

### **2.5.4 Other Potential Topologies**

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

## **3. Napster**

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

### **3.1 The Napster Architecture**

The architecture of Napster is based on the Centralized Model of P2P file-sharing (Martin, 2000). It has a Server-Client structure where there is a central server system which directs traffic between individual registered users. The central servers maintain directories of the shared files stored on the respective PCs of registered users of the network. These directories are updated every time a user logs on or off the Napster server network. Clients connect automatically to an internally designated "metaserver" that acts as common connection arbiter. This metaserver assigns at random an available, lightly loaded server from one of the clusters. Servers appeared to be clustered about five to a geographical site and Internet feed, and able to handle up to 15,000 users each. The client then registers with the assigned server, providing identity and shared file information for the server's local database. In turn, the client receives information about connected users and available files from the server. Although formally organized around a user directory design, the Napster implementation is very data centric. The primary directory of users connected to a particular server is only used indirectly, to create file lists of content reported as shared by each node (David, 2001).

Users are almost always anonymous to each other; the user directory is never queried directly. The only interest is to search for content and determine a node from which to download. The directory therefore merely serves as a background translation service, from the host identity associated with particular content, to the currently registered IP address needed for a download connection to this client. Each time a user of a centralized P2P file sharing system submits a request or search for a particular file, the central server creates a list of files matching the search request, by cross-checking the request with the server's database of files belonging to users who are currently connected to the network. The central server then displays that list to the requesting user. The requesting user can then select the desired file from the list and open a direct HTTP link with the individual computer which currently possesses that file. The download of the actual file takes place

directly, from one network user to the other, without the intervention of the central server. The actual MP3 file is never stored on the central server or on any intermediate point on the network.

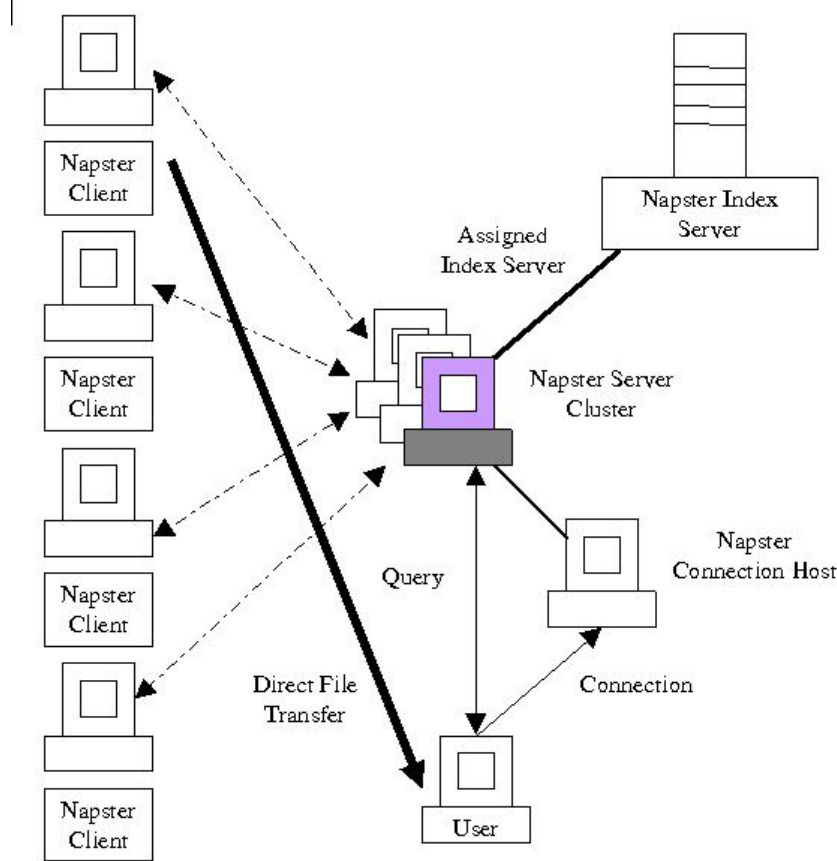


Figure 8: Illustration of the Napster Architecture

### 3.2 The Napster Protocol

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

Napster works with a central server which maintains an index of all the mp3 files of the peers. To get a file you have to send a query to this server which sends you the port and IP address of a client sharing the requested file. With the Napster application it is now possible to establish a direct connection with the host and to download a file.

