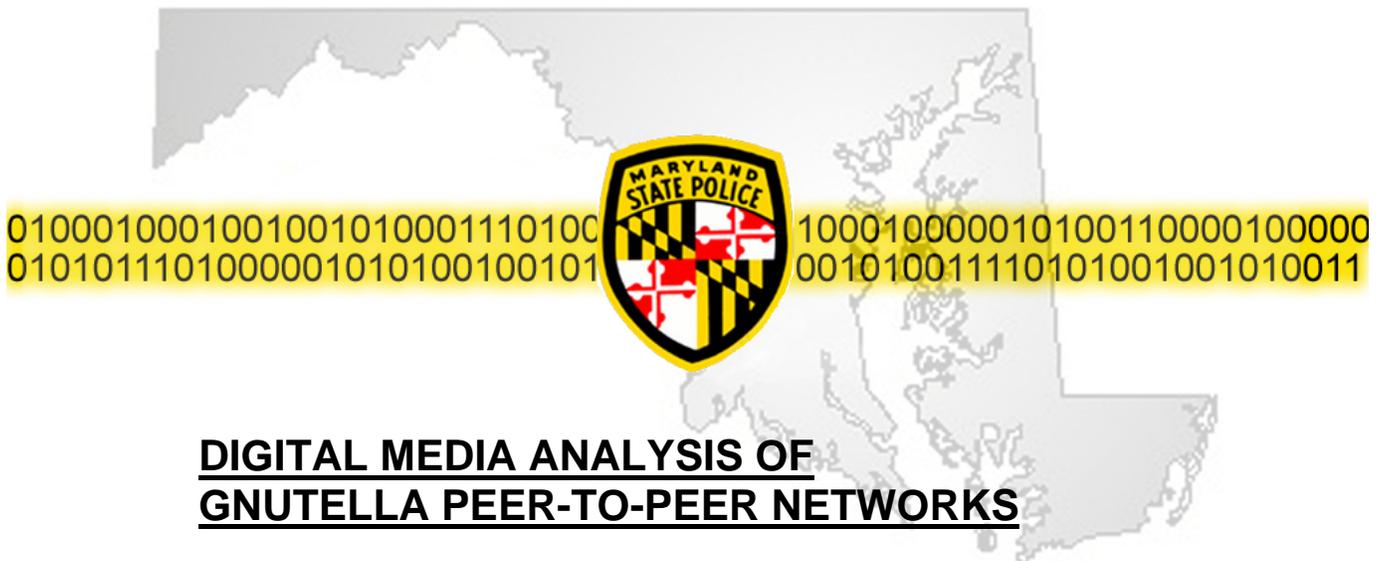


MARYLAND STATE POLICE
COMPUTER FORENSICS LABORATORY

MDSP-CFL₂₀₀₆



LIMEWIRE CASE STUDY

David B. Heslep
Detective Sergeant

Law Enforcement Sensitive

DO NOT DISSEMINATE WITHOUT THE EXPRESS PERMISSION OF THE AUTHOR
Copyright © 2006, David B. Heslep, All Rights Reserved

Contents

TITLE - Digital Media Analysis – Gnutella Peer-to-Peer Networks: Limewire	3
Introduction	3
Intended Audience	3
Goals and Objectives	3
The Gnutella Protocol	4
Gnutella Descriptors	4
Handshaking	5
Gnutella Handshake	5
Ultrappeer Handshaking.....	6
Examples of interactions between Leaves and Ultrapeeers may include the following:.....	6
Standard Message Architecture	7
Payloads (Information Contained in the Different Types of Messages).....	7
Ping.....	7
Pong.....	8
Query	8
Query Hit.....	8
Query Hit Result Item	9
Push.....	9
Normal File Transfer (HTTP 1.1 GET Command)	10
Partial File Transfer (Parallel Downloads) (Swarming).....	11
Examples:.....	12
PUSH Messages	13
LimeWire Design	14
LimeWire Anti-Spam Technology	14
Limewire Installation.....	15
Other Gnutella Clients	18
Analysis Methodology	19
Identify open ports at the time of seizure.....	20
How to determine if Limewire installed on the computer	21
Are the Limewire program’s properties set to allow file sharing?.....	22
Identify the File Name, MD5 hash value, and the SHA1 value for a known file.....	23
Was the file uploaded by another servent?	25
Keyword Searches	26
r@ygold Search Hits.....	26
Other Keywords to Consider.....	26
How to compare SHA1 values recovered from the target computer with known files.....	26
Appendix	27
Known HTTP Connection Headers	27
Known HTTP Download Headers.....	27
Ultrappeer Election Principles	28
Definitions.....	29
File Sharing Networks	29
HASH Algorithms	30
Internet Protocols.....	31
Child Pornography Search Terms (http://www.urbandictionary.com).....	31

TITLE - Digital Media Analysis – Gnutella Peer-to-Peer Networks: Limewire

Abstract – Peer-to-peer networks are used for file sharing between users of these networks. Peer-to-peer networks provide the capability to share any type of file and appear, at least to the user, to offer a certain amount of anonymity. The result is the use of these networks for a variety of illegal activities, including the sharing and distribution of Child Pornography. Our experiences in the examination of digital evidence would suggest that when computers communicate and exchange data, artifacts of those interactions are left behind. This document attempts to explore the use of digital forensics to recover those artifacts and demonstrate with reasonable certainty that these interactions did indeed occur.

Introduction

Computer systems running Limewire or other Gnutella based peer-to-peer file sharing programs leave artifacts that can be demonstrated through digital media analysis, providing evidence of illegal file sharing on these networks. This document provides a basic explanation of the Gnutella network protocol; explains how programs such as Limewire utilize this protocol in conjunction with HTTP1.1 and TCP/IP protocols to share files over the network; and provides an overview of the artifacts that may be left behind on the digital media. Analysis methods for locating, interpreting and documenting this usage are also discussed. By using the model developed here and following the Limewire example, the investigator should be able to extend this experience to other Gnutella GUI programs and to a lesser extent other file sharing networks. Many of the illustrations shown below are from a Child Pornography Distribution criminal case that was generated, as the result of an out of state, ICAC Peer Precision investigation.

Intended Audience

Information in this paper is Law Enforcement Sensitive. It is intended for trained digital forensics practitioners who have a firm understanding of data structures, the use of hash analysis, and digital forensics principles. As it includes a study case involving child pornography, readers are forewarned that this document ***contains sexually explicit text that may be offensive.***

Goals and Objectives

Given a target file, the digital forensics practitioner should be able to identify or determine the File Name, MD5 hash value, and the SHA1 value, as well as answer the following questions:

1. Is a Gnutella peer-to-peer file sharing program installed?
2. Are the program's properties set to allow file sharing?
3. Was the target file downloaded utilizing a Gnutella file-sharing program? When?
4. Was a particular file uploaded utilizing the Gnutella file-sharing program? When? By Whom?

This document will examine the Gnutella Network Protocol and the popular Gnutella file-sharing program, *Limewire*, installed in a Windows™ XP environment and attempt to answer the questions posed above. Finally, suggestions for the analysis of digital evidence that was utilized to participate in sharing of contraband files over a Gnutella network will be proposed.

The Gnutella Protocol¹

Excerpts from the **Gnutella Development Forum**, Gnutella/0.6
online at <http://www.the-gdf.org>
Copyright (C) 2002, Tor Klingberg & Raphael Manfredi, All Rights Reserved.
CREATIVE COMMONS PUBLIC LICENSE

The Gnutella Network (*GNet*) consists of interconnected host computers implementing the Gnutella protocol. According to the Gnutella Protocol Specification, Gnutella is a network protocol for distributed **searches** that utilizes a *peer-to-peer* decentralized model. In this model, each computer is commonly referred to as a *Gnutella Servent* (a contraction of *SER*ver and *cli*ENT) and serves as both a client and a server. The *client-side* interface provides the user with the ability to make queries, view search results and download files. The *server-side* interface accepts and responds to queries from other servents concerning locally stored files and permits the uploading of files when they are available.

In Gnutella Protocol, version 0.6, the concept of *Ultrapeers* was introduced to add more structure and hence, efficiency to the network. The Ultrapeer system categorizes the servents, attached to the network, as "*Leaves*" or "*Ultrapeers*" based on a number of factors, including the servent's capabilities and the network's need for additional Ultrapeers. Leaves connect to three Ultrapeers on average. Ultrapeers can connect to as many as thirty-two other Ultrapeers and will maintain about thirty leaves. All servents on the network, act as servers and share in maintaining the network, however, those servents designated as Ultrapeers bare responsibility for message handling and routing, thereby reducing traffic between the Leaves on the network. There are no central servers such as those used by Napster and therefore, no single point of failure or owner of all of the client data. Each servent, while it is attached to the network, cooperates with all other servents to maintain the network. For the network to be effective, **each servent should contribute to the network by sharing its files**. The Gnutella network topology makes it extremely redundant and very difficult to regulate.

The Gnutella Protocol consists of a set of descriptors used for communicating data between servents and a set of rules governing the inter-servent exchange of descriptors. Gnutella compliant software or clients, such as Limewire, provide an interface through which the user can become part of the network. Data exchanges between nodes are negotiated using TCP/IP and the standard HTTP protocols. Currently, the following descriptors are defined in the Gnutella protocol:

Gnutella Descriptors

	Descriptor	Description
0x00	Ping	Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.
0x01	Pong	The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.
0x02	Bye	Used to terminate a connection
0x80	Query	The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.
0x81	Query Hit	The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.
0x40	Push	A mechanism that allows a firewalled servent to contribute file-based data to the network.

Handshaking

A Gnutella servent connects itself to the network by establishing a connection with another servent currently on the network. Once the first connection is established, the addresses of more hosts will be supplied over the network. The default **Gnutella port is 6346**, but servents MAY use any unused port. If the desired port is used (probably by another Gnutella servent) the servent attempts to listen on another port. This listening port is advertised by the servent through the Pong messages. Once the address of another servent on the network is obtained, a TCP/IP connection to the servent is created, and a handshaking sequence is initiated. The client is the host initiating the connection and the server is the host receiving it. "<cr>" refers to ASCII character 13 (carriage return), and "<lf>" to 10 (new line). Figure 1 shows a typical handshake.

