Unstructured P2P networks by example: Gnutella 0.4, Gnutella 0.6

Author Sebastian Ertel

Current topics in dependable distributed systems
Systems Engineering Group
Dresden University of Technology

Abstract

Owing to the growing popularity of peer-to-peer networks, this paper tries to give a little overview about one of the most popular protocols used in such dynamic environments, the Gnutella protocol. After a short introduction to the protocol internals, the main focus lies on showing the evolution of Gnutella from a decentralized to a hybrid peer-to-peer network protocol thereby focussing the attention on the properties scalability, reliability and flexibility.

1 Introduction

In the year 1999 the first and still widely known file sharing system napster appeared at the market. Napsters' main focus was on the distribution of multimedia files such as *.mp3 or *.wmv. In the following 2 years it gained a lot of new users all over the world and reached its peak at February 2001 with 26.4 million users worldwide. In the same year the music industry raised victoriously in an injunction at the Nineth Circuit Court forcing napster to shut down the whole network and paying $26 million settlement for the past. In the case of napster it was quite easy to switch off the whole network, since it was a centralized peer-to-peer network building on top of TCP/IP at the application level. Like in every P2P network the resources were distributed among the peers of the network. In order for one peer to be able to download data from some other peer, every node in the network played the role of a server as well as a client making the network highly flexible, dynamic and available. Those network participants are, because of their multiple personality, also refered to as *servents.* Although the file sharing was performed directly between two peers, lists about the participating systems and their provided files were kept among centralized servers. So the only thing to do to shutdown the whole napster network was pluging off one of the central servers, which finally happened in July 2001 marking the end of napsters' success story.

In 2000 when the first lawsuit against napster was issued by the band Metallica another peer-to-peer network protocol was introduced, Gnutella version 0.4. In contrast to the protocol used by napster, Gnutella was a file sharing protocol for decentralized P2P networks removing the single point of failure of centralized maintenance servers. The concept was to design a highly available network which contributes to the major design goals of peer-to-peer file sharing systems: flexibility, scalability, reliability and anonymity. This paper investigates the approaches of the Gnutella protocol in fullfilling these goals. After a short intoduction into the protocol version 0.4, a look at several case studies will give an insight into the overall behaviour of the network revealing several issues. In section 4 a quite different approach encountering the major problems of version 0.4 is depicted whose main concept can be found in the hybrid P2P structure of successor version 0.6. After checking upon the reliability by investigating several attacks against the network, section 6 will present some of the new features of the Gnutella version 0.6 protocol. A final discussion and summary will conclude this paper.

2 The Protocol

Once a Gnutella servent has obtained an IP address of another servent, either cached or determined with the help of a Gnutella Host Cache Server, he establishs a TCP/IP connection, the Gnutella network situated at the application level builds upon. Sending a message of the form

```
GNUTELLA CONNECT/<protocol version string>
\n\n
```

signals the remote servent the wish to join the Gnutella network. If the servent is willing to accept the connection the response message looks like

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

There are a couple of reasons why the remote servent could also refuse the connection establishment. One

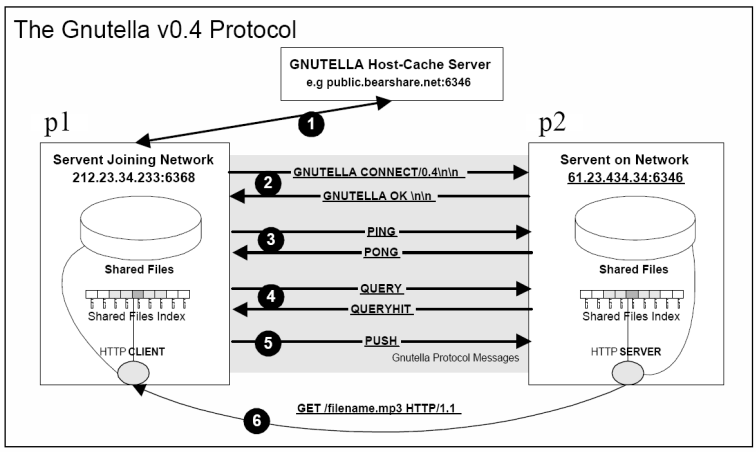

Figure 1: Gnutella Protocol version 0.4

is for instance too heavy load at the node or maybe he does not support the protocol version of the calling servent. In fact any other message than the above sent to the client indicates the failing. An overview of the internals of the protocol, that are subject to the following subchapters, was published in [7] and is shown in figure 1.