In contrast to Gnutella who only uses a handful of messages to communicate between peers, the Napster protocol uses a whole lot of different types of messages. Every state of

the hosts, acting like clients towards the server, is related to the central Napster server. Thus the Napster protocol makes anonymity impossible. At first, one may think that this is a drawback, but this complex protocol actually makes a lot of services possible. Some examples are:

- Creating Hotlists: notifying when users of your own hotlist sign on or off the server
- List of ignored User
- Instant Messaging: sending public or private messages to other users; creating and joining channels of shared interests

### 3.2.1 Napster Messages Data Structures

Each message to/from the Napster central server is of the form in the figure below:

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

Figure 9: Messages Data Structures

Where:

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

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

### 3.2.2 Initialisation

A registered Napster host, acting like a client, sends to the server a LOGIN(0x02) message with the following format:

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

Figure 10: Login Message Structure

Where:

- Nick & Password identify the user
- Port is the port which the client is listening on for data transfer.
- Client\_Info is a string containing the client version info.
- Link\_Type}is a integer indicating the client's bandwidth.

The details are given in Table 1:

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

Table 1: Assessment Details

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

An unregistered host sends a New User Login (0x06) which is similar to the format of Login (0x02), with the addition of the email address on the end. The server sends a Login Ack (0x03) to the client after a successful login. If the nick is registered, the email address given at registration time is returned, else, a dummy value will be returned.

### 3.2.3 Client Notification of Shared File (0x64)

With the Client Notification of Shared File (0x64) message the client sends successively all the files it wants to share.

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

Figure 11: Client Notification of Shared File (0x64)

- MD5 is the hash value of the shared file. The MD5 (Message Digest 5) algorithm produce a 128-bit "fingerprint" of any file. It is nearly computationally infeasible to produce two messages having the same hash value. The MD5 algorithm is intended to provide any user the possibility to secure the origin of his shared file, even if the file is laying on drives of other Napster users.
- Size is the file size in bytes
- Bitrate is the bit rate of the mp3 in kbps
- Frequency is the sample rate of the mp3 in Hz
- Time is the duration of the music file in seconds

### 3.2.4 File Request

The downloading client will first issue either a Search (0xC8) or Browse(0xD3). The first message has the following format:

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

Figure 12: Search Message (0xC8)

- Max is the maximum number of results.
- Link-Type Range is the range of link-types.
- Bit-rate Range is the range of bit-rate.
- Frequency Range is the range of sample frequencies in Hz.

The artist name and the song title are checked from the file name only. Napster does not make use of the ID3 in mp3 files in its search criteria.

The payload of the Browse (0xD3) message does only contains the <nick> of the host. It requests a list of the host's shared files.

### 3.2.5 Response and Browse Response

The server answers respectively with a Search Response (0xC9) or a Browse Response (0xD4) with the formats given in the figure below:

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

Figure 13: Response Message

Where:

- MD5 hash value of the requested file
- Size file size in bytes
- Bitrate bit rate of the mp3 in kbps
- Frequency sample rate of the mp3 in Hz
- Time specify the length of the file
- Nick identify the user who shares the file
- IP 4 Bytes integer representing the IP address of the user with the file.
- Link-Type Refer to Login Message.

