5. First and Second Generation of Peer-to-Peer Systems

Jörg Eberspächer, Rüdiger Schollmeier
(Munich University of Technology)

5.1 General Characteristics of Early Peer-to-Peer Systems

Peer-to-Peer (P2P) networks appeared roughly around the year 2000 when a broadband Internet infrastructure (even at the network edge) became widely available. Other than traditional networks Peer-to-Peer networks do not rely on a specific infrastructure offering transport services. Instead they form "overlay structures" focusing on content allocation and distribution based on TCP or HTTP connections. Whereas in a standard Client-Server configuration content is stored and provided only via some central server(s), Peer-to-Peer networks are highly decentralized and locate a desired content at some participating peer and provide the corresponding IP address of that peer to the searching peer. The download of that content is then initiated using a separate connection, often using HTTP. Thus, the high load usually resulting for a central server and its surrounding network is avoided leading to a more even distribution of load on the underlying physical network. On the other hand, such networks are typically subject to frequent changes because peers join and leave the network without any central control.

While some legal aspects of Peer-to-Peer networks are still heavily contended between the entertainment industry and some user groups, we focus on the technical aspects of this approach. In the last years, several Peer-to-Peer technologies were developed. Figure 5.1 provides an overview of current Peer-to-Peer technologies and compares them to the conventional Client-Server model.

As shown in Figure 5.1, in a Client-Server system the server is the only provider of service or content, e.g. a web server or a calendar server. The peers (clients) in this context only request content or service from the server, the IP address of which is assumed to be available to the peers. Content in this context may be an MP3-compressed audio file, the profile of a person a user wants to establish a call to or context information, e.g. where the next taxi can be found. The clients do not provide any service or content to run this system. Thus generally the clients are lower performance systems and the server is a high performance system. This does not exclude that a server may be set up as a server farm with one specified entry point for the clients, which redirects the clients to different computers to share the load.

R. Steinmetz and K. Wehrle (Eds.): P2P Systems and Applications, LNCS 3485, pp. 35-56, 2005.

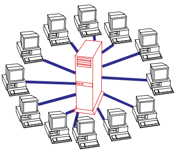
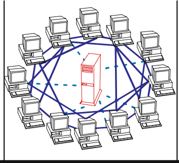
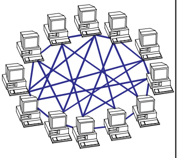
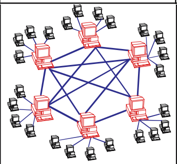
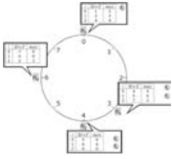
Client-Server	***Peer-to-Peer***			
	1. Resources are shared between the peers 2. Resources can be accessed directly from other peers 3. Peer is provider and requestor (Servent concept)			
	Unstructured P2P			***Structured P2P***
	1st Generation		***2nd Generation***	
1. Server is the central entity and only provider of service and content. → Network managed by the Server 2. Server as the higher performance system. 3. Clients as the lower performance system Example: WWW	*Centralized P2P*	*Pure P2P*	*Hybrid P2P*	*DHT-Based*
	1. All features of Peer-to-Peer included 2. Central entity is necessary to provide the service 3. Central entity is some kind of index/group database Example: Napster	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities Examples: Gnutella 0.4, Freenet	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → dynamic central entities Example: Gnutella 0.6, JXTA	1. All features of Peer-to-Peer included 2. Any terminal entity can be removed without loss of functionality 3. → No central entities 4. Connections in the overlay are “fixed” Examples: Chord, CAN
				

Fig. 5.1: Summary of the characteristic features of Client-Server and Peer-to-Peer networks

In contrast, in Peer-to-Peer systems all resources, i.e. the shared content and services, are provided by the peers. Some central facility may still exist, e.g. to locate a given content. A peer in this context is simply an application running on a machine, which may be a personal computer, a handheld or a mobile phone. In contrast to a Client-Server network, we can generally not distinguish between a content requestor (client) and a content provider, as one application participating in the overlay in general offers content to other peers and requests content from other participants. This is often expressed by the term “Servent”, composed of the first syllable of the term Server and the second syllable of the term Client.

Using this basic concept Figure 5.1 outlines various possibilities currently used. Peer-to-Peer networking started with the first generation centralized concept. In this case some central server is still available. However, contrary to the Client-Server approach this server only stores the IP addresses of peers where some content is available, thus greatly reducing the load of that server. However, the address of that server must be known to the peers in advance. This concept was widely used and became especially well known due to Napster, offering free music downloads by providing the addresses of peers sharing the desired content. This approach subsequently lost much of its importance due to legal issues.

As a replacement for that scheme decentrally organized schemes such as Gnutella 0.4 and Freenet became widely used. These schemes do not rely on any central facility (except possibly for some bootstrap server to ease joining such a network), but rely on flooding the desired content identifier over the network, thus reaching a large number of peers. Peers which share that content will then respond to the requesting peer which will subsequently initiate a separate download session.

It is an important drawback of these schemes that they generate a potentially huge amount of signaling traffic by flooding the requests. In fact, that signaling traffic dominates the Internet traffic in some cases even today. To avoid that, schemes like Gnutella 0.6 or JXTA introduce a hierarchy by defining Superpeers, which store the content available at the connected peers together with their IP address. Thus the Superpeers are often able to answer incoming requests by immediately providing the respective IP address, so that on average less hops are required in the search process, thus reducing the signaling traffic.

The above schemes are generally termed "Unstructured Peer-to-Peer", because the content stored on a given node and its IP address are unrelated and do not follow any specific structure. Contrary to that also Peer-to-Peer approaches have been proposed which establish a link between the stored content and the IP address of a node. In the rightmost column of Figure 5.1 such networks are termed "Structured Peer-to-Peer". The link between a content identifier and the IP address is usually based on Distributed Hash Tables (DHT) (cf. Chapter 7). However, in a frequently changing network such an approach requires frequent redistribution of content. We therefore do not address this approach in more detail.