2.1 Descriptor headers

Once the connection has been established successfully every communication between the servents is done via so called Gnutella protocol descriptors.

For routing purposes each desciptor header consists of the following fields:

- *Descriptor ID* - the identifier of the sender (must not be the IP address because of anonymity), especially needed for routing the response messages
- *Payload Descriptor* - identifying the message type
- *TTL* - Time To Live field, indicating the maximum number of forwards until this message is removed from the network. Therefor it is decremented by every servent it passes until it reaches null.
- *Hops* - number of nodes passed; incremented by every servent passed
- *Payload Length* - length of this messages' descriptor immediately following. This is a very important field since it marks also the beginning of the next message in the input stream of a servent!

At this point two fields are mentioned again as they address special issues. As indicated before, the only way of avoiding a network flooding and the resulting poor bandwith is to trust every servent in computing the TTL field correctly. The other field is the payload length. Since Gnutella builds its network structure on the user level, it does not know anything about packets. So servent synchronization of the input stream has to be managed using the value of this field in a very strict way.

2.2 Message types

Following the desciptor header is the actual message. In the Gnutella Protocol v0.4 there exist mainly 5 message types that can be further classified according to their purpose.

- *Finding friends:* A servent can start a search for other servents throughout the Gnutella network by sending a simple *PING* message. Every servent who receives such a message might respond to it with one or more *PONG* messages including not only an IP address and port, but also the number of files and kilobytes to be shared.
- *Resource retrieval:* To search the network for a certain data, a *QUERY* message is used including a value representing the minimal speed a responding servent should offer and the search criteria, to be queried for, itself. Servents that receive such a message and possess data, e.g. files, meeting strictly the specified search criteria, should respond to it with a QUERY HIT message. Additionally to the network address information and the number of hits, also a result set of file identifiers consisting of index, size and name is embedded in the payload part of this message.
- *Pushing data:* *PUSH* messages have a certain purpose in case of firewalled servents and will therefor be described in more detail in the following section.

Every actual data transmission between two servents is performed outside of the Gnutella network via a direct TCP/IP connection over HTTP using the parameters obtained by the former QUERY request.

```
GET /get/<File Index>/<File Name>/ HTTP/1.0/
\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
User-Agent: Gnutella\r\n
```

Using the range field in the HTTP request makes it also possible to reestablish an aborted transmission at the point where it has been interrupted.

2.3 Firewalls

A problem occurs when the servent, sharing a ressource, is situated behind a firewall that blocks incoming connections to its Gnutella port. For those situations Gnutella provides an opportunity for the requesting client to force its desired communication partner to "push" the data to its machine. This is accomplished by sending a PUSH message, indicating that he needs to be the initiator of the TCP/IP connection. On receipt of such a message the servent should try to a establish a connection to the requesting node, identified by the IP address and port included in the PUSH. A failure implies, the node requesting the file is also behind a firewall. In such a case a connection in the context of the Gnutella Protocol Specifiction is not possible. On success the preceeding conversation works almost like normal via HTTP.

2.4 Routing

PING and QUERY messages are broadcast messages that are flooded to every node in the network except the sender until their TTL field expires. Every resulting PONG/QUERY HIT message is supposed to have the same Descriptor_Id in the Descriptor header as the corresponding PING/QUERY message it responds to. In the case where a servent receives a PONG without having seen a PING, it will remove the message from the network by not forwarding it to any other servent. In contrast to PONG or QUERY HIT messages, Push messages are routed by the Servent_Identifier field and not by the Descriptor_Id included in the header. Like described before it is crucial to the flow control of the network that every servent increments the Hops field and decrements the TTL field of a descriptor header appropriately. One important fact, to keep in mind, is that PONG and QueryHit messages are routed at the same path like the incoming PING and Query messages. This will, as shown in one of the next sections, contributed to a very serious security issue.