Gnutella Handshake

1. The client establishes a TCP connection with the server.
2. The client sends "**GNUTELLA CONNECT/0.6<cr><lf>**".
3. The client sends all capability headers--except for vendor-specific headers--each terminated by "<cr><lf>", with an extra "<cr><lf>" at the end.
4. The server responds with "**GNUTELLA/0.6 200 <string><cr><lf>**". <string> SHOULD be "OK"
5. The server sends all its headers, in the same format as (3).
6. The client sends "GNUTELLA/0.6 200 OK<cr><lf>", as in (4) if after parsing the server's headers, it still wishes to connect. Otherwise, it needs to reply with an error code and close the connection.
7. The client sends any vendor-specific headers as needed, in the same format as (3).
8. Both client and server send binary messages at will, using the information gained in (3) and (5).

Figure 1

Figure 2 demonstrates a sample interaction between a client and a server. Data sent from client to server is shown on the left; data sent from server to client is shown on the right.

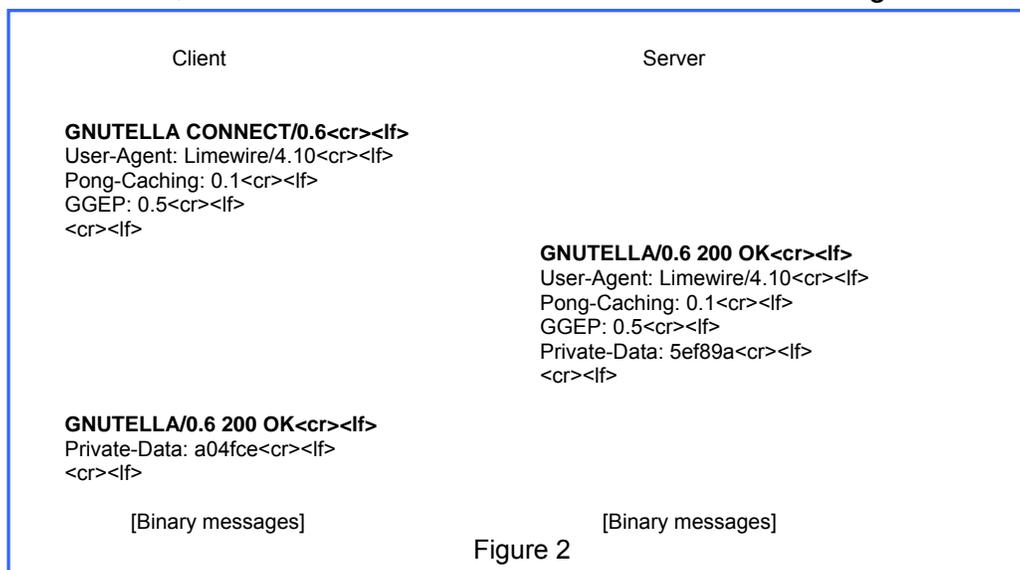


Figure 2

A few notes about the responses: first, the client (server) should disconnect if receiving any response other than "200" at step 4 (6). Second, servents should ignore higher version numbers in steps (2), (4), and (6). For example, it is legal for a future client to connect to a server and send "GNUTELLA CONNECT/0.7". The server SHOULD respond with "GNUTELLA/0.7 200 OK" if it supports the 0.7 protocols or "GNUTELLA/0.6 200 OK" otherwise.

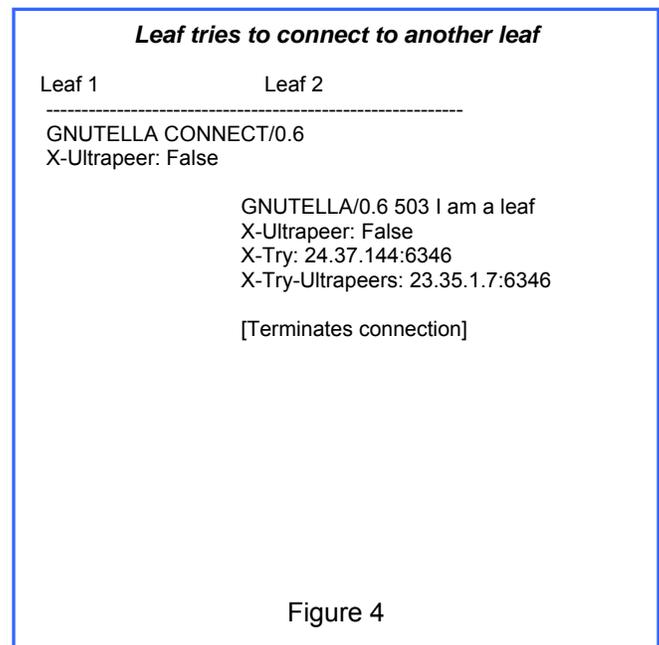
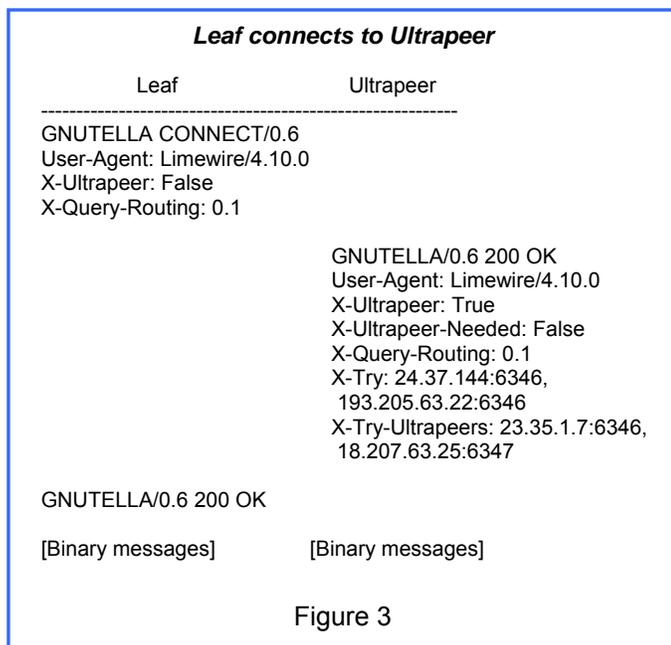
Ultrapeer Handshaking

Ultrapeer capabilities and information is exchanged during the handshaking sequence when trying to establish a new Gnutella connection. The following new headers are used:

- **X-Ultrapeer:** "True" signals that node is an Ultrapeer; "False" signals that the node wants to be a shielded leaf node.
- **X-Ultrapeer-Needed:** Used to balance the number of Ultrapeers
- **X-Try-Ultrapeers:** Like X-Try (see 2.1), but contains only addresses of Ultrapeers.
- **X-Query-Routing:** Signals support for the Query Routing Protocol

Examples of interactions between Leaves and Ultrapeers may include the following:

- A leaf connects to an Ultrapeer. The leaf is now a shielded node of the Ultrapeer. The leaf should drop any non-Ultrapeer connections. If a shielded leaf node receives a connection request, it will refuse to accept the connection by returning a 503-error code together with X-Try and X-Try-Ultrapeer headers to redirect to remote host to other addresses. Figure 3
- A leaf may try to connect to another leaf, in which case the connection will be refused. Figure 4
- Sometimes nodes will be Ultrapeer-incapable but unable to find an Ultrapeer. In this case, they behave exactly like old, un-routed Gnutella 0.4 connections.
- When two Ultrapeers meet, both set X-Ultrapeer: true. If both have leaf nodes, they will remain Ultrapeers after the interaction.
- Sometimes there will be too many Ultrapeer-capable nodes on the network. Consider the case of an Ultrapeer "A" connecting to an Ultrapeer "B". If B doesn't have enough leaves, it may direct A to become a leaf node. If A has no leaf connections, it stops fetching new connections, drops any Gnutella 0.4 connections, and sends a QRP table to B. Then B will shield A from all traffic. If A has leaf connections, it ignores the guidance, as in the above case.



Standard Message Architecture

Once a servent has connected successfully to the network, it communicates with other servents by sending and receiving Gnutella protocol descriptors. **Each descriptor is preceded by a Descriptor Header with the byte structure given below.**

Message Header - The message header is 23 bytes divided into the following fields.

Bytes	Description
0-15	Message ID/GUID (Globally Unique ID)
16	Payload Type
17	TTL (Time To Live)
18	Hops
19-22	Payload Length

Message ID – A 16-byte string (GUID) uniquely identifying the **message** on the network. This is a security measure to ensure that responses are only sent to legitimate requestors and that responses are only accepted after a legitimate request has been made. This protects servents from receiving data and files that were not requested.

Payload Type – Indicates the type of message. Gnutella servents MUST accept all the following types:

Type	Message
0x00	Ping
0x01	Pong
0x02	Bye
0x40	Push
0x80	Query
0x81	Query Hit

TTL (Time-to-Live) – The number of times the message will be forwarded by Gnutella servents before it is removed from the network. Each servent will decrement the TTL before passing it on to another servent. When the TTL reaches zero, the message will no longer be forwarded.

Hops – The number of times the message has been forwarded. As a message is passed from servent to servent, the TTL and Hops fields of the header must satisfy the following condition: $TTL(0) = TTL(i) + Hops(i)$ where $TTL(i)$ and $Hops(i)$ are the value of the TTL and Hops fields of the message, and $TTL(0)$ is maximum number of hops a message will travel (usually 7).

Payload Length – The length of the message immediately following this header

Payloads (Information Contained in the Different Types of Messages)

Ping (0x00) – Ping messages MAY contain a GGEP extension block, but no other payload.

Pong (0x01) – Pong messages contains information about a Gnutella host. The message has the following payload:

Bytes	Field name	Description
0-1	Port Number	The port number on which the responding host can accept incoming connections.
2-5	IP Address	The IP address of the responding host. Note: This field is in big-endian format.
6-9	Number of shared files	The number of files that the servent with the given IP address and port is sharing on the network.
10-13	Number of kilobytes shared	The number of kilobytes of data that the servent with the given IP address and port is sharing on the network.
14-	GGEP block	OPTIONAL <i>extension (Gnutella Generic Extension Protocol (GGEP) allows arbitrary extensions in Gnutella messages).</i>

Pong messages are only sent in response to an incoming Ping message. It is common for a servent to send all recently received Pongs in response to every single Ping message. This enables host caches to send cached servent address information in response to a Ping request.

*The **Message ID** of a Pong message **MUST** be the Message ID of the Ping message it is sent in reply to.*

*The fields specifying the **number of shared files** and the **number of kilobytes shared** were intended to allow one to measure the amount of data available on the network. With a very large Gnutella network, and minimized Ping and Pong message traffic, this can no longer be done. Still, these fields **SHOULD** be filled out correctly.*

Query (0x80) – A Query message has the following payload:

Bytes	Field name	Description
0-1	Minimum Speed (Flags)	The minimum speed (in kb/second) of servents that should respond to this message. A servent receiving a Query message with a Minimum Speed field of n kb/s SHOULD only respond with a Query Hit if it is able to communicate at a speed $\geq n$ kb/s.
2-	Search Criteria	This field is terminated by a NUL (0x00). The Search Criteria is a string of keywords. A servent SHOULD only respond with files that have all the keywords. A space is the standard separator between words. See Gnutella Protocol Section 2.2.7.3 for rules and information on how to Interpret the Search Criteria
Rest	Extensions Block	The rest of the query message is used for extensions to the original query format. The allowed extension types are GGEP, HUGE and XML. The type of each block can be determined by looking for the prefixes "urn:" for a HUGE block, "<" or "!" for XML and 0xC3 for GGEP.