5.2 Centralized Peer-to-Peer Networks

5.2.1 Basic Characteristics

As described above, centralized Peer-to-Peer networks are characterized by the fact that they rely on one central lookup server. The overlay topology of a centralized Peer-to-Peer network can therefore be described as a star network. Every peer is connected to the centralized lookup server, to which it can issue requests for content matching the keywords stated in the request. If the request can be resolved by the centralized lookup server, it returns the access coordinates of the peers (mostly IP-addresses and ports) offering content which is described with the same keywords as stated in the request. The content itself is then transmitted out of band, i.e. not via the signaling (overlay) network, but via an additional, mostly HTTP-based, connection.

The most prominent example application, which is based on a centralized Peer-to-Peer network, is Napster `http://www.napster.com/what_is_napster.html`. Napster is used for (free) file sharing between Internet users and is considered as the starting point of Peer-to-Peer networks. Due to legal issues and the centralized responsibility Napster had to change its service to a legal file sharing service. The basic concept and architecture of the Napster file sharing system is still used by other applications, e.g. Audiogalaxy [38] or WinMX [625]. BitTorrent [127, 320] is a similar file sharing system, the major objective of which is to quickly replicate a single large file to a set of clients.

As depicted in Figure 5.1 the Napster network can be characterized by its centralized topology. The file searching protocol uses a Client-Server model with a central index server. However the file transfer is done in a true Peer-to-Peer way. The file exchange occurs directly between the Napster hosts without passing the server.

With Napster, no file can be found, if the central lookup table is not available. Only the file retrieval and the storage are decentralized. Thus the server represents a bottleneck and a single point of failure. The computing power and storage capabilities of the central lookup facility must grow proportional to the number of users, which also affects the scalability of this approach.

As every node wanting to log into the Napster network has to register at the central server, no keep alive signal or any electronic heart beat must be exchanged between the Napster server and the peer. The server acts comparable to a DNS server to guide each requesting peer to those peers, which host the demanded content. No additional application layer routing is necessary, as the server has a complete network view.

Further on, if the content is shared by at least one participant, the content can be found instantly with one lookup. Thus the content availability in a Napster network can only take the values zero or one. Zero, if the content is not shared by any node, one if the content is shared by at least one node, assuming that the server and the peers work correctly. If the content is available more than once, only the replication rate, and thus in this case the download performance increases, but not the availability of content.

5.2.2 Signaling Characteristics

The messages employed in Napster are fairly simple and easy to track, as they are transmitted as plain text messages. We describe in the following the basic messages used in Napster to announce and to search for content.

Each message to/from the Napster server has the basic structure given in Figure 5.2. The first four bytes provide the <Length> parameter, which specifies the length of the payload of this message. The <Function> parameter stated in the following four bytes, defines the message type, e.g. login or search, which are described in the following. The payload finally carries parameters necessary for the different messages, e.g. the keywords of a search message.

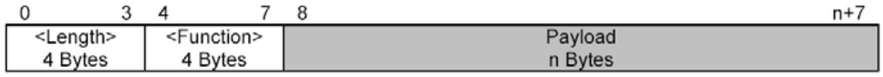


Fig. 5.2: Basic Napster message structure

The blocks/parameters in the payload are separated by spaces. This makes the separation of the information provided in each incoming message possible, as most blocks have no fixed length. We divide the messages in two phases, the initialization and the file request. The <Function> parameter of each message is given in brackets, e.g. SEARCH (0xC8).

Initialization

A registered Napster host, acting as a client, sends to the Napster server a LOGIN (0x02) message to become a member of the overlay network. For user verification this message includes the nickname (<nick>) and <password> of the user who started the application. Further on this message also includes the port number (<port>) on which the peer listens for incoming data requests and information about the clients access data-rate (<Link Type>). The <Client-Info> parameter contains information about the version of the used software. On average a LOGIN-message is about 40 bytes long.

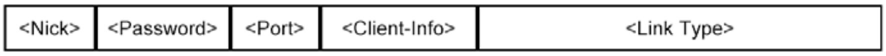
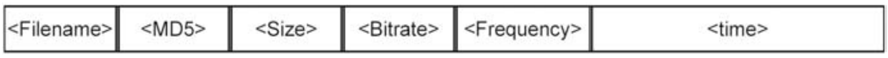


Fig. 5.3: LOGIN (0x02) message

After a successful login, the server sends a LOGIN ACK (0x03) (size: 20 bytes) to the client. If the <nick> is registered, the email address given at registration time is returned.

If the <nick> is not registered, a dummy value is returned. As soon as the peer is logged in, it sends one "CLIENT NOTIFICATION OF SHARED FILE" (0x64) message for every file it wants to share (see Figure 5.4). Thus routing is possible, as every client announces its shared objects to the Napster server. This message contains the filename (<Filename>) of the file, the MD5-hash value of the file <MD5> [519] and the size in byte of the file (<Size>). The MD5 (Message Digest 5) algorithm produces a 128-bit "fingerprint" of any file. It is extremely unlikely that two messages contain the same hash value.

The MD5 algorithm is therefore intended to provide any user with the possibility to secure the origin of the shared file, even if parts of the file are provided by different Napster users. As specific parameters of the music file, this message additionally provides the bitrate (<Bitrate>), the sampling rate of the MP3 (<frequency>), and the playout time of a music file (<time>). The bit rate represents the quality of the used coding and compression algorithm. The average size of this message is 74 bytes.

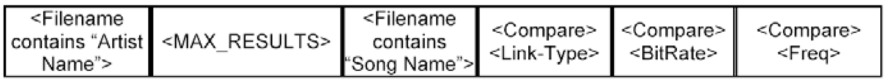


<Filename>	<MD5>	<Size>	<Bitrate>	<Frequency>	<time>

Fig. 5.4: CLIENT NOTIFICATION OF SHARED FILE message (0x64)

File Request

To be able to download a file from the Napster network, peers which share the requested file have to be found. The format of a request is shown in Figure 5.5. Therefore the requesting peer sends a SEARCH (0xC8) message to the Napster server. To specify the search this message contains several parameters stating keywords describing the requested object (artistname and parts of the songname). Further on this message also specifies a filter, e.g. to state a certain quality of the requested file, like the bitrate and the sampling frequency of the requested file. The parameter <compare> can have the values “at least”, “at best” or “equal to”. Thus the requesting peer can choose the quality of the file and also the file size, which together with the link type (parameter <Link Type> e.g. a T1 connection) of the providing peer can strongly influence the download speed. The parameter <MAX_RESULTS> finally states the maximum number of results the requesting peer wants the Napster server to return. The average size of such a message is 130 bytes.