3 Studying the Gnutella network

To spy the Gnutella network a servent joins the network and gathers information by using the protocol without violating its means. Those network participants are often also refered to as *network crawlers.* With the help of such techniques typical subjects of interest, like the caused traffic, the distribution of data, the network structure as well as the behaviour of the network itself, could be studied in the passed couple of years.

3.1 Gnutella traffic

In [3] one of the main reasons for the deficit in scalability of version 0.4 had been discovered. As a consequence of the, until then, very naiv approach concerning the connection maintenance, nearly 55% of the generated traffic was caused by PING/PONG message and only 35% was user generated as shown in figure 2 taken out of [8].

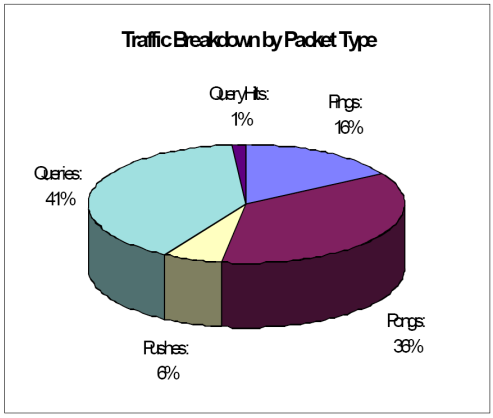

Figure 2: Gnutella network traffic in protocol version 0.4

Furthermore it turned out that most of the queries did not result in a positive response of the servents, leading to nothing else but traffic overhead(see figure 3 observed in [8]). Responsible for that could have been again the very inflexible protocol definition which says: [*"A servent should only reply to a Query with a QueryHit if it contains data that strictly meets the Query Search Criteria."*] [1]

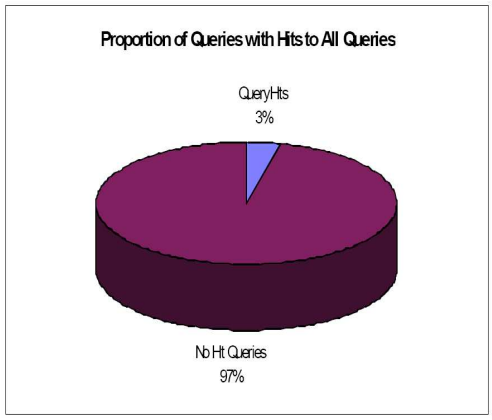

Figure 3: Queries with QueryHit message compared to all queries

Figure 4, published in [8], contributes to that assumption by showing the number of Hits sent as an reply to a successful query message.

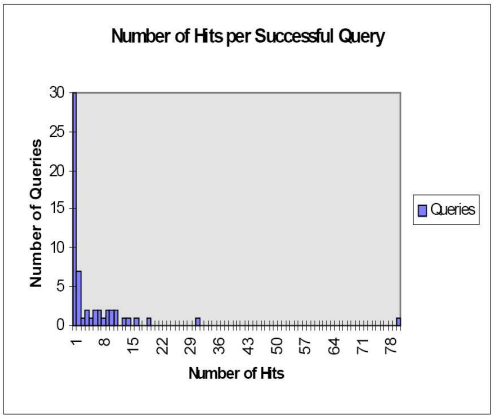

Figure 4: Number of responding QueryHits belonging to one Query message

Those difficulties vanished when the new version of the protocol was introduced in the middle of 2001. The result: almost 92% QUERY messages and only 8% protocol generated traffic led to a big increase of the Gnutella user community.

3.2 Network structure

Since the TTL values can be set by the user in several applications figure 5, that was also published in [8], gives an indication of how far the search of the individual messages expands.

