to it with a "REPLY". All nodes which receive this ping, reply to it and the packets get routed back, through B, to A. These reply packets contain the IP of the replier, its port, the number of files shared and the total number of bytes. Now node A has the IP addresses and the port numbers of all reachable hosts, through node B. (These obviously

contain the peer-groups of the peer-group of node B, etc. etc.) So A can now construct its own peer-group from the addresses received back.

Node A is now fully up and running and can conduct searches, forward messages to its peer-group and, of course, expand its own peer group by pinging again.

Suppose node A receives a search request. This is a GNUTELLA message with the search string as its message body. First, node A checks if it has seen this message ID already (if so, the message is dropped.) Assuming that this is a new search, node A forwards the search message to all members of its own peer-group (after decrementing the TTL). Next, node A looks in its list of shared files. If a match is found, A returns a packet with all the matches, indexed with unique numbers.

Lets say, for instance, that node C sends out the search packet described above, and that node A indeed found several matches. A returned the (indexed) matches to C. If node C decides to download one of the files from A, which matched its search query, C makes a direct connection with node A and requests the file by the index number. The file is transported via the HTTP protocol, directly between A and C. Node C doesn't necessarily have to be part of A's peer-group.

If node C is behind a firewall, A cannot connect to it to download the file. In that case, node A sends a so called PUSH packet to C. This is a request to C to connect to A and upload the file. This way the connection can be made through the firewall protecting C.

For a technical description of the GNUTELLA protocol, please refer to the appendix.

4 Gnutella Scalability

Recent testing has showed that GNUTELLA produces a lot of network traffic just to keep the connections alive. A typical node drops a peer-group connection after a minute or so. This happens for several reasons. First, this eliminates the problem of nodes shutting down, since this could happen without notice. Furthermore, this makes the peer-group dynamic and with it, the search scope. Two exactly equal search queries could have completly different results when issued only five minutes apart. After each change in the peer-group, a node has to send out a PING to find out which nodes are reachable.

We construct a simple formula to get an idea of the scaling issues. As said before, all GNUTELLA messages are treated equally. Let C be the (average) amount of direct connections (peer-group) at each node. Let T be the (average) Time To Live of the issued messages.

Position in network (Hops)	*Amount of messages*
0 (issuer)	C messages $+C$ replies
1 (level 1 peer group)	C^2 messages $+C^2$ replies
2 (level 2 peer group)	C^3 messages $+C^3$ replies
...	...
T (TLL=1 expires next hop)	C^T messages $+C^T$ replies

Table 2: Maximum number of messages with respect to TTL and Peer-Group Size

This boils down to:
Total amount of messages $= 2 \times \sum_{t=0}^{T} C^t \simeq \mathcal{O}(C^T)$

Even in the minimal case (typically $C = 25$ and $T = 5$) this already adds up to 500 milion packets for one PING. Normally, C would be around 100 and T up to 7.

Because of the limited number of actual nodes on the internet and the vast interconnectivity of the structure, most of the messages get terminated after the first two or three hops. Our experience is that currently an average peer-group (of 25 hosts) reaches about 500 hosts, generating sustained bandwidth of 150 KB/sec. These numbers include all (routed) message traffic (thus excluding the direct file transfers). With 2000 reachable hosts, it filled-up a T1 line. (And still there weren't any file transfers, just messages.)

(For clarity we will only refer to the group of reachable nodes from our local host as static, regardless of the fact that this group continuously changes.)

The amount of bandwidth required is highly dependent on the number of reachable hosts. As the total number of GNUTELLA nodes on the internet increases, it is clear that the total number of reachable hosts increases, thus meaning that there will be more search requests to process. Since the connection topology is so dynamic it is very hard to give an exact model for network traffic and bandwidth. Although searches get selected on network speed, connections are not. Important parts of the peer-group might only be reachable through a phone line, thus decreasing performance drastically.

Regardless of the physical network layer, some things are certain. The number of messages generated is linear with respect to the number of reachable hosts. To see this, every message that propagates through the network will only be sent on if it is unique. All duplicates get discarded. If one host is added, one extra message gets send on. Testing showed that the linearity holds roughly until the connection starts to fill up. That is also the point where requests get discarded.

It is still unclear which type of messages are the main cause of the bandwidth problem and if it really is just one message type. Depending on the structure of the GNUTELLA net (which obviously changes every second), wildly varying results have been reported. In some cases the PING/REPLY took 70% of the total bandwidth, in other the PUSH requests took 60%. Of course, depending on the intensity of search, the bandwidth taken by SEARCH/SEARCH REPLY messages will vary greatly. Cases have been reported it taken up to 90% of the total bandwidth.