Query Hit (0x81) – Query Hit messages has the following fields:

Bytes	Field name	Description
0	Number of Hits	The number of query hits in the result set (see below).
1-2	Port	The port number on which the responding host can accept an incoming HTTP file request. This is usually the same port used for Gnutella network traffic, but any port MAY be used.
3-6	IP Address	The IP address of the responding host. Note: This field is in big-endian format.
7-10	Speed	The speed (in kb/second) of the responding host.
11-	Result Set	A set of responses to the corresponding Query. This set contains <i>Number_of_Hits</i> elements.
x	Extended QHD	This block is not strictly required, but strongly recommended. It is sometimes called EQHD, or (incorrectly) just QHD.
x	Private Data	Undocumented vendor-specific data. This field continues until the servent Identifier, which uses the last 16 bytes of the message.
Last 16	Servent Identifier	A 16-byte string uniquely identifying the responding servent on the network. This SHOULD be constant for all Query Hit messages emitted by a servent and is typically some function of the servent's network address. The servent Identifier is mainly used for routing the Push Message.

Query Hit Result Item – Each item contained in the query-hit result is structured as follows:

Bytes	Field name	Description
0-3	File Index	A number, assigned by the responding host, which is used to uniquely identify the file matching the corresponding query.
4-7	File Size	The size (in bytes) of the file whose index is " File Index ". For large files whose size cannot be expressed with a 32-bit integer, a GGEP LF block can be used in the extensions block.
8-	File Name	The name of the file whose index is " File Index ". Terminated by a null byte (i.e. 0x00).
x	Extensions block.	<p>Allowed extension types are HUGE, GGEP and plain text metadata. This field is terminated by a null (0x00), even if there are no extensions (resulting in a double null). Also, the extensions block itself MUST NOT contain any null bytes.</p> <p>If two or more of these extension types exist together, they are separated by a 0x1C (file separator) byte. Since GGEP blocks can contain 0x1C bytes, the GGEP block, if present, MUST be located after any HUGE and plain text blocks.</p> <p>The type of each block can be determined by looking for the prefixes "urn:" for a HUGE block, 0xC3 for GGEP and anything else is probably plain text metadata.</p> <p>Plain text metadata is intended to be displayed directly to the user. It was first invented by Gnutella (a now discontinued Gnutella server) to tag MP3 files.</p> <p>Examples: "192 kbps 44 kHz 3:23" "120 kbps(VBR) 44kHz 3:55" (variable bitrate) Other plain text formats MAY be used.</p>

Push (0x40) – A Push message has the following fields:

Bytes	Field name	Description
0-15	Server Identifier.	The 16-byte string uniquely identifying the server on the network who is being requested to push the file with index File Index. The server initiating the push request MUST set this field to the Server_Identifier returned in the corresponding QueryHit message. This is used to route the Push message to the sender of the Query Hit message.
16-19	File Index	The index uniquely identifying the file to be pushed from the target server. The server initiating the push request MUST set this field to the value of one of the File Index fields from the Result Set in the corresponding QueryHit message.
20-23	IP Address	The IP address of the host to which the file with File Index should be pushed. This field is in big-endian format.
24-25	Port	The port number the receiver of this message should push to.
26-	OPTIONAL GGEP extension block	

Normal File Transfer (HTTP 1.1 GET Command)

Once a server receives a QueryHit message, it may initiate the direct download of one of the files described by the message's Result Set. **Files are downloaded out-of-network: i.e., a direct connection between the source and target server is established in order to perform the data transfer. File data is never transferred over the Gnutella network.** The file download protocol is HTTP. It is RECOMMENDED to use HTTP 1.1, but HTTP 1.0 can be used instead.

The server initiating the download sends a request string on the following form to the target server:

```
GET /get/<File Index>/<File Name> HTTP/1.1<cr><lf>
User-Agent: Limewire 4.10.1<cr><lf>
Host: 123.123.123.123:6346<cr><lf>
Connection: Keep-Alive<cr><lf>
Range: bytes=0-<cr><lf>
<cr><lf>
```

Figure 5

Where <File Index> and <File Name> are one of the File Index/File Name pairs from a QueryHit message's Result Set. For example, if the Result set from a QueryHit message contained the entry:

```
File Index: 2468
File Size: 4356789
File Name: Oh I want to be a cowboy.mp3
```

Then a download request for the file described by this entry would be initiated as follows:

```
GET /get/2468/ Oh I want to be a cowboy.mp3 HTTP/1.1<cr><lf>
User-Agent: Limewire 4.10.1<cr><lf>
Host: 123.123.123.123:6346<cr><lf>
Connection: Keep-Alive<cr><lf>
Range: bytes=0-<cr><lf>
<cr><lf>
```

Figure 6

The Host header is required by HTTP 1.1 and specifies what address you have connected to. It is usually not used by the receiving server, but its presence is required by the protocol.

The allowable values of the User-Agent string are defined by the HTTP standard. Server developers cannot make any assumptions about the value here. The use of 'Gnutella' is for illustration purposes only.

The server receiving this download request responds with HTTP 1.1 compliant headers such as

```
HTTP/1.1 200 OK<cr><lf>
Server: Gnutella<cr><lf>
Content-type: application/binary<cr><lf>
Content-length: 4356789<cr><lf>
<cr><lf>
```

Figure 7

The file data then follows and should be read up to and including, the number of bytes specified in the Content-length provided in the server's HTTP response.

Partial File Transfer (Parallel Downloads) (Swarming)

The server allows HTTP requests for partial files, at URIs chosen by the server. They can for example be assigned a file index and shared at `"/get/index/filename"`, or simply at `"/partials/filename"`. If requests by URN are supported, the best way is probably to share only at:

`"uri-res/N2R?urn:sha1:HASH_OF_COMPLETE_FILE"`.

Only partial requests (with a *Range* header) are accepted. The X-Available-Ranges header is used by the server to inform the client about what ranges are available. Note that 2xx or 503 responses without an X-Available-Ranges header means the complete file is available. The format is as follows:

X-Available-Ranges: bytes 0-10,20-30

The client requests the range it wants using the *Range* header. `Range: bytes=0-` means the client wants any ranges the server can provide. The server then provides the range it wants to upload using a **206 Partial Content** response. This allows the server to upload different ranges to different hosts, and save bandwidth by allowing them to get the other parts from each other. The server can decide to upload any range inside the requested range. This means that the client cannot be sure that the first byte in the response is first requested byte. The 206 response contains a Content-Range header on the form:

Content-Range: bytes <start>-<end>/<total_size>

Note that `<total_size>` is the size of the **complete** file.

Tree hashes – TigerTree can be implemented if/when corrupt files become a problem. The reason that it is in this document is that Partial File Sharing might cause corrupt files to spread faster. TigerTree hashes are computed using a 1024 byte base size. The tree is provided as specified in the **Tree Hash EXchange** format. It basically says that the hash tree is provided as a long stream of binary data starting with the root hash, then the two hashes it is computed from, and so on.

To inform the client about where the hash tree can be retrieved the server includes an X-Thex-URI header on this form:

X-Thex-URI: <URI> ; <ROOT>

`<URI>` is any valid URI. It can be to an uri-res translator, and can even point to another host. The client can then retrieve desired parts of the hash tree by doing range requests for the specified URI.

Parallel Downloads – Some Gnutella clients support parallel downloads, downloading parts of a file from several servers at the same time. In this case, the download request and response will include a *range request* in the form of `"Range: bytes=range<cr><lf>"`

Download Request

Range requests are on the form

```
GET /get/2468/Foobar.mp3 HTTP/1.1<cr><lf>
User-Agent: Limewire 4.10.1<cr><lf>
Host: 123.123.123.123:6346<cr><lf>
Connection: Keep-Alive<cr><lf>
Range: bytes=4932766-5066083<cr><lf>
<cr><lf>
```

Figure 8

Note that the Range header does not have to specify both start and end positions.

Response

The response is on the form

```
HTTP/1.1 206 Partial Content<cr><lf>  
Server: Limewire 4.10.1<cr><lf>  
Content-Type: audio/mpeg<cr><lf>  
Content-Length: 133318<cr><lf>  
Content-Range: bytes 4932766-5066083/5332732<cr><lf>  
<cr><lf>
```

Figure 9

Examples:

A simple GNet search query and file download

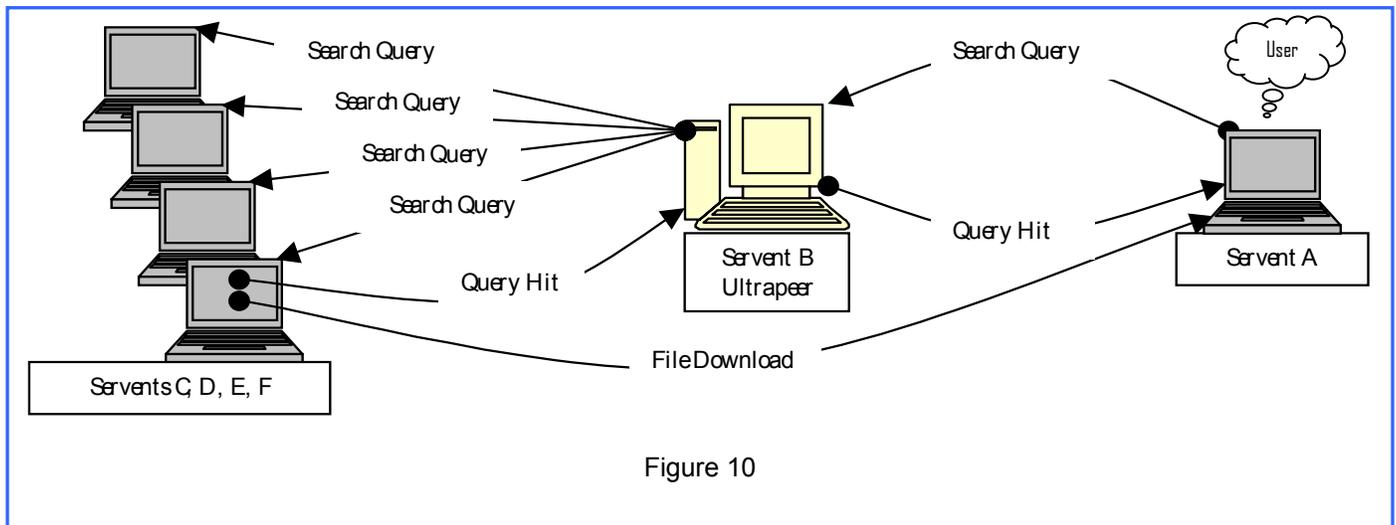


Figure 10

Actual download requests recovered from suspect's computer

```
GET /uri-res/N2R?urn:shal:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF HTTP/1.1  
Host: 24.239.69.187:6348  
User-Agent: iMesh 0.0.0.1 (GnucDNA 1.1.0.8)  
Listen-IP: 168.234.252.14:3039  
Connection: Keep-Alive  
Range: bytes=8469264-8912895  
X-Queue: 0.1  
X-Features: g2/1.0  
X-Alt: 152.30.115.34:6346, 166.82.115.36:6346, 217.44.2.228:6346, 200.95.12.237:3089, 66.41.170.41:6346  
X-NAlt: 83.200.24.228:1602, 24.17.20.14:6346, 208.59.174.6:6346, 210.49.73.251:6346, 82.136.196.127:6346
```