<Filename contains “Artist Name”>	<MAX_RESULTS>	<Filename contains “Song Name”>	<Compare> <Link-Type>	<Compare> <BitRate>	<Compare> <Freq>

Fig. 5.5: SEARCH message (0xC8)

Upon receiving a SEARCH message, the Napster server tries to match the parameters stated in the SEARCH message with the entries of its database, consisting of data previously received from other peers upon initialization (CLIENT NOTIFICATION OF SHARED FILE (0x64) messages). If the server can resolve the query, it answers with at least one RESPONSE (0xC9) containing information about shared files matching the previously stated criteria (see Figure 5.6). To provide the requesting peer with information about the available data and where it can be downloaded from, this message contains the full filename (<File-Name>) and the IP-address (<IP>) of the providing peer, so that the requesting peer can download the requested file directly via its HTTP-instance [365]. Further on the file size (<Size>), the playout time (<length>), the sample and the bitrate of the file are stated (<Freq>, <Bitrate>). To check the integrity of the file and to be able to download the file from multiple sources the MD5 hash value of the shared file is also stated (<MD5>). The average size of such a message is 200 bytes.

<FILE-NAME>	<MD5>	<Size>	<Bit-rate>	<Freq>	<length>	<Nick>	<IP>	<Link-Type>

Fig. 5.6: RESPONSE message (0xC9)

5.2.3 Discussion

To summarize the details of the Napster protocol we provide as an example the message sequence chart for the communication between two Napster peers and the Napster server in Figure 5.7. Here the requesting peer (Req) first initializes at the Napster server. As mentioned above the requesting peer (Req) therefore sends a LOGIN message to the Napster server with a payload of 36 bytes, which equals to 0x24 bytes in hexadecimal notation. Upon receiving the acknowledgement it announces its three shared objects to the Napster server. In this example we assume the same message lengths, given by the average message length stated above.

Now the new peer is fully registered with the Napster network and can start a search. Therefore it sends a SEARCH message to the Napster server, including the search keywords describing the requested object. As the Napster server in our example knows two possible peers which share the requested object, it answers with two RESPONSE messages. Thus the peer can now request a download of the requested object from one of the providing peers with a HTTP-Get-request. In case of success, as assumed in this example, the providing peer responds to this request with an OK message, which includes the requested file. In this figure we can clearly see, that besides the initialization traffic only few traffic is caused by this Peer-to-Peer network. The reason is that only one central lookup table is available and therefore no flooding is necessary to find the requested object. The Napster server thus works similar to a DNS-lookup server.

If we assume a user, which shares 10 files and requests one comparatively popular file, which thus would result in 20 responses, we can compute the generated bytes to:

$$\begin{aligned} 1 \cdot (login + login\_ack) + 10 \cdot notif + 1 \cdot search + 10 \cdot response = \\ = 40 + 4 + 10 \cdot 74 + 130 + 10 \cdot 200 = 2914 bytes \end{aligned} \tag{5.1}$$

If we further on assume an average session length of 10 minutes, we can compute the average necessary signaling data rate to 38.85 bits/s, which is very reasonable.

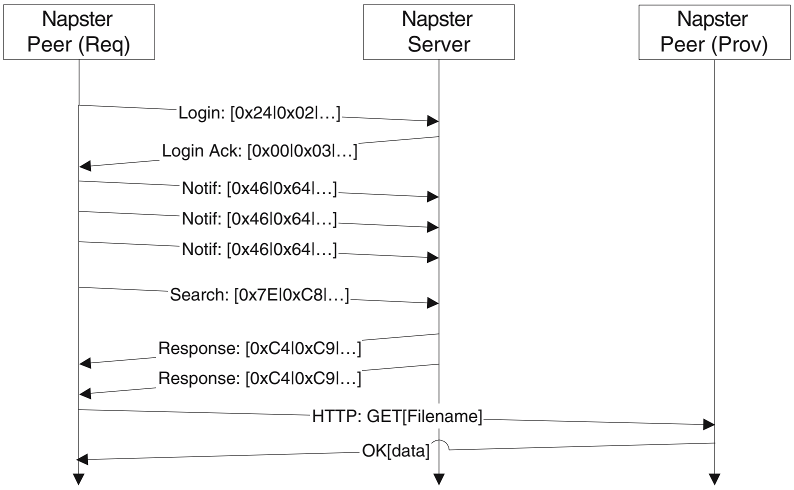


Fig. 5.7: Sample message sequence chart for one Napster server with one requesting and one providing peer

5.3 Pure Peer-to-Peer-Networks

5.3.1 Basic Characteristics

Pure Peer-to-Peer networks/protocols came up shortly after the introduction of Napster. Examples of these protocols are the Freenet protocol and the Gnutella 0.4 protocol [123, 126, 232] . To analyze the properties, possibilities and limitations of pure Peer-to-Peer networks, we describe the Gnutella 0.4 protocol in this section. The Gnutella 0.4 network [126] consists of a large number of nodes which may be distributed throughout the world, without any central element. The overlay topology can be characterized by a node degree distribution as given by equation 5.2 [328]. With this truncated powerlaw distribution, ranging from degree (d) one to a maximum degree of seven, we can describe the topology of a Gnutella 0.4 network and can generate networks graphs as given by Figure 5.8. Here we can observe that separated subcomponents may occur due to the random connection establishment. This is also expected to happen in real networks, although in this case the subcomponents are magnitudes larger, as also the total number of considered nodes is magnitudes larger.

$$p\left(d\right) = \left\{ \begin{array}{l} c \cdot d^{-1.4},\ 0 < d \leq 7 \\ 0,\ in\ any\ other\ case \end{array} \right. ,\ with\ c = \left(\sum_{d} \frac{p(d)}{c} \right)^{-1} \tag{5.2}$$

$$average:\ \bar{d} = 2.2$$
$$\operatorname{var}\left(d\right) = 1.63$$

A node becomes part of the Gnutella network by establishing an average of 3 TCP-connections to other active Gnutella nodes, whose IP addresses it may receive from a bootstrap server [549]. New nodes, to which the node can connect if an active connection breaks, are explored by broadcasting PING messages in the virtual overlay network. These PING messages are also used as keep alive pattern and are broadcasted in regular time intervals.