### 3.2.6 Download Request

To request a download, a DOWNLOAD REQUEST (0xCB) message is sent to the server. The client requests to download <filename> from <nick>. This message has the following payload format:

<Nick>	<Filename>
--------	------------

Figure 14: Download Request Message

### 3.2.7 Download ACK

The server will answer with a DOWNLOAD ACK (0xCC) containing more information about the file (Linespeed, Port Number, etc). This message has the following payload format:

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

Figure 15: Download Acknowledgement Message

### 3.2.8 Alternate Download Request

It is like the normal "Download Request", only difference is that it is for use when the person sharing the file can only make outgoing TCP connection because of the firewall that is blocking the incoming messages. The ALTERNATE DOWNLOAD REQUEST (0x1F4) message should be used to request files from users who have specified their data port as '0' in their login message.

### 3.2.9 Alternate Download Ack

This ALTERNATE DOWNLOAD ACK (0x1F5) is sent to the uploader when its data port is set to 0 to indicate they are behind a firewall and need to push all data. The uploader is responsible for connecting to the downloader to transfer the file.

### 3.2.10 File Transfer

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

<Nick>	<Filename>	<Offset>
--------	------------	----------

Figure 16: Request Message

- Nick is the client's nick.
- Offset is the byte offset in the file to begin the transfer at. It is needed to resume prior transfer.

The remote host will then return the file size and, immediately following, the data stream. The direct file transfer between Napster hosts uses a P2P architecture. Once the data transfer is initiated, the downloader should notify the server that they are downloading a file by sending the DOWNLOADING FILE (0xDA) message. Once the transfer is complete, the client sends a DOWNLOAD COMPLETE (0xDB) message.

### **3.2.11 Firewalled Downloading**

Napster also has method to allow clients behind firewalls to share their contents as well. As described above, when the file needs to be pushed from a client behind a firewall, the downloader sends a message ALTERNATE DOWNLOAD REQUEST (0x1F4) message to the server. This causes an ALTERNATE DOWNLOAD ACK (0x1F5) to be sent to the uploader, which is similar to the DOWNLOAD REQUEST (0xCB) message for a normal download.

Once the uploader receives the (0x1F5) message from the server, it should make a TCP connection to the downloader's (0x1F5) data port (given in the message). Upon connection, the downloader's client will sent one byte, the ASCII character '1'. The uploader should then send the string "SEND" in a single packet, and then the message (format was shown before).

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

## **3.3 Implementation**

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

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

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

Nonmusic exchanges over Napster proper were never more than marginal, at least compared to alternative, content-agnostic systems such as Gnutella. Some alternative Napster servers such as OpenNap started as "safe-havens" for Napster users when Napster began filtering content, did for a while begin to fill the gap, tying together former

Napster clients, clones and variations with a new kind of server that extended the original Napster protocol to all file types. No matter how much the Napster model was reengineered, however, the fundamental requirement of a "Napster-compatible" central server remained a serious constraint for a network based on this technology or any of its clones. To transcend this limitation, other protocols and architecture models are needed- for example, serverless networks in the style of Gnutella.

## **4. Gnutella**

In the early of March 2000, Gnutella was originated by Justin Frankel and Tom Pepper, working under the GnuLLsoft, which is one of the AOL subsidiaries. However, Gnutella's development was halted by AOL shortly after it was published, but during that time several curious programmers already completed downloading it. Thanks to those downloadings, many open-source developers quickly reverse-engineered Gnutella's communication protocol and published a number of Gnutella clones with several improvements, e.g., LimeWire, BearShear, Gnucleus, XoloX, and Shareaza.

### **4.1 The Gnutella Architecture**

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

In a Gnutella network, servents are connected to each other in a flat ad-hoc topology, or a P2P networks for servents. A servent works like a client and a server by itself. As a server, it responses to queries from another peer servent. As a client, it issues queries to other peer servents.

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

It is thus important to stay in contact with several servents at the same time to prevent being disconnected from the network. Once a server receive the ping message, it sends back a pong message to the server that originated the ping message using the same path that the ping message came from. A pong message contains the details of the servent like port, IP address, the number of files shared, and the number of kilobytes shared.