SHA1 Hash of requested file
IP Address/Port for computer providing file
Agent used by requestor
IP Address of requestor

```
GET /uri-res/N2R?urn:shal:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF HTTP/1.1  
Host: 24.239.69.187:6349  
X-Features: g2/1.0  
Range: bytes=17885184-17950719  
User-Agent: Shareaza 1.8.11.2  
Listen-IP: 68.68.79.164:6346  
X-Nick: TTTBone  
X-Queue: 0.1  
X-Content-URN: urn:shal:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF  
Alt-Location: http://24.240.45.175:6346/uri-res/N2R?urn:shal:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-01-31T20:36Z,  
http://205.250.224.239:6346/uri-res/N2R?urn:shal:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-01-31T20:36Z
```

SHA1 Hash of requested file
IP Address/Port for computer providing file
Agent used by requestor
IP Address of requestor
Chat Nickname of requestor

PUSH Messages

Firewalled Servents – It is not always possible to establish a direct connection to a Gnutella servent in an attempt to initiate a file download. The servent may for example, be behind a firewall that does not permit incoming connections to its Gnutella port. If a direct connection cannot be established, the servent attempting the file download may request that the servent sharing the file "push" the file instead. A servent can request a file push by routing a Push request back to the servent that sent the QueryHit message describing the target file. The servent that is the target of the Push request (identified by the Servent Identifier field of the Push message) SHOULD, upon receipt of the Push message, attempt to establish a new TCP/IP connection to the requesting servent (identified by the IP Address and Port fields of the Push message). If this direct connection cannot be established, then it is likely that the servent that issued the Push request is itself behind a firewall. In this case, file transfer cannot take place by the means of what is described in this document.

Usage of Push Messages – A servent may send a Push message if it receives a QueryHit message from a servent that doesn't support incoming connections. This might occur when the servent sending the QueryHit message is behind a firewall. When a servent receives a Push message, it SHOULD act upon the push request if and only if the servent_Identifier field contains the value of its servent identifier. **The Message ID field in the Message Header of the Push message should NOT contain the same value as that of the associated QueryHit message, but SHOULD contain a new value generated by the servent's Message ID generation algorithm.** (This is the GUID described in the *Limewire.props* file.) Push messages are forwarded back to the originator of the Query Hits message using the Servent Identifier value. This means multiple Push messages can have the same Servent Identifier. Push messages MUST only be considered as duplicates if the Message ID in the header is the same. Since Push messages are not broadcasted, duplicate messages should be very rare.

Pushing the file to the Downloader – If a direct connection can be established from the firewalled servent to the servent that initiated the Push request, the firewalled servent should immediately send the following:

`GIV <File Index>:<Servent Identifier>/<File Name><lf><lf>`

Where <File Index> and <Servent Identifier> are the values of the File Index and Servent Identifier fields respectively from the Push request received, and <File Name> is the name of the file in the local file table whose file index number is <File Index>. The File Name MAY be url/uri encoded. The servent that receives the GIV (the servent that wants to receive a file) SHOULD ignore the File Index and File Name, and request the file it wants to download. The servent that sent the GIV MUST allow the client to request any file, and not just the one specified in the Push message. The GET request and the remainder of the file download process is identical to that described in the section 4.1 (Normal File Transfer) above.

The <Servent Identifier> is formatted as hexadecimal, and must be read case-insensitively. For instance:

`GIV 36:809BC12168A1852CFF5D7A785833F600/Foo.txt<lf><lf>`
`GIV 124:d51dff817f895598ff0065537c09d503/Bar.html<lf><lf>`

LimeWire Design² Version 4.10.9

LimeWire is a free and open source peer-to-peer file sharing client for the Gnutella network. It is released under the GNU General Public License. The program allows users to share files using the Gnutella peer-to-peer protocol. It is written in SUN Java and therefore runs on any computer with the Java virtual machine installed. To facilitate installation for casual users, the developers release installation packages for Microsoft Windows, Mac OS X, and for Linux, in RPM format.

LimeWire uses the SHA-1(base32) and Tiger tree hash cryptographically secure hash functions to ensure that downloaded data is uncompromised. Although researchers have identified possible vulnerabilities in the SHA1 algorithm, because LimeWire does not rely on SHA1 alone, these vulnerabilities do not have many adverse implications for LimeWire's verification of downloaded files. Limewire's official website is located at <http://www.limewire.com/>.

LimeWire Anti-Spam Technology

There are three well-known techniques for eliminating Gnutella spam:

- **IP Filtering:** ignore packets from certain hosts. (pongs and query replies) Also deny incoming connections from these hosts.
- **Keyword filtering:** ignore packets containing certain undesirable words, e.g., "xxx" or ".vbs". (queries and query replies)
- **Duplicate filtering:** ignore duplicate packets, or very similar packets, that come within any given time frame. (All packets, though this currently is only implemented for queries and pings).

These techniques can be applied on two different levels:

1. **Personal filtering:** ignore what I see in the search monitor and results window, but pass them on to other users. Currently if any file in a query reply is considered spam, all the files are considered spam. This could change in the future.
2. **Route filtering:** ignore spam completely. Do not pass on to other users. Do not display.

LimeWire currently supports the following policies:

- IP filtering on the route level
- Keyword filtering on the personal level. Rationale: don't want to censor what others can see.
- Duplicate filtering on the route level.
- Adult filtering: this is just a simple way of setting up keyword filtering on the personal level without having to type in dirty words.
- VBS filtering: similar to adult filtering, but with "vbs" (Visual Basic script extension) entered instead.
- HTML filtering: similar to adult filtering, but with "htm" entered instead. Rationale: most HTML files on GNet are spam right now. Hopefully this will change.
- Greedy query filtering: a kind of keyword filtering technique looking for queries that are bound to have lots of results, e.g., a.mp3. Applied on the route level.

Limewire Installation³

THE INSTALLATION WIZARD

NOTE: Screenshots are taken from a Windows installation, English version. The installation process depends on whether or not you already have the Java Runtime Environment installed on your machine. For specific problems with installation, check the FAQ document.



Step 1: The install window will open. Click "Next" to initiate installation.



Step 2: You will be asked to choose a folder to install LimeWire. You may choose the default folder or install in a different folder. Afterwards, you will be asked to select where to place the shortcut/alias for LimeWire. We recommend placing the LimeWire shortcut on your desktop.



Step 3: LimeWire will install on your computer.



Step 4: Select "Finish" or click on the shortcut on your desktop to launch LimeWire.

THE SETUP WIZARD



Step 1: Select your preferred language for LimeWire.



Step 2: Follow and complete The Setup Wizard.

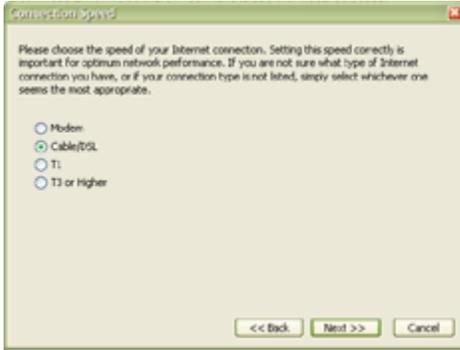


Step 3: You will be asked to designate a folder that will contain the files you would like to share. With the "Browse Host" feature, a user can look at any file in this folder. You can change your folder location at any time by going to Tools/Options/Saving. Please be aware that private material can be viewed if it is in your shared folder.

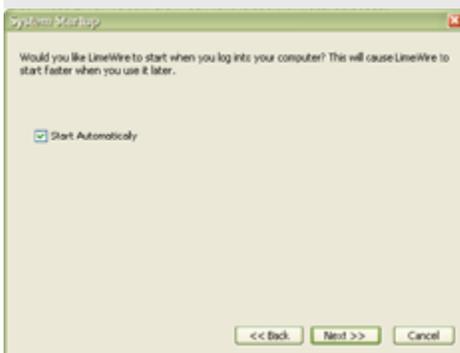
You can share more than one folder by adding additional folders to your Shared list in Library. **WARNING: DO NOT SHARE YOUR ENTIRE HARD DRIVE.** If you share your entire hard drive, the contents of your whole computer may be exposed.

To limit sharing to specific file types, enter that file extension in the Shared Extensions input field (Tools> Options> Sharing). For example, to share only JPEG files enter only "jpg" or "jpeg" in the Shared Extensions input field.

If the Share Finished Downloads option is checked, then all completed downloads will be shared. To share or unshare individual files-- independent of their folder-- go to the Library and select a specific folder from the folder list in the left window, and right-click for sharing options.



Step 4: Choose your connection speed. If you do not know your speed, select "Modem" if your connection is generally slow, or "Cable/DSL" if it is generally fast. You can change your connection speed at any time by going to Tools/Options/Speed.



Step 5: Choose whether or not you would like LimeWire to start automatically when your computer starts. It is recommended that you do this for the best LimeWire experience. LimeWire will load into your system tray.



Step 6 & 7: If you have any firewalls or security programs installed on your computer, you may receive a prompt asking if you would like to grant permission to LimeWire. **IMPORTANT:** You must grant permission or unblock access in order for LimeWire to function.



Step 8: You will have the option to allow LimeWire to scan your hard drive for directories you might like to share. If you choose to do this, the media files currently on your computer will be available automatically in your Library.