All messages are coded in plain text. This results in large message sizes of QUERY and especially QUERY-HIT messages, as they contain meta data about the queried objects. Similar to Napster, Gnutella uses MD5 hash keys [519] to identify objects explicitly. For routing Gnutella employs simple flooding of the request messages, i.e. QUERY and PING messages. Every new incoming PING or QUERY, which has not been received before, is forwarded to all neighbors except the one it received the message from, if the Time-to-Live (TTL) value (default set to seven hops) is at least one. If a node receives the same message more than once, these messages are not further flooded. Response messages, like PONG or QUERY-HIT messages, are routed back on the same path the request message used, which is called backward routing.

In Gnutella 0.4 the virtual Peer-to-Peer layer is not matched to the physical layer, which leads to zigzag routes, as described in [550]. Only enhancements, as described by the approach of geo-sensitive Gnutella [550], provide means to adapt the virtual network to the physical network.

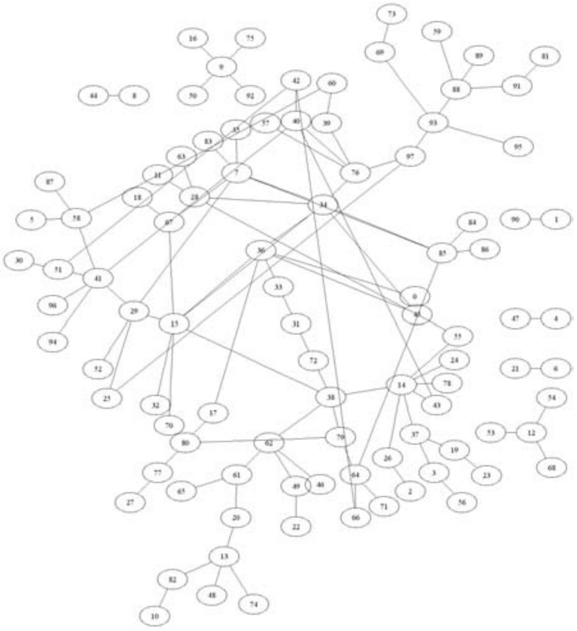


Fig. 5.8: Sample graph of a simulated Gnutella 0.4 network (100 nodes)

5.3.2 Signaling Characteristics

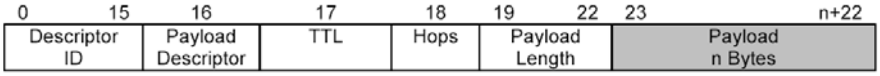
The nodes communicate directly with each other without any central instance . However at the beginning, i.e. in a bootstrap phase, a central entity like a beacon server, from which IP addresses of active nodes can be retrieved, is necessary. If a node already participated in the network, it may also be able to enter the network by trying to connect to nodes, whose addresses it cached in a previous session. As soon as a new node knows the IP address and port of one active Gnutella node it first establishes a TCP connection to this node and then connects to this node by sending the ASCII encoded request string "GNUTELLA CONNECT/<protocol version string>\n\n" to it. If the participating peer accepts this connection request it must respond with a "GNUTELLA OK\n\n".

Gnutella mainly uses four messages as stated above. The messages are setup in a similar manner as in Napster. They consist of a general message header and the additional payload (see Figure 5.9). However since in Gnutella the messages are flooded through the overlay network, some additional parameters are necessary beyond those used for Napster. The <Descriptor ID> is a 16-byte string uniquely identifying the message on the network. Thus circles can be detected, i.e. every message which is received twice by a node is not forwarded any further. Simultaneously and backward routing of possible response messages is possible.

Every node therefore has to store this ID and the IP address from which it received the message for a certain time. The <TTL> (Time-to-Live) value determines the number of hops a message is forwarded in the overlay network. This value is decreased by every node which received the message before the message is forwarded. When the TTL value reaches zero, the message is not forwarded any further, to avoid infinitely circulating messages. Generally a TTL value of seven is considered to be sufficient to reach a large fraction of the nodes participating in the overlay network. The <Hops>-value states the number of hops a message has already been forwarded and is therefore increased by one by every forwarding peer. It can be used to guarantee, that initially no larger value than seven has been used by a requesting peer, as

$$TTL(0) = TTL(i) + Hops(i) \leq 7 \tag{5.3}$$

The <Payload length> parameter states the size of the message so that the next message in the incoming stream can clearly be identified.



0 15	16	17	18	19 22	23 n+22
Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length	Payload n Bytes

Fig. 5.9: Basic Gnutella message structure

However the most important field, which determines the payload is the <Payload-Descriptor> field. The messages we distinguish here are 0x00 for a PING, 0x01 for a PONG, 0x80 for a QUERY and 0x81 for a QUERYHIT message [126]. The network exploration message PING does not contain any payload, whereas in the payload of the PONG message in addition to the contact information (IP address+port) information about the amount of shared files is stated. To search for data, the QUERY message contains, besides the parameter which states the requested minimum download speed, a null terminated search string containing the keywords separated by blanks, describing the requested object. The average size of this message is 78.4 bytes. If we now assume, that an average word has a length of eight characters plus one character for the blank, we can also compute the average number of words a user states as search criteria, as every character is described with one byte. For Gnutella this results in an average of 7.35 words per QUERY. Similar to the PING messages, the QUERY messages are flooded through the overlay. As soon as one node receives a QUERY-message, it compares the search keywords to the keywords describing the locally shared content. In case of at least one hit, it sends back a QUERYHIT message which is routed back on the same way the QUERY message was distributed through the network (backward routing). A QUERYHIT message contains the information, as shown in Figure 5.12 and Figure 5.13. However in contrast to Napster one QUERYHIT message can contain in its result set more than only one file. The average size of one QUERYHIT message is 748.8 bytes, which is comparatively large.