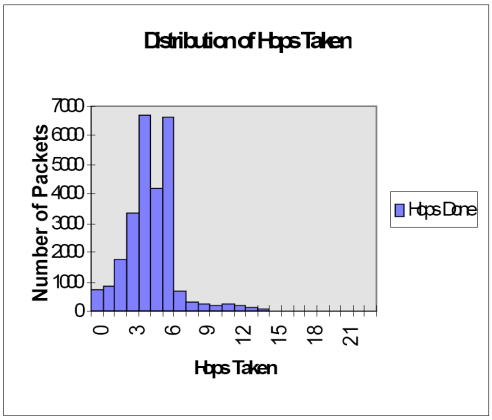

Figure 5: Hop count of all packets

Surprisingly most of the packets had an TTL of 15 or less, but it had also been observed that there were some servents trying to receive a brighter amount of peers by increasing the value around 20 (figure 6; appeared in [8]).

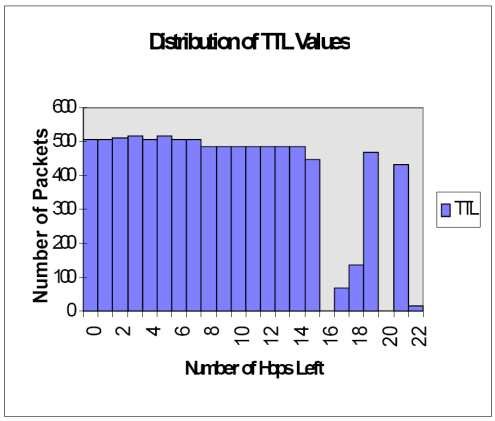

Figure 6: TTL values of all packets

It had also been observed that the shared data is not evenly distributed. It turned out that a big amount of the data as well as the connectivity was situated among just a small group of peers which of cause put a hit according to the reliability of the network. So 71% of the files shared were provided by 10% of the nodes. The authors of [3] made "free riders", servents that do not share data, responsible for this. Towards version 0.6 several improvements were introduced to encounter that issue, including a default sharing directory and the dropping of all HTTP messages having a different application entry in its User Agent field. According to that, there has been a major shift to nodes using Gnutella protocol version 0.6 . Additionally it seems more likely that session last longer when the size of the network is rather small. This accounts to the fact of increased operability of the individual hosts, which now spend much less time on changing their connections as before. Concerning the change of the structure according to the daytime, it is quite intuitive that the network is very busy in the evening hours with the people returning home from work as depicted in figure 7 from [3]. The same behaviour could be observed for certain weekdays. So the most nodes, joining the network, were registered on the weekend.

4 The Query/Advertise approach

Before several enhancements of version 0.6 are discussed in one of the following sections, one completely different approach in improving some of the network's properties shall be mentioned. The idea of [4] is to use a publish/subscribe middleware to encounter the existing Gnutella problems. A publish/subscribe network consists of subscribers, which are nodes interested in certain kinds of events, and publishers, that in turn distribute events asynchronously to the nodes. For a client to be informed of a certain event, he has to subscribe itself to one or more subscribers defining exactly which type of event he is interested in. Therefor the most important requirement is a highly structured

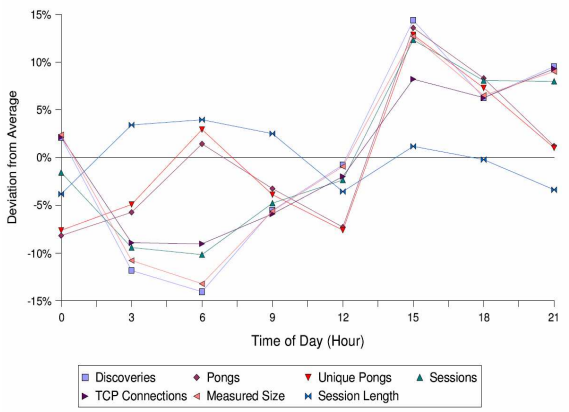

Figure 7: Network structure over the day

content with respect to the subscribe message. This is achieved by the creation of so called filter patterns, that typically consist of attribute-value pairs and may have the following structure:

```
{(director, "Michael Mann")(title, "Heat")
(Special Edition, True)}
```

As seen in the example, the type of a value can be integer, double, string or date. The final routing is done by an underlying content based algorithm, operating via some peer-to-peer-interconnected server nodes. This algorithm is responsible for routing messages to the clients whose subscription exactly matches the appeared event.