Step 9: Your settings are complete.

When you complete this setup, LimeWire will be ready to go.

Other Gnutella Clients

Name	Platform	License
Acquisition	Mac OS X	proprietary
Apollon (GUI)	Unix-like/KDE GNU	GPL
BearShare	Microsoft Windows	proprietary
Cabos	Java GNU	GPL
CocoGnut	RISC OS	Freeware
DM2	Microsoft Windows	Freeware
FrostWire	Java GNU	GPL
giFT	Microsoft Windows, Mac OS X, AmigaOS GNU	GPL
Gnucleus	Microsoft Windows GNU	GPL, GNU LGPL
Gtk-gnutella	Unix-like GNU	GPL
Gluz	Java	proprietary
iMesh	Microsoft Windows	proprietary
KCeasy	Microsoft Windows GNU	GPL
Kiwi Alpha	Microsoft Windows	proprietary
LimeWire	Java GNU	GPL
MLdonkey	Microsoft Windows, Mac OS X, MorphOS GNU	GPL
Morpheus	Microsoft Windows	proprietary
Mutella	Unix-like GNU	GPL
Phex	Java GNU	GPL
Poisoned	Mac OS X GNU	GPL
Qtella	Unix-like GNU	GPL
Shareaza	Microsoft Windows GNU	GPL
Swapper.NET	Microsoft Windows	proprietary
Symella	Symbian OS GNU	GPL
XFactor	Mac OS X GNU	GPL
XNap	Java GNU	GPL
XoloX	Microsoft Windows	proprietary

Analysis Methodology

This methodology is based on the analysis of the computer that possesses and shares a target file or file of interest, such as Child Pornography. Gnutella client programs are designed for efficient file sharing; therefore, most only save information that is required for operation of the program. The rest of the information is simply transient on the computer sharing the files. Information saved and the location of information may be different from program to program. What these programs do have in common is the Gnutella network and its dependence on Http1.1 and TCP/IP.

1. **Examine open ports before shutting the computer system down at the time of seizure.** If the computer is powered on and connected to the internet at the time of seizure, use a program such as FPort v2.0 - TCP/IP Process to Port Mapper, Copyright 2000 by Foundstone, Inc. <http://www.foundstone.com> to determine ports that are being utilized by peer to peer programs before seizing the computer. F.R.E.D. v1.3 - 2 September 2004 [modified for HELIX 09/2004], may be ran to collect live system information. The FRED report can be output to one 3.5" floppy disc. This report documents a host of information including system information, network connections, network user information, running processes, hidden files, and open ports. **If the computer is off, do not turn it on.**
2. **Examine the logical file system to determine what peer-to-peer programs are installed.** Keep in mind that the program folders from some programs are different from the name of the program itself (e.g. Etomi/Shareaza, Streamcast/Morpheus Ultra) and not all of the programs are installed in the **Program Files** folder by default. In addition, the install location can be altered during the installation of most of these programs.
3. **Identify files installed by each program that may contain useful data.** Depending on the program these files may be easy to difficult or even impossible to interpret. Some programs such as Morpheus Ultra utilize several Peer-to-Peer protocols and as a result, files such as the Morpheus Ultra\GnuHashes.ini file contain a wealth of information and are easy to interpret.
 - i. Properties File – stores current properties of program
 - ii. Urn Hashes File – stores at minimum the Index, SHA1 of the file, and the File Name with full path.
 - iii. Gwebcache – stores information concerning other servents
4. **Identify the Shared Folder(s).** Users may select more than one folder to share on a network, or may even share their entire hard drive, though this is not recommended.
5. **Identify known files provided by the preceding investigation.** Calculate the file's MD5 and SHA1 Hash Values, and determine its file name. Are these files located in the Shared folder(s), or subfolders created within the shared folder(s)? Are the files categorized? Have files been relocated to non-shared folders. Are file names and hash values located within other files?
6. **Locate Contraband Files** (Child Pornography, etc.). Are these files located in the Shared folder(s), or subfolders created within the shared folder(s)? Are the files categorized? Have files been relocated to non-shared folders. Have the file been uploaded by other servents?
7. **Search the computer**, including unallocated space, slack space and the swap file (pagefile.sys) for remnants of file sharing.
 - i. SHA1 and MD5 Hash Values
 - ii. Known File Names
 - iii. Http1.1 commands (GET)
 - iv. Known Users Names/nicknames
 - v. Known IP Addresses
 - vi. Known GUID strings

Identify open ports at the time of seizure

This computer was running Limewire and WinMX at the time of seizure and was in the process of downloading multiple files. As you can see, both Limewire and WinMX are using multiple ports.

```
=====
FPORT (fport /p)
=====
FPort v2.0 - TCP/IP Process to Port Mapper, Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com
```

Pid	Process	Port	Proto	Path
1032		-> 135	TCP	
4	System	-> 139	TCP	
4	System	-> 445	TCP	
1888	ccApp	-> 1030	TCP	C:\Program Files\Common Files\Symantec Shared\ccApp.exe
3352		-> 1042	TCP	
504	LimeWire	-> 1046	TCP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 1047	TCP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 1289	TCP	C:\Program Files\LimeWire\LimeWire.exe
204	WinMX	-> 1523	TCP	C:\Program Files\WinMX\WinMX.exe
504	LimeWire	-> 5214	TCP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 6350	TCP	C:\Program Files\LimeWire\LimeWire.exe
0	System	-> 6350	TCP	
204	WinMX	-> 6699	TCP	C:\Program Files\WinMX\WinMX.exe
504	LimeWire	-> 45100	TCP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 123	UDP	C:\Program Files\LimeWire\LimeWire.exe
0	System	-> 137	UDP	
504	LimeWire	-> 138	UDP	C:\Program Files\LimeWire\LimeWire.exe
1032		-> 445	UDP	
4	System	-> 500	UDP	
0	System	-> 1025	UDP	
504	LimeWire	-> 1027	UDP	C:\Program Files\LimeWire\LimeWire.exe
204	WinMX	-> 1029	UDP	C:\Program Files\WinMX\WinMX.exe
504	LimeWire	-> 1039	UDP	C:\Program Files\LimeWire\LimeWire.exe
4	System	-> 1093	UDP	
504	LimeWire	-> 1094	UDP	C:\Program Files\LimeWire\LimeWire.exe
204	WinMX	-> 1095	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 1096	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 1097	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 1098	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 1099	UDP	C:\Program Files\WinMX\WinMX.exe
504	LimeWire	-> 1900	UDP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 2625	UDP	C:\Program Files\LimeWire\LimeWire.exe
204	WinMX	-> 4500	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 4502	UDP	C:\Program Files\WinMX\WinMX.exe
204	WinMX	-> 5353	UDP	C:\Program Files\WinMX\WinMX.exe
504	LimeWire	-> 6257	UDP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 6347	UDP	C:\Program Files\LimeWire\LimeWire.exe
504	LimeWire	-> 6350	UDP	C:\Program Files\LimeWire\LimeWire.exe
0	System	-> 21780	UDP	

The analysis of open ports and running processes may also be useful in Virus and Hacker defense tactics.

How to determine if Limewire installed on the computer

When Limewire is installed, in addition to creating the typical folder under *Program Files*, the installation program also creates a set of folders under the current user's name in the *Documents and Settings* folder. Most of the investigative data is located in these two folders and their subfolders. By default the following folders are created:

- C:\Program Files\Limewire\
- C:\Documents and Settings*<Username>*\.limewire\
- C:\Documents and Settings*<Username>*\Shared\

Folders and files listed below were located in the ".limewire" folder

Folders	Files [.limewire]	Data
.limewire	limewire.props	This is the Limewire properties file, which contains several items of investigative interest including the Client ID.
META-INF	createtimes.cache	java.util.HashMap that includes SHA1 values
themes	fileurns.cache	java.util.HashMap that includes the filename with complete path and associated SHA1 values.
black_theme	fileurns.bak	Backup file for fileurns.cache
classic_theme	installation.props	LimeWire installs file
default_osx_theme	tables.props	LimeWire tables file [table properties]
default_theme	Display.props	Display properties
limewire_theme	filters.props	LimeWire Filters File
limewirePro_theme	questions.props	LimeWire questions file
windows_theme	bugs.data	Bugs data
xml	simpp.xml	Firewall data
data	update.xml	Update data.
display	public.key	sun.security.provider.DSAPublicKey
misc		
schemas		

The Limewire executable file is located in the *C:\Program Files\Limewire* by default.

The shared folder is created in *C:\Documents and Settings\ <Username> \Shared*, under the user account that ran the Limewire installation and setup program, by default. This could be the *Administrator* account or the account of another user on the computer. Sub-folders within the shared folder are also shared.

Are the Limewire program's properties set to allow file sharing?

The following procedure was followed to determine changes with the limewire.props file when file sharing is turned enabled or disabled.

1. Windows 2000 Professional was installed on a VMWare virtual machine.
2. Java runtime environment was downloaded and installed.
3. Limewire 4.10.4 was downloaded and installed with the default options, except for the shared folder, which was set to C:\stuff.
4. Options were examined to determine the status of file sharing. The default Limewire setup enables sharing. A copy of the Limewire.props file was saved.
5. The sharing option was changed to Disable Sharing and applied. A copy of the Limewire.props file was saved under a different name.
6. The saved files were then imported into the table below and a side-by-side comparison was conducted.