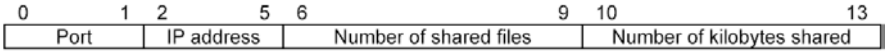


Fig. 5.10: PONG (0x01) payload structure

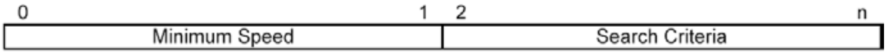


Fig. 5.11: QUERY (0x80) payload structure

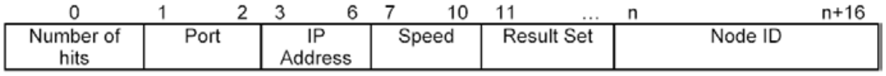


Fig. 5.12: QUERYHIT (0x81) payload structure

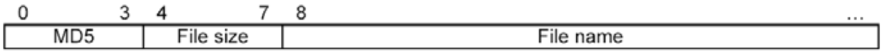


Fig. 5.13: Result set structure

5.3.3 Discussion

To summarize the basic signaling behavior of a Gnutella network we assume a sample Gnutella network, where node 1 just joined (see Figure 5.14). Therefore node 1 first sends a CONNECT message to the nodes 5, 2 and 3 (see Figure 5.15). To explore its surrounding further on, node 1 also sends a PING message to its neighbors, which forward this message further and thus this message and its corresponding PONG messages propagate through the network, as shown in Figure 5.15.

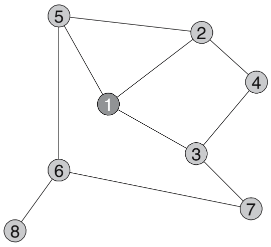


Fig. 5.14: Sample Gnutella 0.4 network

In our example the flooding of the request messages results, as we can see from Figure 5.15, in 12 PING and 12 PONG messages, and 6 messages for the initial connection establishment. Taking the message sizes from above into account (PING: 23 byte, PONG: 37 byte) and assuming for a each connection (GnuCon+OK) message pair 34 byte, this results in a total of 462 bytes. This traffic is necessary to merely explore the network. We can also observe in Figure 5.15, that several messages are not forwarded any further, because they are received for a second time.

If we further on assume that the node would start a search in this small network, this would result in 12 QUERY messages. Assuming that three nodes answer this QUERY, and this results in eight additional QUERYHIT messages, we can calculate a total traffic this node caused in this small network to 6.928 bytes. Together with the initialization traffic we can compute a total of 7.390 transmitted bytes. This is significantly more than the traffic caused by the Napster peer. For a larger network we can assume that the amount of traffic grows even further as the messages are flooded via more hops. The main reason is the distributed nature of the Gnutella network. This causes on the one hand a lot of traffic as no central lookup is available, but on the other hand also makes this network hard to attack, as no central single point of failure exists.

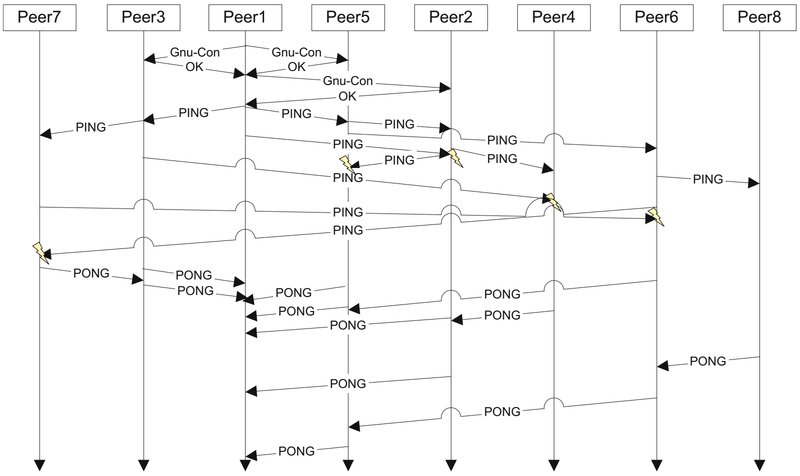


Fig. 5.15: Sample message sequence chart to illustrate the basic signaling behavior of Gnutella 04

Thus the amount of traffic caused by this application is high, although we only considered the traffic on the application layer . If we have a look at the topology of the Gnutella network on a geographical level, it turns out that the overlay topology differs significantly from the physical network, which results in zigzag routes, as depicted by Figure 5.16, Figure 5.17 and Figure 5.18. The route starts in New Mexico/USA. A PING or a QUERY message is sent in the first hop to other nodes located in the USA but also to a node located in Poland, i.e. the request is transmitted for the first time across the Atlantic (see Figure 5.16). Most of the connections of the node located in Poland lead directly back to the USA again. Thus in the second hop this message is transmitted e.g. to a node in California/USA and therefore crosses the Atlantic a second time (see Figure 5.17). In the third hop the message is then transmitted to a node located in Sweden (see Figure 5.18), resulting in a third transmission of the message across the Atlantic. Thus within three hops the message has been transmitted three times across the Atlantic, which results in the zigzag structure shown in Figure 5.18.

Every message routed/flooded in this overlay via the node in New Mexico has to be transmitted at least three times across the Atlantic, before it reaches its destination. The behavior depicted by Figure 5.16 to Figure 5.18 is only one example of a common behavior of the Gnutella topology.

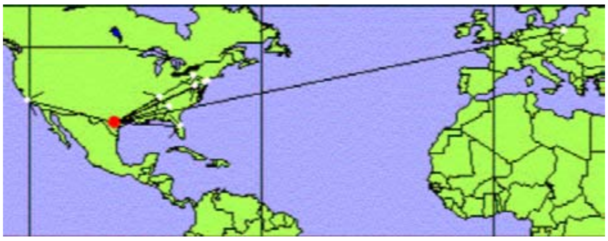


Fig. 5.16: Map of Gnutella Network measured on 12.08.2002 up to 1st hop level

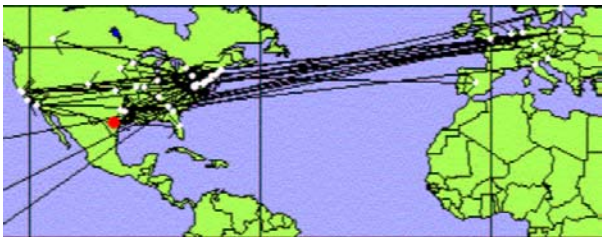


Fig. 5.17: Map of Gnutella Network measured on 12.08.2002 up to 2nd hop level

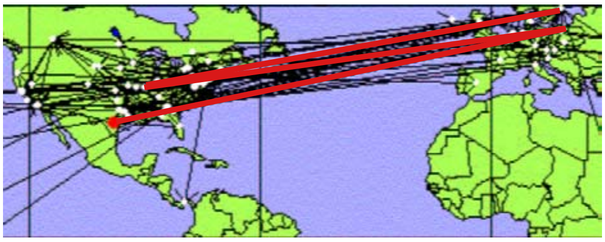


Fig. 5.18: Map of Gnutella Network measured on 12.08.2002 up to the 3rd hop level, including the zigzag PING-PONG route (bold line)

In addition to the unnecessary consumption of bandwidth between the USA and Europe, the zigzag routes cause high delays, which can be perceived by any user logged onto the overlay network. At least every third message crosses several times the Atlantic. The performance of the overlay network could certainly be improved, by directing the message first to nodes in the local proximity of the querying node and then only once across the Atlantic and similar long distances to then distribute the query in the respective local area.