5 Gnutella Reliability

Due to the constant changing of the peer-group, it is very difficult to ensure a connection between two given machines. It is even possible for a node to completly drop out of the network, although that is unlikely. Also, due to limited network bandwidth and processor power, search queries get dropped frequently. This typically happens when a GNUTELLA message has to pass through a node which cannot handle the huge amount of traffic.

The GNUTELLA protocol doesn't guarantee any reliability in any form.

6 Gnutella Security

6.1 General

Security is not part of the GNUTELLA protocol. All messages are sent in plain text, readable, *and modifiable*, by everyone. GNUTELLA has seen some serious cases of SPAM lately. This is quite simple. Nodes just return a commercial message on each and every search query. This can be done even without an actual GNUTELLA client because the message header of all GNUTELLA messages is formatted the same way. Filtering for the pattern allows a custom application to do this.

It would be very hard to embed conventional security technologies in the protocol. Every node through which a message passes should be able to read the full contents of the message. Encryption doesn't make sense since everyone with a GNUTELLA client/server should be able to to decrypt the package.

To avoid alteration of the message digital fingerprinting could be used, but this still leaves the problem of nodes returning SPAM on every search query. The actual file transfer, which is done between client and server directly, could be made secure with conventional methods for obvious reasons. This is a regular peer-to-peer file transfer.

To keep GNUTELLA within a specified domain, the software implementation should prevent connects to and from the outside world. This case offers more possibilities for encryption.

6.2 General threats to GNUTELLA

This section deals with the possible threats to the GNUTELLA network and its users.

As noted above, none of the transmitted information is encrypted. This leaves all messages vulnerable to any sort of attack, aimed at modifying the transmitted information. Furthermore, anyone connected to the internet through a static link (ie. uses the same link every time a connection is made) can be monitored on that link. Personal interests and search habits can be easily extracted from the packets originating from the users host.

Another threat is the well known DOS (Denial Of Service) attack, or its bigger brother the DDOS (Distributed DOS) attack. DOS attacks are usually aimed at one host, though,

through the GNUTELLA protocol, a whole network can be crippled. First of all, the bandwidth of the network connecting the GNUTELLA hosts can be flooded quite easily by sending tons of GNUTELLA messages. As described earlier, all messages are propagated through the network and, if appropriate, retransmitted by the receiving hosts. So by sending an enormous amount of messages to a GNUTELLA net, the bandwidth can be filled up easily, as normal operation already takes up a large amount of bandwidth.

Another type of (D)DOS attack is aimed at the slower hosts on a GNUTELLA net. By sending a lot of random search queries, connected hosts can loose substantial amounts of time trying to reply those queries. Even during normal operation slower hosts are reported to drop most of the search queries due to limited system resources.

A different type of direct threat is the modification of messages in transmit. This has already been seen on GNUTELLA net as SPAM. Since packets have a very flexible structure and no encryption or signing at all, they can be modified at every hop. Attackers have been reported to add random messages to search query replies and ping-reply messages.

Using IP spoofing it is possible to disconnect a client from GNUTELLA net. By persistantly connecting a node (eg. once every second) using bogus connections, thus offering no files to download and no message forwarding. A bogus connection can consist of one or several (owned or compromised) hosts with modified clients. These clients do not forward messages and do not offer any files, thus blocking all GNUTELLA traffic. With the node accepting the new (bogus) connections in rapid succession, it aquires no new hosts because its ping messages don't get forwarded. This node can become separated from the net, since the bogus connections keep filling up its peer-group while its group of known GNUTELLA hosts is shrinking quickly.

Finally, depending on the quality of the client, bugs could present various other problems. Security breaches have been reported where the whole root filesystem tree was exported by a faulty client.

For an exploration on effectively implementing a GNUTELLA-like protocol (also named Gossip protocol) for productive purposes, see the appendix. This is not a part of the GNUTELLA analysis.

7 Other Issues

7.1 Network Traffic

The enormous amount of message traffic is, as already noted, the most serious problem of the GNUTELLA protocol. We would like to point out two other probable causes.

First, the repeated queries are problematic. A lot of queries are issued multiple times by either the same or different hosts. It might be wise to store search-query replies in all nodes they are routed through. There should be a small timeout on this information because of the potential rapidly changing structure of the GNUTELLA net. Instead of forwarding the search query, a node can look it up in its search-reply cache and answer

the query.

Second, there are a lot of duplicate messages on the net. This is due to the currently high interconnectedness of the participating hosts. This means a lot of messages arrive at any given hosts via different paths. Meaning that each host has to discard several messages which it already received. All these messages are duplicates and generate unnecessary network traffic. In some situations more than 80% of all incoming messages are discarded because they have already been processed. The major problem here is that there is no way of knowing if a node in your peer-group already received the message, unless it sent it to you in the first place.