Limewire 4.10.4 [Limewire.props]	
File Sharing Enabled	File Sharing Disabled
#LimeWire properties file	#LimeWire properties file
#Wed Jan 25 15:16:36 EST 2006	#Wed Jan 25 15:26:13 EST 2006
PORT=6349	PORT=6349
RUN_ON_STARTUP=false	RUN_ON_STARTUP=false
UPDATE_DELAY=431999998	UPDATE_DELAY=431999998
UPDATE_GIVEUP_FACTOR=24	UPDATE_GIVEUP_FACTOR=24
DOWNLOAD_SNAPSHOT_BACKUP_FILE=C:\\Incomplete\\downloads.bak	DOWNLOAD_SNAPSHOT_BACKUP_FILE=C:\\Incomplete\\downloads.bak
FILTER_HASH_QUERIES=true	FILTER_HASH_QUERIES=true
INSTALLED=true	INSTALLED=true
INCOMPLETE_DIRECTORY=C:\\Incomplete	INCOMPLETE_DIRECTORY=C:\\Incomplete
UI_LIBRARY_TREE_DIVIDER_LOCATION=130	UI_LIBRARY_TREE_DIVIDER_LOCATION=130
AVERAGE_UPTIME=1255	AVERAGE_UPTIME=1825
TOTAL_UPTIME=1255	TOTAL_UPTIME=1825
MIN_CONNECT_TIME=7	MIN_CONNECT_TIME=7
COUNTRY=	COUNTRY=
TREE_NODE_PREFIXES=a	TREE_NODE_PREFIXES=a
	SHARE_DOWNLOADED_FILES_IN_NON_SHARED_DIRECTORIES=false
LAST_ACCEPTABLE_BUG_VERSION=4.10.4	LAST_ACCEPTABLE_BUG_VERSION=4.10.4
CONNECTION_SPEED=1000	CONNECTION_SPEED=1000
LAST_EXPIRE_TIME=1138220003500	LAST_EXPIRE_TIME=1138220003500
DIRECTORY_FOR_SAVING_FILES=C:\\Stuff	DIRECTORY_FOR_SAVING_FILES=C:\\Stuff
UPDATE_DOWNLOAD_DELAY=10000000	UPDATE_DOWNLOAD_DELAY=10000000
MAX_DOWNLOAD_BYTES_PER_SEC=1	MAX_DOWNLOAD_BYTES_PER_SEC=1
DOWNLOAD_SNAPSHOT_FILE=C:\\Incomplete\\downloads.dat	DOWNLOAD_SNAPSHOT_FILE=C:\\Incomplete\\downloads.dat
MAX_SIM_DOWNLOAD=12	MAX_SIM_DOWNLOAD=12
DIRECTORIES_TO_SEARCH_FOR_FILES=C:\\Stuff	DIRECTORIES_TO_SEARCH_FOR_FILES=C:\\Stuff
LAST_GWBCACHE_FETCH_TIME=1138220152703	LAST_GWBCACHE_FETCH_TIME=1138220152703
UNSET_FIREWALLED_FROM_CONNECTBACK=true	UNSET_FIREWALLED_FROM_CONNECTBACK=true
CLIENT_ID=E6924F9AB064E3625F9BE6929D022600	CLIENT_ID=E6924F9AB064E3625F9BE6929D022600
FILTER_WHATS_NEW_ADULT=false	FILTER_WHATS_NEW_ADULT=false
FLUSH_DELAY_TIME=256	FLUSH_DELAY_TIME=256
IDLE_CONNECTIONS=2	IDLE_CONNECTIONS=2

As you can see from the table, the file is updated and the date and time (line 2) change when the changes are applied. The only other change to the file was the addition of Line 17 when File Sharing was disabled. The *Client ID* is a globally unique identifier for this *<Username>* on this computer. Peer precision cases now collect and provide this information, allowing the investigation, the user, and the computer to be tied together with reasonable certainty.

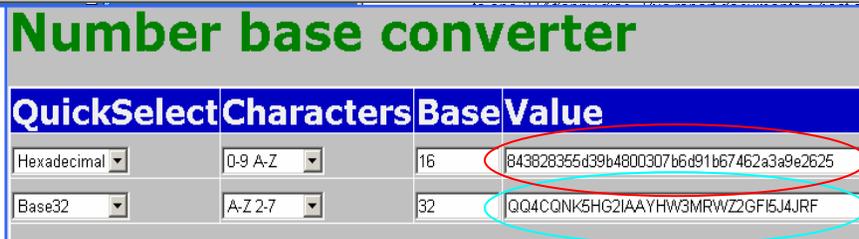
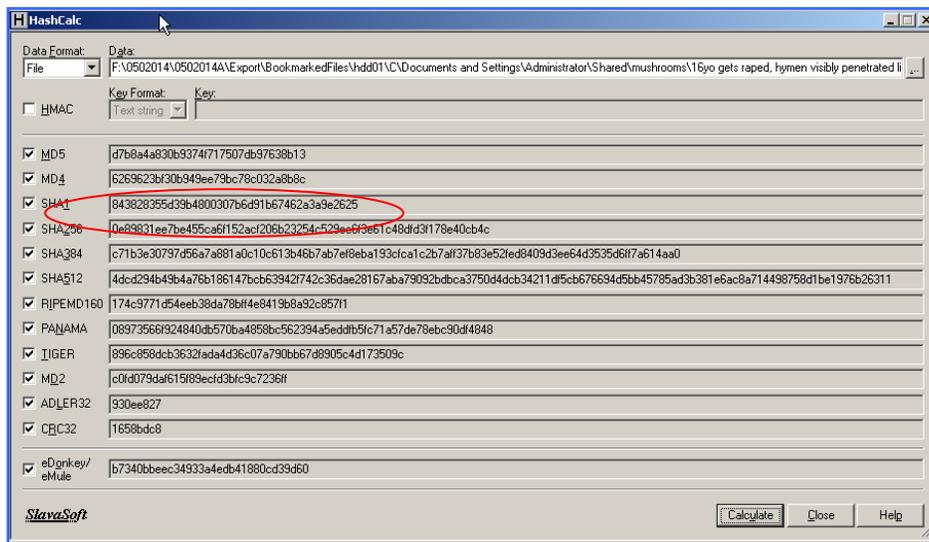
Identify the File Name, MD5 hash value, and the SHA1 value for a known file.

LimeWire like many other Peer-to-Peer programs utilizes Sha1 hash values⁴ to identify files. These hash values are calculated and stored in the UrnCache file. However, they are expressed in base32.

Hash values may be located or calculated, depending on the information that you received from the investigator. An initial investigation, such as a Peer Precision investigation should provide the following information at a minimum and should include a logical copy of the downloaded file. Note that the file name provided by the investigator may not match the file name on the suspect's computer. Do not rely solely on the file name to determine whether the file is present.

Reported File Name = **-Best illegal coitus penetration by flashlight (underage xxx r@ygold pedo nude fuck tiny babyj lolita sister incest girl).mpg**
Reported SHA1 Value = **QQ4CQNK5HG2IAAYHW3MRWZ2GF15J4JRF**
Reported I.P. Address = **24.239.69.187**

Hash values for the target file can be calculated using a program such as *SlavaSoft HashCalc Version 2.01*. However, when you calculate the SHA1 value for the complete target file, you will notice that it may not match the SHA1 reported by the Peer-to-Peer program. This is because Limewire and other Peer to Peer programs that utilize the SHA1 hash value, express the value in Base32, while most hash calculations are reported in Hexadecimal (Base16). If you calculate the SHA1 in Hexadecimal, it will have to be converted to Base32 to match the information reported by Limewire.



<http://darkfader.net/toolbox/>

Information about the file can also be gleaned from UrnCache files used by Limewire or another Peer-to-Peer file sharing program. This is a text fragment from the LimeWire UrnCache file, *C:\Documents and Settings\Administrator\limewire\fileurns.cache*.

```
C:\Documents and Settings\Administrator\Shared\mushrooms\16yo gets raped, hymen visibly penetrated little girl young  
porn real child sex baby.mpgxsq~_sq~  
w_?@_sq~_w+)urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRFq~_xqsq~_w_ü_¥Q¼tW  
C:\Documents and Settings\Administrator\Shared\accounting\Demi Moore nude in shower.jpgxsq~_sq~  
w_?@_sq~_w+)urn:sha1:CTPKHQUBPC5V5VGCWURT25KSBO4IV3IXq~_xqsq~_w_îêXStu  
C:\Documents and Settings\Administrator\Shared\zzxap+\Playboy Playmate - 1996_07 - Angel Boris (2185x4800) CF  
NT.jpgxsq~_sq~ w_?@_sq~_w+)urn:sha1:WYP5PGJOCY2IULGNNNWSBYW2G7EKTOLSq~_xqsq~_w_ýËC  
té C:\Documents and Settings\Administrator\Shared\yanks\blonde shollgirl doggy style - [ search for EroTrix ] - [teen  
young sex xxx erotrix cheerleader nude naked mpg asian ebony amateur erotix lolita porn pussy college girl  
teenage.jpgxsq~_sq~ w_?@_sq~_w+)urn:sha1:MILEBERY2NAPV3ACGWUEK4JSZSYOC16Kq~_xqsq~_w_ü_=_»tW  
C:\Documents and Settings\Administrator\Shared\accounting\jennifer-lopez-nude-3_012.jpgxsq~_sq~  
w_?@_sq~_w+)urn:sha1:SI775ATCBQYJT6WH4KWGBEISI7IPIL3Tq~_xqsq~_w_ý_ŠQÑti
```

The computer in the case study also had Morpheus Ultra set up. The text fragment below was found in *C:\Program Files\StreamCast\Morpheus Ultra\GnuHashes.ini*. Both programs used the same “shared” folder. Because Morpheus Ultra is designed to work with multiple file sharing networks, it stores more data in its UrnCache file.

```
Name:c:\documents and settings\administrator\shared\mushrooms\16yo gets raped, hymen visibly penetrated  
little girl young porn real child sex baby.mpg  
Index:1582  
Size:18058700  
Time:2004 11 17 03 29 31  
urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF  
urn:md5:D7B8A4A830B9374F717507DB97638B13  
urn:ed2k:B7340BBEEC34933A4EDB41880CD39D60  
urn:tree:tiger/:TWSEUDTESKPHSV27QRWHZISTTSFFLB46J7CFCOQ  
urn:bitprint:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF.TWSEUDTESKPHSV27QRWHZISTTSFFLB46J7  
CFCOQ  
TreeSize:6720  
TreeDepth:9  
TigerTree:TWSEUDTESKPHSV27QRWHZISTTSFFLB46J7CFCOQ.X2ZOZBEDYCSIQKWD5VDTJTKKS5VP  
5JZN4XF5XZI.SCCYFMRI254IUWBQOFLWIKPZC6V62T3BKC7O2GA.RFXJJVAPPK7ZXV3FPC2MMN55AM46HJIZ  
NEDVHXQ.5Q5YEKJY6SEBD2RTQRZ2IV6PHOEOW4FD5ARV6QY.SCCYFMRI254IUWBQOFLWIKPZC6V62T3BK  
C7O2GA. End
```

This file will contain comparable data for each file located in the program’s shared folder. In addition, notice that the Morpheus Ultra file is easier to parse out and read than the Limewire file; this is a result of software programming choices. The data and the ability to interpret data in these files vary widely between file sharing client programs and the different file sharing networks.

As you can see, the *Morpheus Ultra\GnuHashes.ini* file relates the file name with full path, Index, Size, and File Created Time with sha1, md5, ed2k (edonkey2000 network), Tiger, Bitprint, and TigerTree hash values. If this data is in text format, it can be easily located by doing a search on the last six characters of the SHA1 value. Once these values are associated, they can be compared against, the file path; hash values, and as described in the next section, uploads of the same file. In this case, the noted file was found in the same path and the md5 hash values matched.