A solution of this problem would be to adapt the virtual overlay to the physical network via cross-layer communication as e.g. proposed in [550]. Further on the large message sizes are also a concern. Here further compression of the signaling traffic can reduce the traffic significantly [424, 530, 585]. However another solution which is discussed in more detail in the next section is the introduction of a dynamic hierarchy, so that not every message has to be flooded through the whole network.

5.4 Hybrid Peer-to-Peer Networks

5.4.1 Basic Characteristics

As outlined above, hybrid Peer-to-Peer networks are characterized by the introduction of another dynamic hierarchical layer. As an example of such a hybrid Peer-to-Peer network we consider in this section the Gnutella 0.6 network.

A major goal of the Gnutella 0.6 architecture is to reduce the high message load, which can be observed in a Gnutella 0.4 network. Therefore several protocol enhancements have been proposed in [522], [523] resulting in the creation of a hierarchy in the network to establish a hub based network. These extensions are subsumed in the Gnutella protocol 0.6 [359]. The messages used in Gnutella 0.4 stay the same to guarantee downward compatibility. However, they are handled differently as explained below.

An efficient way to reduce the consumption of bandwidth is the introduction of hierarchies, as e.g. in Napster the Napster Server. To keep the advantages of Gnutella, i.e. the complete self organization and decentralization, Superpeers and Leafnodes are introduced in [563].

By introducing such enhancements, the load on the network can be reduced without introducing preconfigured, centralized servers. The network is still scalable, but one Superpeer should not have more than 50 to 100 Leafnodes, depending on the processing power and the connection of the Superpeer. Thus it is necessary, that the number of Superpeers increases according to the total number of leafnodes (peers) in the network.

A hybrid Peer-to-Peer network can not be modeled any further with a simple truncated powerlaw distribution. This can not reflect the Superpeers, which are characterized by a high degree and a significantly smaller number than simple peers. However the powerlaw degree distribution for the leafnodes is still expected to hold. Therefore we can assume a node degree distribution as given by equation 5.4.

$$p\left(d\right)=\begin{cases} c\cdot d^{-1.4},\ 1<d\leq 7\\ c\cdot 1^{-1.4}-0.05,\ d=1\\ c\cdot 0.05,\ d=20\\ 0,\ \mathit{in\ any\ other\ case}\end{cases}\quad ,\ \mathit{with}\ c=\left(\sum_{d}\frac{p(d)}{c}\right)^{-1} \tag{5.4}$$

$$\begin{aligned}&\mathit{average}:\ \bar{d}=2.8\\ &\operatorname{var}\left(d\right)=3.55\end{aligned}$$

Figure 5.19 depicts a sample network which is based on the Superpeer distribution stated above. Due to the nodes with degree 20 it has a hub-like structure, similar to the measured structure of a Gnutella 0.6 network (see Figure 5.20). These hubs dominate the structure of this overlay network. Because of their high degree these nodes establish with a higher probability connections between each other (marked by dashed lines in Figure 5.19). This results in a kind of second hierarchical layer which occurs in the Gnutella 0.6 network. The nodes with a small degree are mainly located at the edge of the network.

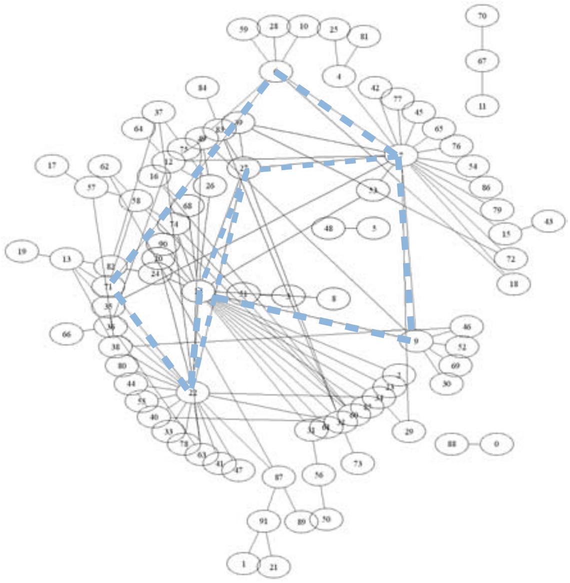


Fig. 5.19: Sample graph of a simulated Gnutella 0.6 network (100 nodes)

Although the number of nodes with degree one is high (47%) in this distribution, the number of separate sub-components is small, which can be observed by inspection. This results in a comparably high number of reachable nodes, within a few hops. This behavior can be explained by the fact, that the average degree of the Superpeer distribution with $\overline{d} = 2.80$ is higher than in the powerlaw distribution for a Gnutella 0.4 network used earlier.

If we transform the abstract network structure depicted by Figure 5.20 into the geographical view, depicted by Figure 5.21, we can make the network visible and can determine e.g. the popularity of the Gnutella network in different countries (see Figure 5.21). Further on we can observe the hub like structure of the Gnutella 0.6 network, which can not be retrieved from the geographical view. However, comparing both figures we can again observe the problem of the random structure, resulting in zigzag routes.