4.1 Flaws of the Gnutella network

Before proceeding it is necessary to identify some of the flaws of the Gnutella network with respect to the defined design goals anounced above, excluding security issues since those are subject of the next section. One of the largest goals to be achieved in a peer-to-peer network according to the user is surely anonymity. In the Gnutella protocol this is accomplished by using the servent identifier field in the description header for routing purposes, but it is absolutely not hard to track the IP of one servent by just using simple network sniffing tools. Another main point of attack is a malicious user. Since every servent routes packets of other network participants it is very easy for an attacker to harm the core of network. The following section will delve into such an attack in more detail. As it was explained in the section about the network charateristica, especially for users of version 0.4 efficiency in terms of network traffic is still a big problem. Those nodes may easily become a serious bottleneck in the network. The lack of standardization concerning the query strings in the protocol specification additionally causes issues, hence every node is priviledged to interpret them without any limitations. This may cause a lot of misunderstandings between the peers and leads to additional message overhead.

4.2 Using Publish/Subscribe services

The proposed query/advertise system as introduced in [4] consists of three global concepts: *advertisments*, *queries* and *responses* on top of a middleware publish/subscribe architecture. Advertisments are used by the clients to tell the servers what kind of resource they are willing to share by using filter patterns. Those are send to a server where they are forwarded among several other servers in the network. If a client is now looking for a resource he injects a message into the Q/A system describing exactly what he is looking for. What he receives in return are Responses of every advertiser with a match, including detailed information about the resource.

4.3 ... in the end

It is obvious that there are a couple of advantages to this approach antagonizing the flaws described. In terms of anonymity there is a great benefit, since the complete routing is done by the servers. So it is not necessary for the clients to have any information further about other peers in the network than the ones requested by a query. In fact the only participant who gets to know the real identity of a client is the server that recieves the advertisment. Every forwarding is done by a set of characteristics of the client that in reverse do not reveal its identity. When it comes to efficiency, this approach is very hard to beat thanks to the removal of the network flooding caused by the Query messages. Every server knows exactly where to route the queries according to the forwarded advertisments he received from the other servers. Another adavantage in the security issue of malicious users is the contamination of those on the first communication level between the client and the server. The standardization of the query messages by the underlying Publish/Subscribe system avoids misunderstandings between the peers and therewith reduces the traffic overhead in the network significantly. But nevertheless there are some major aspects that will prevent this approach from replacing the old fashioned approach from Gnutella. The weightiest reason for that lies in the network structure. It violates the Gnutella idea of a peer-to-peer network where every node act in a client as well as in a server role. This makes the Gnutella network so extremely flexible. In the Q/A system the programs running on the client machines differ from those on the servers. Additionally the resource retrieval is also a problem of the descibed design, since exchanging IP addresses would decrease the anonymity goal. Otherwise sending it via the Query/Advertise network would fail the efficiency property. A solution proposed by the author is to use a third party to manage the resource exchange. But what if the advertise servers,

which are the only ones worth considering, are more than 1 hop away from each other? One last disadvantage to be mentioned here is the caching. As no client is involved in the routing algorithm the caching of very frequently requested files is not quite easy to accomplish and includes major changes in the design. There are some other further research issues labeled in [4] which also contribute to the fact that this approach remains rather theoretical than practical.

5 Attacking the network

In the former section it was stated that the Gnutella network is lacking of some serious security issues concerning not only the anonymity of the peers but also the vunerability of individuals. The major reason for that lies in the Trust Model of Gnutella which bases on the hope of well-behaving users rather than on mechanisms to prevent attacks. This chapter will give proof to that argument by describing two attacks and their effects in more detail.