Was the file uploaded by another servent?

By running a keyword search on the SHA1 value or the text string, "GET /uri-res/N2R?" you should locate file fragments similar to the example below. Below are samples from the study case, which were all, located in the C:\pagefile.sys files (both logical and deleted) and unallocated space. Note that "GET" is frequently missing from the string so you may find it more productive to use the test string, "/uri-res/N2R?" By searching the SHA1 value of a specific file (highlighted string), the investigator can articulate a separate attempt to upload that file and potentially a separate count of distribution. For example in the study case, 12 instances were located. In addition, Alt-Location may identify additional computers that have this file available for download and potentially additional targets for investigation. The content of the information provided is based on the User-Agent requesting the file and will vary with between occurrences.

```
GET /uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF HTTP/1.1
Host: 24.239.69.187:6349
Connection: Keep-Alive
X-Features: g2/1.0
Range: bytes=16651969-16717504
User-Agent: Shareaza 2.1.0.0
X-Queue: 0.1
X-Content-URN: urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF
Alt-Location: http://200.84.202.236:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:54Z, http://66.8.229.156:6348/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:53Z, http://217.44.84.201:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:53Z, http://24.59.100.79:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:52Z, http://24.198.80.173:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:52Z, http://67.11.140.28:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:52Z, http://69.169.185.84:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:52Z, http://209.23.240.140:6346/uri-res/N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF 2005-02-02T10:52Z
```

SHA1 Hash of requested file
IP Address/Port for computer providing file
Agent used by requestor
IP Address of Alternate locations where the file is available for download

```
GET /N2R?urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF HTTP/1.1
Host: 24.239.69.187:6350
User-Agent: BearShare 4.6.3.3
Range: bytes=11534336-11796479
X-Alt: 4.28.246.22
X-Gnutella-Content-URN: urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI
X-Connection-Type: Broadband
FP-1a: 128,ØIQföUÍÓÜVëm©0- YÉ *KŠ.º"øè)"3YiV!"
FP-Auth-Challenge: VIYWFLKCAR7HNQOROIPXVFFZXGCO4AHC
Content-Disposition: inline; filename="16yo gets raped, hymen visibly penetrated little girl you ng porn real child sex baby.mpg"
X-Features: browse/1.0, queue/0.1
X-Queue: 0.1
```

SHA1 Hash of requested file
IP Address/Port for computer providing file
Agent used by requestor
IP Address of requestor
File Name

- GET = HTTP command for uploads.
- Host = this was the IP address of the suspect's computer.
- User-Agent: Shareaza 2.1.0.0 = the peer-to-peer program used by the person downloading the file.
- Alt-Location: = Alternate location from which the file is available, potentially new targets.

Keyword Searches

Although Peer-to-Peer file sharing does not create, a lot of log files. Significant information can be located in swap files and unallocated spaced using keyword text string searches. We discussed the use of “/uri-res/N2R?urn” above to locate additional upload attempts. Other characteristics of these networks lend themselves to exploitation using keyword text string searches. For instance, Gnet Search Hits may be returned if a keyword matches any portion of a file name; GNet users exploit this by using descriptive file names, which in turn are easily located with text string searches. For example, in the study case, a keyword search on the string, “r@ygold” returned 5,151 Hits. The majority of the hits were from file names.

r@ygold Search Hits

```
pagefile.sys lolita ass asian preteen raped girl fuck pee cock old anal r@ygold hentai nude group org.jpg@ G P URLP CSC P PVUHOMD<image wi
pagefile.sys s - WWW.EROTRIX.COM - [xxx sex lolita porn erotic flatrix r@ygold young defloration nude naked mpg little tits teen preteen g
pagefile.sys oy asian preteen raped girl fuck pee cock old anal fucking r@ygold handjob hentai nude.jpg urn:sha1:IMKU75CVYBPVKJEDJS7QA6ZJRS
pagefile.sys ss - WWW.EROTRIX.COM - [xxx sex lolita porn erotic flatrix r@ygold young defloration nude naked mpg little tits teen preteen g
pagefile.sys oy asian preteen raped girl fuck pee cock old anal fucking r@ygold handjob hentai nude.jpg@ G P URLP CSC P PVUHOMD<image widt
pagefile.sys lolita ass asian preteen raped girl fuck pee cock old anal r@ygold hentai nude group org.jpg@ G P URLP CSC P PVUHOMD<image wi
pagefile.sys s - WWW.EROTRIX.COM - [xxx sex lolita porn erotic flatrix r@ygold young defloration nude naked mpg little tits teen preteen g
pagefile.sys s - WWW.EROTRIX.COM - [xxx sex lolita porn erotic flatrix r@ygold young defloration nude naked mpg little tits teen preteen g
pagefile.sys oung teen pussy xxx lolita asian virgin spread legs preteen r@ygold britney spears naked fuck cock feet old anal ml.jpg@ G P U
pagefile.sys oy asian preteen raped girl fuck pee cock old anal fucking r@ygold handjob hentai nude.jpg@ G P URLP CSC P PVUHOMD<image widt
pagefile.sys ss - WWW.EROTRIX.COM - [xxx sex lolita porn erotic flatrix r@ygold young defloration nude naked mpg little tits teen preteen g
```

Other Keywords to Consider

Text String	Results
/uri-res/N2R?urn:sha1:	All upload requests
/uri-res/N2R?urn:sha1:<sha1 value>	All upload requests for a specific file
urn:sha1:.....	All SHA1 values [GREP “.” = any character. A SHA1 value has 40 characters] NOTE: Many Hits
<last 6 characters of a known sha1>	All transactions involving that specific file
HTTP/1.1	All HTTP1.1 transactions (Ping, Pong, Query, QueryHits, GET, GIV)
Preteen	Child pornography search terms
Babyj	
Lolita	
Underage	
r@ygold	
<Known username or nick names>	Chat
<Known IP Addresses>	Transactions including a specific IP [the police officer’s IP address may yield the proactive investigation)
<Known GUID>	Transactions including the suspect or police officers GUID (Servent Identifier)

How to compare SHA1 values recovered from the target computer with known files.

Peer Precision utilizes a number of known SHA1 values of suspected child pornography files. By comparing these values to Sha1 values recovered from the suspect computer, it may be possible to demonstrate transactions involving the sharing of child pornography, without actually having any child pornography files. This may be useful in a case where the suspect destroyed the child pornography files prior to the seizure.

- Step 1 - Search Keyword (Grep) urn:sha1:.....
- Step 2 - Export Search hits
- Step 3 - Open new Access Database. Import Exported search hits (Get External Data) into a table.
- Step 4 - Run a Find Duplicates Query on SHA1 values (14,290 unique SHA1 values)
- Step 5 - Import Peer Precision, "alshas" file or other known SHA1<base32> hash sets into a new table
- Step 6 - Create a relationship between the fields containing the SHA values in both tables. Under Join Type, select, "Only include rows where the joined fields from both tables are equal."
- Step 7 - Create a Query including the fields containing the SHA1 values from both the Find Duplicates table and the allshas.

Appendix

Known HTTP Connection Headers

Header	Status	Usage	Example
User-Agent	Mandatory	Name and version of the servent	User-Agent: LimeWire/3.1.0
Remote-IP	Recommended	IP of the remote host as seen by the servent	Remote-IP: 1.2.3.4
X-Try	Recommended	Addresses of known active servents (between 10 and 20)	X-Try-Ultrapeers: 1.2.3.4, 1.2.3.5
Pong-Caching	Recommended	Support of Pong Caching	Pong-Caching: 0.1
GGEP	Recommended	Support of GGEP extensions	GGEP: 0.5
Bye-Packet	Optional	Support of bye messages	Bye-Packet: 0.1
Uptime	Optional	Uptime of the servent	
Vendor-Message	Optional	Support of vendor messages	Vendor-Message: 0.1
Accept-encoding	Optional	Support of message compression	Accept-Encoding: deflate
Content-Encoding	Optional	Acknowledgement of requested compression	Content-Encoding: deflate
X-Try-Ultrapeers	Recommended for ultrapeers	Addresses of known active Ultrapeers (between 10 and 20)	X-Try-Ultrapeers: 1.2.3.6, 1.2.3.7
X-Ultrapeer	Mandatory for ultrapeers	Ultrapeer mode	X-Ultrapeer: Yes
X-Ultrapeer-Needed	Mandatory for ultrapeers	Regulation of Ultrapeers rate	X-Ultrapeer-Needed: Yes
X-Query-Routing	Mandatory for ultrapeers	Support of QRP	X-Query-Routing: 0.1
Hops-Flow	Optional		
Machine	Private	BearShare only	
X-Leaf-Max			
X-Token		GTKG only	
X-Live-Since			

Known HTTP Download Headers

Client headers

Header	Status	Usage	Example
User-Agent	Mandatory	The user agent (for the client only)	User-Agent: LimeWire 2.9.1
Host	Recommended	Address of the server, seen by the client	Host: 123,123,123,123:6346
Connection	Optional (HTTP/1.1)	Support of persistent connections	Connection: Keep-Alive
Range	Mandatory	Requested range	Range: bytes=4932766-5066083
X-Gnutella-Alternate-Location	Recommended	Known alternate locations for the file (HUGE)	see below

Server headers

Header	Status	Usage	Examples
Server	Mandatory	Same as User-Agent (for the server)	Server: BearShare 2.9.0
Content-Type	Recommended	Content-type of requested file	Content-type: application/binary
Content-Length	Mandatory	Length of requested file	Content-length: 133318
Content-Range	Recommended (partial GET)	Range requested from file	Content-Range: bytes 4932766-5066083/5332732
Accept-Ranges	Optional	Support of ranges (this header is not required to request a partial GET)	Accept-Ranges: bytes
Retry-After	Recommended	Time to wait before next attempt to get the file	
X-Gnutella-Content-URN	Recommended	HUGE URN of the requested file	X-Gnutella-Content-URN: urn:sha1:PLST_____YPFB
X-Gnutella-Alternate-Location	Recommended	Known alternate locations of the file (see HUGE for more information)	X-Gnutella-Alternate-Location: http://www.clip2.com/GnutellaProtocol04.pdf X-Gnutella-Alternate-Location: http://10.0.0.10:6346/get/2468/GnutellaProtocol04.pdf X-Gnutella-Alternate-Location: http://10.0.0.25:6346/uri-res/N2R?urn:sha1:PLSTUPAKUZWUGYQYPFB 2002-04-30T08:30Z

Ultrapeer Election Principles

Source - <http://rfc-gnutella.sourceforge.net>

Since Gnutella is a decentralized system, Ultrapeers are elected without the use of a central server. It is up to each node to determine if it is to become an Ultrapeer or a shielded leaf node. First, there are some basic requirements that must be satisfied to even consider becoming an Ultrapeer.