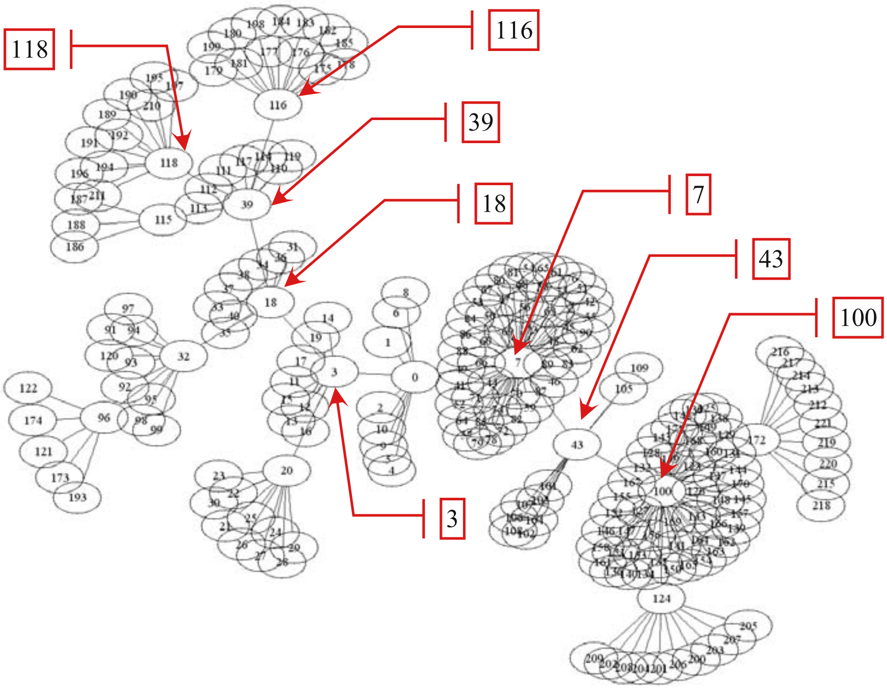


Fig. 5.20: Abstract network structure of a part of the Gnutella network (222 nodes Geographical view given by Figure 5.21, measured on 01.08.2002)

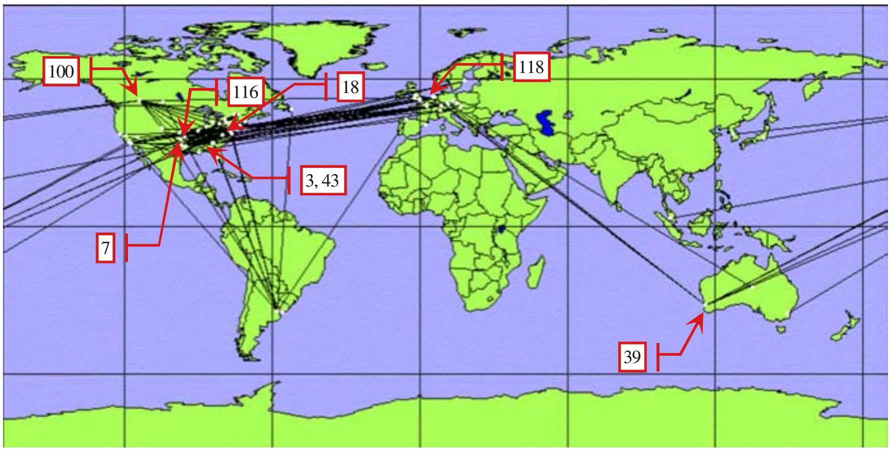


Fig. 5.21: Geographical view of a part of the Gnutella network (222 nodes); the numbers depict the node numbers from the abstract view (measured on 01.08.2002)

5.4.2 Signaling Characteristics

All messages, i.e. PING, PONG, QUERY and QUERYHIT, defined in Gnutella 0.4, are also used in Gnutella 0.6 . However, to reduce the traffic imposed on the Leafnodes and to use the Superpeer layer efficiently, the Leafnodes have to announce their shared content to the Superpeers they are connected to. Therefore the ROUTE_TABLE_UPDATE message (0x30) is used (see Figure 5.22 and Figure 5.23). The <Variant> parameter is used to identify a ROUTE_TABLE_UPDATE message either as Reset or as an Update.

The Reset variant is used to clear the route-table on the receiver, i.e. the Superpeer. Therefore additionally the table length (<Table_Length>) to be cleared must be stated. The parameter <Infinity> is not used currently and was intended to clear the route-table on several nodes, if the route-table would be broadcasted in the overlay.

The variant Patch is used to upload and set a new route-table at the Superpeer. To avoid one large table to be transferred at once, which might block the communication channel of a Gnutella node, it is possible to break one route table into a maximum of 255 chunks, which are numbered with the parameter <Seq_No>, where the maximum number of used chunks is stated with the parameter <Seq_Size>. To reduce the message size further on, the parameter <Compression> can be used to state a compression scheme which is used to compress the route table (0x0 for no algorithm, 0x1 for the ZLIB algorithm). For details of the decompression the parameter <Entry_Bits> is used, which is not explained in detail here. The parameter <DATA> contains 32 bit long hash-values of the keywords describing all objects shared by the

Leafnode. These values are concatenated to each other and transmitted as one data-string, or if necessary broken into smaller chunks, as mentioned above. The average message size of a ROUTE_TABLE_UPDATE is 269 byte.

The Superpeer uses the route tables, to decide which QUERY to forward to which Leafnode. Only in case that at least one keyword stated in the QUERY matches at least one entry in the route table of a Leafnode, the QUERY is forwarded to this Leafnode by the Superpeer.

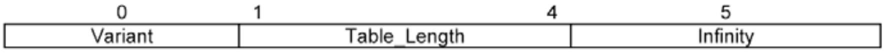


Fig. 5.22: ROUTE_TABLE_UPDATE (0x30) payload structure (Reset, Variant =0x0)

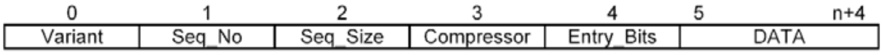


Fig. 5.23: ROUTE_TABLE_UPDATE (0x30) payload structure (Patch, Variant=0x1)

As mentioned above, Superpeers establish a higher hierarchy level, in which they form a pure Peer-to-Peer network, i.e. are connected to each other directly via TCP connections. To one Superpeer several Leafnodes are connected. The Superpeer shields its Leafnodes from the PING and PONG traffic. The Superpeer does not forward incoming PING messages to its Leafnodes. If a Superpeer receives a PING from one of its Leafnodes it answers with a series of previously received PONG messages from other Superpeers, so that the Leafnodes know which other Superpeers are currently available. Therefore the Superpeer also has to initialize PING messages in regular intervals in the Superpeer layer.