5.1 The QueryHit attack

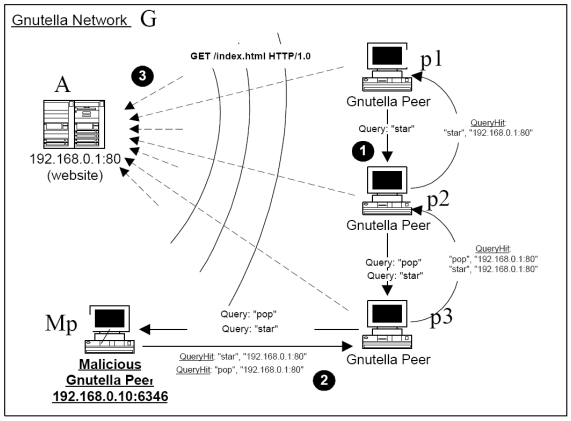

Figure 8: A Distributed Denial of Service attack exploiting the QueryHit messages in Gnutella protocol version 0.4

The most effective Distributed Denial of Service attack, with the view of an attacker, is the so called QueryHit attack where, as the name indicates, a security leak in the QueryHit messages is exploited. The principle is rather simple and works just as email spam. So a peer might "lie" according to an incoming Query. This is possible because the Gnutella Protocol does not set any limit on the information included in an QueryHit message thereby making several Caching strategies applicable. The attacker, that wants to run a DDoS attack against a globally known webserver like for instance cnn.com, might use this loophole by replying with the address of this server and a file that is very likely to be found there, for example "index.html", thereby making every poluted peer an active part in this attack. Figure 8 taken from [7] shows how this attack works. As explained a peer might send a request for any resource (step 1). This Query is forwarded by other peers in the network until it reaches the attacker which answers with a QueryHit including the IP of the victim (step 2). Mind that this does not have to be a participant of the Gnutella network since the HTTP request issued in step 3 by the nodes aiming to download the ressource is performed outside of the network. In the following there are a couple of tricks to make this attack even more efficient. To make the resource of the attacker as attractive to the servents as possible by for example offering a fast LAN connection and naming the resource as similar to a Query as possible as well as having it end with a very popular suffix like ".mp3" or ".avi" is self-evident. This will result in a very high ranking among the Gnutella clients such as BearShare or LimeWire. An additional way in poluting as many peers as possible is to connect to a Gnutella IP Host Cache server and retrieve a random number of peers. After providing them with spammed QueryHit messages, the attacker could just repeat this procedure by reconnecting again to the Host Cache server using an other random number (figure 9 appeared in [7]). An important fact, contributing to this attack, is that a peer that could not succeed in downloading the desired resource will retry automatically after a certain timeout which even increases the load on the attacked server. A quite interesting fact is that this kind of DDoS attack is very often used for commercial purposes also, by just leading to a certain website. As a result of the experiment described in [7] not only the attacked Apache Webserver but also the shielding Zonealarm firewall collapsed under the high load. As for traceability, since Gnutella does not possess any auditing system it is impossible to find out who was the originator of the attack. Even more concerning is the fact that it is theoretically not possible to stop such an attack because of the nature of the network itself. To prevent such Distributed Denial of Service attacks, the authors of [7] suggest a very simple as well as efficient solution using an additional handshake sequence between the server and the client before the final TCP/IP connection, for downloading the file, is established.

5.2 The Pong attack

A second kind of DDoS attack, that operates in a quite similar way as the former introduced, is the Pong attack. Instead of QueryHit, Pong messages are used to route the traffic to a peer that is the traget of an attack. This is achieved by responding to a PING message with an IP and portnumber of another peer (step 1 and 2 of figure 10; published in [7]). As a result the poluted peer (p1) will route all its Query traffic to the attacked system (step 3). For fairness sake one has to mention that this attack is not even close as dan-