7.2 Locating the First GNUTELLA Host

To get a client running, it is important to have one or more initial hosts. These hosts provide the portal to finding other hosts and forwarding the GNUTELLA messages. Since most of the nodes in a GNUTELLA net are usually only up for several hours a day, there is no one reliable way to connect to a running GNUTELLA network. Currently most clients tackle this problem by keeping a large lists of all hosts they've ever heard of, hoping that at least one of them will still be running next time this client is used. This is no reliable method to bootstrap a GNUTELLA network though.

Scripts have been proposed to search random IP addresses trying to find a working GNUTELLA port. Since the number of hosts on the internet *not* running a GNUTELLA client by far exceeds the ammount of hosts running a client, this approach is not a realistic option.

8 Conclusion

This report has reviewed several of the currently popular peer-to-peer (P2P) file sharing protocols. Pointers to additional reference material and details have been included. Legal and law enforcement issues have been briefly reviewed as well.

Our detailed analysis of the GNUTELLA protocol has lead us to conclude that GNUTELLA seems to be a viable protocol for informal file sharing across the internet. There is no security, anonymity or reliability imbedded in the current versions of the protocol. It suffers from a lot of free-riding (users which only download, but offer no files themselves) and spamming (users replying on every search query with an inappropriate response.) The biggest problem right now seems to be the enormous amount of network traffic involved. Network traffic scales linearly with the numbers of nodes connected, but is already very high for very low numbers of nodes. Measurements have shown typically 300 bytes/sec per connected host, for just routed messages. So this excludes file transfers. This makes GNUTELLA an option for users who would like to share files on the internet, not minding the shortcomings noted above. GNUTELLA is, in its current from, not suited for a production environment where both reliability and security are required. Most reliability problems are a direct result of bandwidth problems and the rapid dynamic changing of peer-groups.

A Requirements for Using P2P Protocols in a Production Environment

This section deals with the possible use of GNUTELLA like Gossip protocols in production environments.

A.1 General

The strongest and most important feature of peer-to-peer protocols is that each connected node is both client and server. This is useful in environments where no one server can be guaranteed to exist or be reliable. Reasons for this to be the case include constant changing of network topology (mobile agents) or explicitly not wanting one strategically weak point in the network (military applications.)

Besides this, two other things are important. First, reliability. The network has to be reliable to ensure messages reach all hosts. Second, security. The messages are not to be read/modified by anyone other than sender and receiver. Furthermore, security means that the nodes in the network are known. No 'strangers' can infiltrate the communications network.

So what we have is a group of nodes, each with a means of unique identification. Ideally, every node knows of the (possible) existence of every other node. This can be done by enforcing a specific mathematical verification on the identification. Only valid hosts can pass the test.

To ensure message integrity, only the sender can modify the message and only the receiver can read the message. This point is discussed later on. GNUTELLA type peer-groups should be used to reduce traffic.

A.2 Layered Protocol

The peer-group is formed by the lower layer, while the actual communication and forwarding is done by the upper layer. The lower layer is the hardware layer physically making connections with other nodes (eg. mobile agents working out a proper network topology) and reporting a group of closest or best connected peers to the upper layer. Ideally these are all the nodes in the network. Furthermore, the lower layer identifies and verifies all other nodes to which it is connected. This is the first authentication and ensures no unwanted guests can infiltrate the network.

The upper layer provides the communication and forwarding. The peer-group was provided by the lower layer and the upper layer has to take care of the routing. In this layer, messages get encrypted and signed, preferably using a public/private keypair system to protect messages in transmit.

A.3 Routing and Network Traffic

GNUTELLA and other P2P protocols are known to generate enormous amounts of network traffic. Fortunately the number of messages scales linearly with the number of connected hosts.

Several ideas have been proposed to keep the traffic under control. Mark Hayden and Ken Birman proposed the model of Probablistic Broadcast, which, instead of forwarding all incoming messages to the every node in the peer group, forwards messages with some probability. A message gets forwarded depending on a probability P. Ideally, this P depends on:

- Number of participating nodes.
- Size of peergroup.
- Connectedness of the network.

Thus, some sort of metric function must be defined which, for each case, doesn't flood the network with packets, or starves nodes from information.

Another possible solution is to include sender information in every packet. When a node sends a packet to another node, it includes all the other hosts to which it has send that same packet. The receiving hosts then knows which hosts at least received the packet already. It will than add its own list to the forwarded packets.

This works as follows. The originator of the message creates the packet, adds the list of hosts to which the packet will be send, signs it and encrypts it. The receiving peer-group, will decode the packet, verify the encryption and digital signature and create a list of hosts to which they are going to forward the packet. They subtract the hosts which already received it and add this list. The packet is signed and encrypted and the whole process starts all over again, until the TTL expires.