Further on Superpeers provide better QUERY routing functionalities by indexing the shared objects of all of their Leafnodes (with the ROUTE_TABLE_UPDATE). Thus the Superpeer forwards QUERY messages to all Superpeers, but only to those Leafnodes which announced to host content described with the same keywords, as given in the QUERY (except the one it received it from and if the TTL is not exceeded and it has not received the same message before). Additionally the Superpeer broadcasts the request in the Superpeer layer, to receive more results.

5.4.3 Discussion

To summarize the properties of a hybrid Peer-to-Peer network, we consider the example Gnutella 0.6 overlay network depicted by Figure 5.24. This figure shows three Superpeers S1 to S3 and seven Leafnodes L1 to L7, whereas node L1 just joined. We assume that the rest of the network is stable, i.e. all ROUTE_TABLE_UPDATE messages have been exchanged successfully between the Leafnodes and their corresponding Superpeers.

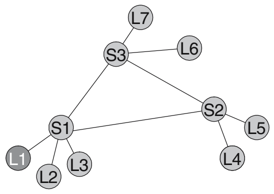


Fig. 5.24: Sample Gnutella 0.6 network

To join the network, node L1 first has to establish a TCP connection to node S1, whose IP address is assumed to be known by L1 (either from a previous session or from a bootstrap server). After it successfully established its TCP connection, node L1 performs the regular handshake with the Superpeer (see Figure 5.25). To announce its shared content to its Superpeer, node L1 sends a ROUTE_TABLE_UPDATE message (RTU) to S1, containing all keywords which describe the content shared by L1. Thus the content of L1 is available in the Gnutella 0.6 network. To be on the safe side, L1 additionally sends a PING message to S1, from which it receives in response three PONG messages announcing the presence of S1, S2 and S3. Thus L1 can still stay connected to the Gnutella 0.6 network, even when S1 fails. For illustration Figure 5.25, also shows how a PING message initiated by S1 is flooded in the Superpeer layer. This is completely in accordance with the Gnutella 0.4 protocol, except that the PING messages are not forwarded to the Leafnodes.

To illustrate further on the search behavior in a Gnutella 0.6 network we assume, that L1 searches an object, whose description matches objects shared by L3, L5 and L7. To initiate the search L1 sends a QUERY message containing the description of the requested object to S1. As only L2 announced to share an object matching the request, S1 forwards this QUERY only to L3. Additionally S1 floods the QUERY in the Superpeer layer, i.e. forwards it to S2 and S3. S2 and S3 also flood it further on in the Superpeer layer, i.e. S2 sends it to S3 and vice versa (assumption: S2 did not receive the QUERY before from S3 and S3 neither from S2). These two QUERY-messages are taken from the network and not flooded any further. However, as the routing table of L5 on S2 and the routing table of L7 on S3 result in a match upon a comparison with received request, S2 forwards the QUERY to L5 and S3 forwards it to L7. Upon receiving the QUERY, L3, L5 and L7 initiate each a QUERYHIT (QUHIT) message, which is then routed back on the shortest path through the overlay to node L1. Now L1 is able to establish a HTTP connection to L3, L5 or L7 to download the requested object.

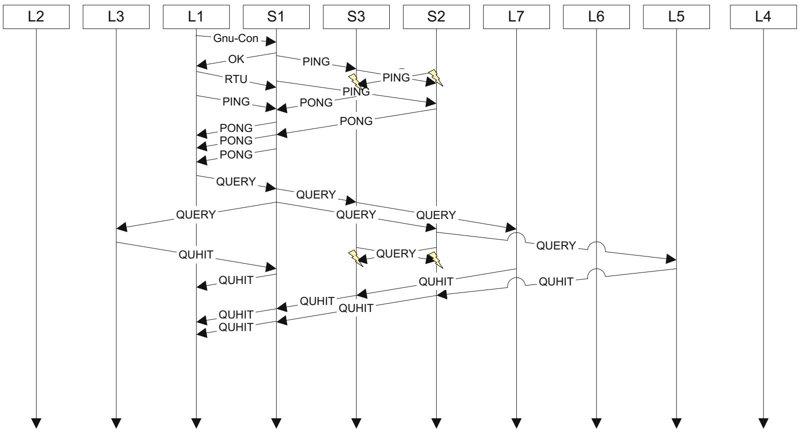


Fig. 5.25: Sample message sequence chart to illustrate the basic signaling behavior of Gnutella 0.6

If we calculate the traffic caused by one node, we again assume 34 byte for the connection message pair (GnuCon+OK). Further on we have to take into account the ROUTE_TABLE_UPDATE (RTU) message with 269 byte. Further on we have to take one seventh (as seven peers participate) of the PING/PONG traffic between the Superpeers into account, which results in our example in 4 PING messages and two PONG messages for every Superpeer, i.e. in total in 12 PING and 6 PONG messages. In addition for S1 we have to take into account three additional PONG messages and one PING message. This results in a total of approximately 508 byte. For the search traffic we also have to take into account one seventh of the previously exchanged RTU-messages, which results in 269 messages, i.e. one RTU message, which we took into account already above. Further on we have to take into account in this example eight QUERY messages (each with 78.4 byte) and eight QUERYHIT messages (each with 748.4 byte). This would result in a total of 6614 bytes for the search traffic. In total the traffic caused by this node can thus be computed to 7122 bytes. This is already less than the traffic caused by the Gnutella 0.4 network. However especially in large, i.e. more realistic, networks the advantage of hybrid Peer-to-Peer networks becomes more evident, as the amount of flooded messages is reduced significantly by the introduction of Superpeers.

Other protocols which establish a similar hierarchical overlay routing structure are edonkey2000 [410] and FastTrack. Applications like Kazaa [343] or Skype [567] and emule [191] or mldonkey [423] are based on these. In FastTrack the peers in the higher hierarchical level are called Superpeers and in edonkey2000 they are simply called servers. Both protocols are proprietary and therefore they are not officially and publicly documented. However a number of details of the two protocols have been re-engineered by users and creators of open source alternative applications [184, 188, 243, 358]. Already some basic measurements, mostly on IP-level, are available, concerning packet and data rates [43, 336, 600].