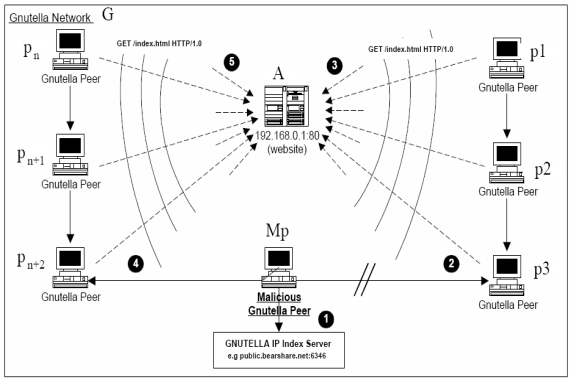

Figure 9: Maximizing the number of spammed peers by using a Gnutella host cache server

gerous as the QueryHit attack because every servent in the Gnutella network reissues PING messages periodically after a certain time interval to discover new peers and eliminate the old ones out of its cache. So this attack will terminate after the spammed peer has send his next PING broadcast and because A is not part of the Gnutella network it will be removed out of the cache of p1.

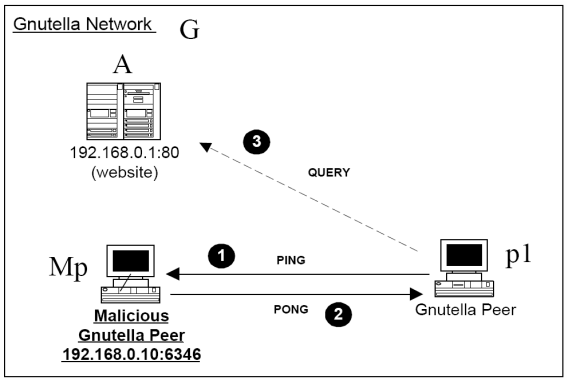

Figure 10: A Distributed Denial of Service attack exploiting the Ping messages in Gnutella protocol version 0.4

5.3 Going even one step further

As it has been noted in one of the previous sections, anonymity is a serious issue of the Gnutella protocol since it is very easy to discover the IP addresses of the peers. This makes it much easier for an attacker than dealing with highly secured Internet systems that may even use dial-up connections or DHCP to receive their IPs. As a consequence especially hosts with a static IP are very vulnerable, for example users of the ".edu" domain. Another minor weakness of the Gnutella protocol is the PUSH message and the possibility of using it to inject a virus into the system. This can be achieved by just lying to any arriving Query message by pointing to a file that does not exist. Since upon receive of a PUSH the requesting peer tries to establish a connection with it accepting any file send, a virus can be intruded. It has to be mentioned that this is not as easy as it sounds, because some Gnutella clients like LimeWire actually do a filtering of potentially dangerous file types like ".exe" or ".vbs". To summarize this section, Gnutella offers a lot of possiblities for attacking the whole network system not only restricted to the Gnutella network itself. So a user should at least be aware of these security leaks when entering the Gnutella network and maybe take some precautions that help to minimize the success of attacks such as reconnecting from time to time.

6 Extensions of Version 0.6

Like shown in the former chapters the Gnutella protocol version 0.4 had a lot of issues making it hard to scale. Indeed there exists a mathematical proof in [6] concluding that Gnutella can't scale. This paper was published in February 2001. In the middle of 2001 a new version of the Gnutella protocol appeared on the market introducing major extensions while the main focus was on increasing the efficiency of the network and therewith also the availability. In the following I will cover only the most important ones. For further knowledge please have a look at [2].

6.1 Protocol extensions

The version 0.6 protocol itself possess two improvements with markable impact on both, the structure as well as the traffic of the network. Both of them making the network more dependend on the capabibilities of the individual peers.

6.1.1 Connection initialization

Protocol version 0.4 introduced a very simple approach in connection establishment. Neither could the peer, entering the Gnutella network, choose whether it wants to be connected to this certain node nor does the peer, to be connected to, knew anything about the new clients abilities. The final decision about a successful connection setup always resided with the peer the new servent elected as entrance into the network. To avoid this kind of unfairness a handshaking sequence is initiated after establishing a TCP/IP connection. A new peer with the intension to enter the Gnutella network sends a header containing not only the connection request but also additional information about its capabilities.

```
GNUTELLA CONNECT/0.6
User-Agent: BearShare/1.0
Pong-Caching: 0.1
GGEP: 0.5
```

On receive of such a message the remote servent can base its decision whether to support this request or not. If he does a positive reply is sent to the new servent including additional charaterizing information of the server.

```
GNUTELLA/0.6 200 OK
User-Agent: BearShare/1.0
Pong-Caching: 0.1
GGEP: 0.5
Private-Data: 5ef89a
```

So, for example, the client might only want to connect to peers using BearShare applications. Thus in the running example he successfully finishes the handshake sequence and establishes the final connection.

```
GNUTELLA/0.6 200 OK
Private-Data: a04fce
```

According to the specification it is strictly recommended to use standard HTTP headers. The usage of own headers is also possible but they have to confirm the syntax of HTTP.