- Not Firewalled – This can usually be approximated by looking at whether the host has received incoming connections.
- Suitable operating system – Some operating systems handle large numbers of sockets better than others. Linux, Windows 2000/NT/XP, and Mac OS/X will typically make better Ultrapeers than Windows 95/98/ME or Mac Classic.
- Sufficient bandwidth – At least 15KB/s downstream and 10KB/s upstream bandwidth is recommended. This can be approximated by looking at the maximum upload and download throughput.
- Sufficient uptime – Ultrapeers should have long expected uptimes. A reasonable heuristic is that the expected future uptime is proportional to the current uptime. That is, nodes should not become Ultrapeers until they have been running at least a few hours.
- Sufficient RAM and CPU speed – Ultrapeers need memory for storing routing tables and CPU speed for routing all incoming queries. Exactly how much is needed depends how efficiently it is implemented and must be experimented with.

If the above criteria are met, a node is said to be Ultrapeer capable. Note that this is not the same as actually being an Ultrapeer. When either an Ultrapeer capable node will actually become an Ultrapeer depends on if there is need for more Ultrapeers on the network, and on how well the above criteria are met. The need for Ultrapeers can be estimated from the number of Ultrapeers found, and can be communicated when new connections are established.

Definitions

<http://en.wikipedia.org/wiki/>

File Sharing Networks

A **peer-to-peer (or P2P)** computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files (see file sharing) containing audio, video, data or anything in digital format is very common, and real-time data, such as telephony traffic, is passed using P2P technology. A pure peer-to-peer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. A typical example for a non peer-to-peer file transfer is an FTP server where the client and server programs are quite distinct, and the clients initiate the download/uploads and the servers react to and satisfy these requests. Some networks and channels, such as Napster, OpenNAP, or IRC @find, use a client-server structure for some tasks (e.g., searching) and a peer-to-peer structure for others. Networks, such as Gnutella or Freenet, use a peer-to-peer structure for all purposes, and are sometimes referred to as true peer-to-peer networks, although Gnutella is greatly facilitated by directory servers that inform peers of the network addresses of other peers.

Direct Client-to-Client (DCC) is an IRC-related sub-protocol enabling peers to interconnect using an IRC server for handshaking in order to exchange files or perform non-relayed chats. Once established, a typical DCC session runs independently from the IRC server. Originally designed to be used with IrCII it is now supported by many IRC clients. A DCC fserve, or file server, lets a user browse, read and download files located on a DCC server. There are many implementations of DCC file servers, among them is the **FSERV** command in the popular mIRC client.

Internet Relay Chat (IRC) is a form of instant communication over the Internet. IRC is designed for group (Many-to-many) communication in discussion forums called channels, but also allows one-to-one communication. IRC is an open protocol that uses TCP and optionally SSL. An IRC server can connect to other IRC servers to expand the IRC network. Users access IRC networks by connecting a client to a server. There are many client and server implementations, such as mIRC and the Bahamut IRCd, respectively. Most IRC servers do not require users to log in, but a user will have to set a nickname before being connected. Using scripts like Sysreset, UPP, Polaris and, most commonly, OmenServe, users can create file servers (**FSERVs**) that allow them to share files.

Napster is an online music service which was originally a file sharing service created by Shawn Fanning. Napster was the first widely used peer-to-peer music sharing service, and it made a major impact on how people, especially university students, used the Internet. Its technology allowed music fans to easily share MP3 format song files with each other, thus leading to the music industry's accusations of massive copyright violations. Although the original service was shut down by court order, it paved the way for decentralized P2P file-sharing programs such as Kazaa, Limewire, and BearShare, which have been much harder to control. Napster continues to live on with pay services today.

FastTrack is a peer-to-peer protocol, used by the Kazaa (and variants, Grokster and iMesh) file sharing programs. The file sharing application Morpheus originally utilized this network, but was later banished from it. The following programs are or have been FastTrack clients:

- Kazaa and variants
- KCeasy
- Grokster
- iMesh
- Morpheus, until 2002
- Apollon - A KDE-Based P2P client
- giFT-FastTrack [2] – a giFT plugin
- A program that supports giFT is Torrent Searcher [3]
- MLDonkey, a free multi-platform multi-network file sharing client

eDonkey network (also called eDonkey2000 network or ed2k) is a file sharing network used primarily to exchange music, films and software. Like most file sharing networks, it is decentralized; files are not stored on a central server but are exchanged directly between users based on the peer-to-peer principle. There are numerous clients for the eDonkey network, some of which are open source or free software:

- eMuleShareaza: an open source multi-network client for Windows.
- MLDonkey: runs on many platforms and supports numerous other file-sharing protocols as well.
- eDonkey2000 (a client of MetaMachine):
- Hydranode: an open source multi-network crossplatform core-gui separated client.
- MediaVAMP (later changed to Pruna): a Korean-only client based on eMule.
- Lphant: a multi-network (eDonkey and BitTorrent) crossplatform client
- Jubster: a multi-network client for Windows

WinMX is a peer-to-peer file-sharing program authored by Frontcode Technologies that runs on Windows operating systems. The official WinMX website and WinMX servers have been offline since September 2005 due to a lawsuit (see the "Decline" section below), though the application remains operable through third-party modifications. WinMX began its life as an OpenNAP client capable of connecting to several servers simultaneously, although Frontcode later created a proprietary protocol, termed WinMX Peer Network Protocol (WPNP), which was used starting with WinMX 2 in May 2001. WPNP version 2 was phased out as WinMX 3.0 and its WPNP version.3 protocol came into existence. Frontcode had operated several cache servers to aid WPNP network operation. Downloads can be very fast for popular songs since the user can run a "multi-point download" that simultaneously downloads the same file in small pieces from several users.

HASH Algorithms

SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions. The most commonly used function in the family, SHA-1, is employed in a large variety of popular security applications and protocols, including TLS, SSL, PGP, SSH, S/MIME, and IPsec. SHA-1 is considered the successor to MD5, an earlier, widely used hash function. The SHA algorithms were designed by the National Security Agency (NSA) and published as a US government standard. SHA1 is normally expressed as a 40 character hexadecimal string, but can also be expressed in <Base32>

MD5 (Message-Digest algorithm 5) is a widely used cryptographic hash function with a 128-bit hash value. As an Internet standard (RFC 1321), MD5 has been employed in a wide variety of security applications, and is commonly used to check the integrity of files. MD5 is normally expressed as a 32 character hexadecimal string.

Tiger is a cryptographic hash function designed in 1995 for efficiency on 64-bit platforms. The size of a Tiger hash value is 192 bits. There also exists 128 and 160-bit versions of this algorithm, called Tiger/128 and Tiger/160. Both variants return truncated Tiger/192 hash values. Tiger is frequently used in **Merkle hash tree** form, where it is referred to as **TTH (Tiger Tree Hash)**. TTH is used by many clients on the Direct Connect and Gnutella file sharing networks.

Hash trees—also known as Merkle trees—are an extension of the simpler concept hash list, which in turn is an extension of the old concept of hashing, for instance; hashing a file. Hash trees where the underlying hash function is Tiger are often called Tiger trees or Tiger tree hashes. The Tiger tree hash is probably the most widely used form of hash tree. It uses a binary hash tree (two child nodes under each node), usually has a data block size of 1024-bytes and uses the cryptographically secure Tiger hash. Tiger tree hashes are used in the Gnutella and Gnutella2 p2p file sharing protocols and in file sharing applications like Direct Connect, BearShare, LimeWire, Shareaza and DC++.

Internet Protocols

Hypertext Transfer Protocol (HTTP) is the method used to transfer or convey information on the World Wide Web. It is a patented open Internet protocol whose original purpose was to provide a way to publish and receive HTML pages. Development of HTTP was coordinated by the World Wide Web Consortium and working groups of the Internet Engineering Task Force, culminating in the publication of a series of RFCs, most notably RFC 2616, which defines **HTTP/1.1**, the version of HTTP in common use today.

Uniform Resource Identifier (URI), is an Internet protocol element consisting of a short string of characters that conform to a certain syntax. The string comprises a name or address that can be used to refer to a resource. It is a fundamental component of the World Wide Web. A **URI** can be classified as a locator or a name or both.

Uniform Resource Locator (URL) is a string of characters conforming to a standardized format, which refers to a resource on the Internet (such as a document or an image) by its location. An HTTP URL is commonly called a **web address** and is usually shown in the address bar of a web browser. Example: <http://www.limewire.com>

Uniform Resource Name (URN) is a Uniform Resource Identifier (URI) that is a string of characters, which refers to a resource on the Internet (such as a document or an image) by its name or a substitute such as a hash value.

Example: <urn:sha1:QQ4CQNK5HG2IAAYHW3MRWZ2GFI5J4JRF HTTP/1.1>

Child Pornography Search Terms (<http://www.urbandictionary.com>)

BabyJ - This is a popular, underage, girl who's victimized countless times by **R@yGold** using popular p2p file sharing programs and newsgroups to spread Child Porn.

r@ygold - Actually NOT a real person; R@ygold is simply a codename used by pedophiles so that they can easily locate each other's media. R@ygold is a keyword added to image and video files with illegal pornographic content, so that those dealing in child porn can locate and share files over P2P networks.

Lolita – Lolita is a nickname for Delores. It is also a term used to describe a prepubescent or adolescent girl who is attractive and sexually responsive. She lusts after older men, and is lusted by

them in return. The term originates from the Vladimir Nabakov novel "Lolita" which told the tale of the love affair between middle aged Humbert and his 12 year old stepdaughter Lolita. Lolita is also defined as type of pornography depicting underage girls, or woman made to appear as underage girls

PTHC – Short for "preteen hardcore" or child porn. Used mostly on p2p programs.

References

¹ "The **Gnutella Development Forum**, Gnutella/0.6 online at <http://www.the-gdf.org>
Copyright (C) 2002, Tor Klingberg & Raphael Manfredi, All Rights Reserved.

² **LimeWire Design**, Christopher Rohrs, 8/20/2001, online at <http://www.limewire.org/techdocs/design.html>

³ **Limewire Installation**, http://www.limewire.com/english/content/ug49/ug_installation.shtml

⁴ **Hash/URN Gnutella Extensions (HUGE) v0.94**, Gnutella Developer Forum, G. Mohr Bitzi, Inc, April 30, 2002,
http://rfc-gnutella.sourceforge.net/src/draft-gdf-huge-0_94.txt