This system only reduces network traffic if this list doesn't exceed the amount of data (payload), which is send on.

A.4 Note on key-pairs and digital signatures

When the sender transmits a message, it can be encrypted using the private key of the sender. A digital certificate may then be added. This requires all hosts participatong in the network to know each other, so each host can act as a certficate authority for every transaction. This shouldn't be necessary if the lower layer verifies the identity, though. To ensure that only the recipent can read the message, the already encrypted message should be encrypted again with the public key of the recipent. That way only the recipent can decode the message. The decoded message is still encrypted with the private key of the sender, so the recipent should use the public key of the sender to decode the message.

B Technical Description of the GNUTELLA Protocol

This description conforms to the reverse-engineerd protocol description as proposed by "Cap'n Bry".

All GNUTELLA connections are passed on through TCP/IP STREAM socket connections. Data is thus guaranteed to arrive. All messages are routed. This means that a reply on a message always returns through the same path as the request was sent. All messages have a unique message ID number which is stored, at every node, and used to check if the message is new or already processed and should thus be discarded. All new messages get passed on to the nodes with which there is a direct connection. Each message gets an initial TTL, which is decremented each hop, thus every time a node passes it on to its direct connections. A message gets discarded when the TTL reaches 0. File transfers are done through a direct HTTP connection between the sending and receiving node. If the sender resides behind a firewall and the receiver is thus not allowed to connect to the sender, the receiver sends a PUSH request to the sender. The sender then connects to the receiver and uploads the requested file. A PING request is always replied to. A search request is executed when the receiving node has at least the specified minimum connection speed. A reply is sent back only when a match is found.

The connection is opened by sending a packet containing only:

```
GNUTELLA CONNECT/0.4\n\n
```

The other side answers by sending a packet containing:

```
GNUTELLA OK\n\n
```

HTTP GET request:

```
GET /get/[File Index Number]/[File Name] HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
\r\n
```

The [File Index Number] is the file index number as returned by a query response packet. The [File Name] is the full name of the file as returned by the query response packet. The server will respond with normal HTTP headers.

Byte Position	*Name*	*Description*
0-15	Message ID	Unique message ID. Pick one
16	Function ID	Type of message, see below
17	TTL remaining	Maximum hops left for this packet
18	Hops taken	Total hops already taken
		TTL of response messages are set to this
19-22	Data Length	Size in bytes of the message payload

Table 3: GNUTELLA message header

Function Code	*Description*
0x00	Ping. Datalen=0. Request 0x01 from every node reached
0x01	Ping reply. Replier supplies IP, port, numer of files and total bytes
0x40	Client push request. For servers behind a firewall. Client cannot reach it, server is requested to connect to client instead
0x80	Search. Payload contains querry
0x81	Search results. List of indexed search results

Table 4: GNUTELLA function ID

The following formats are all preceded by the GNUTELLA message header, since they are all routed messages.

Byte Position	*Name*	*Description*
23-24	Host port	TCP port number of listening host
25-28	Host IP	IP address of listening host, network byte order
29-32	File Count	Total amount of shared files
33-36	Total Size	Total amount of KB shared

Table 5: GNUTELLA ping response

Byte Position	*Name*	*Description*
23-24	Minimum speed	Minimum connection speed for servers which should perform the search
25+	Search query	NULL terminated character string

Table 6: GNUTELLA query header

Byte Position	*Name*	*Description*
23	Num records	Number of GNUTELLA query response RECORDS which follow this headera
24-25	Host port	Listening port number of host returning this query response packet
26-29	Host IP	IP of this host
30-33	Host speed	Speed of this host
34+	Results	Array of GNUTELLA query response RECORDS containing the search results
Last 16	Footer	GNUTELLA query response FOOTER is the clientID128 of this host, an unique number.

Table 7: GNUTELLA query response header

Byte Position	*Name*	*Description*
+0	File index	Index number of file
+4	File size	File size in bytes
+8	File name	Name of file, without path information. Double NULL terminated.

Table 8: GNUTELLA query response RECORD

Byte Position	*Name*	*Description*
23-38	ClientID128	ClientID128 of server the client requests the push from. Obtained from query response.
39-42	File Index	Index of requested file, see query response.
43-46	Requester IP	IP of client requesting the file. (Network order)
47-48	Requester port	Port of requesting client.

Table 9: GNUTELLA push request

C Bibliography

- www.gnutella.wego.com
- freenet.sourceforge.net
- Mark Hayden, Ken Birman. *Probablistic Broadcast*, Department of Computer Science, Cornell University, 1995