6.1.2 X-Try headers

In the former protocol version a rejection of a connection setup was performed by any response different from the above shown. The refused client than had to look for other peers in the network willing to open a connection with him. In version 0.6 the strategy of X-Try headers is introduced. So in case, for whatever reason, a servent may reject the connection to an entering host it must, as possible, supply him with other information of other peers in the network that may be willing to accept his request. Therefore a so called X-Try header is that consists of a list of IP host addresses and appendant port number.

```
X-Try: 1.2.3.4:1234, 5.6.7.8:5678
```

This reduces not only the effort of a client to find a communication partner but also the message overhead in the network. Another new technique contributing to this goal and also added to this new specification is the caching of Pong messages among the peers.

6.2 Restructuring the network

Those optimizations to the protocol itself involved several changes to the structure of the network. The main benefit out of it was scalability.

6.2.1 The ultrapeer system

As it is described in the case study section, the network structure evolved in terms of a so called power law, meaning the existence of just a couple of powerful peers with a high amount of network data and connectivity. The idea, the new approach is based upon, can be seen as direct result of these meassurements. It kind of adapts the concept of the explained Query/Advertise architecture to the needs of a peer-to-peer network. So the Ultrapeer system developed for specification 0.6 builds a hierarchical network structure with Ultrapeers and Leaves. The latter ones only have connections to Ultrapeer nodes while those on the other hand act as a kind of proxy. An ultrapeer is connected to further ultrapeers and only forwards a query to a leaf if it "thinks" the leaf node is able to generate a query hit. For a servent to become ultrapeer capable some requirements like the property of free internet access (not firewalled), a suitable operating system, sufficient bandwith, uptime, RAM and CPU speed have to be fullfilled. Whether he becomes one or not is decided by the network itself and according to the need of another ultrapeer. This is negotiated in the ultrapeer handshaking sequence. A normal connection setup starts with a message of the peer saying that it wants to join the network as a leaf.

```
GNUTELLA CONNECT/0.6
User-Agent: LimWire/1.0
X-Ultrapeer: False
X-Query-Routing: 0.1
```

The requested ultrapeer may respond with a positive answer.

```
GNUTELLA/0.6 200 OK
User-Agent: LimWire/1.0
X-Ultrapeer: False
X-Ultrapeer-Needed: False
X-Query-Routing: 0.1
X-Try: 24.37.144:6346, 193.205.63.22:6346
X-Try-Ultrapeers: 23.35.1.7:6346,
18.207.63.25:6347
```

[1] The field X-Ultrapeer-Needed is used to make the above explained decision in case a new ultrapeer capable node enters the network. Included in the reply are also references to other ultrapeers since the final decision still resides on the client side. In the running example the client becomes a new shielded leaf of the ultrapeer by just sending an HTTP OK message.

```
GNUTELLA/0.6 200 OK
```

To also provide backward compatibility, a peer, incapable of supporting the ultrapeer system, just behaves to the guidelines of the former specification. Examples for other szenarios are listed in [2]

6.2.2 Query routing protocol

The query forwarding inside the ultrapeers is taken care of by the new Query Routing Protocol. As noted

[1]This is an example taken out of [2] that, in my opinion includes two mistakes. At first the value in the X-Ultrapeer field of the ultrapeer response should be "True". Second the first IP address given in the X-Try field is lacking of a further ...

in the specification [*"The aim of the QRP is to avoid forwarding a query that cannot match, it is not to forward only those queries that will match."*][2] In doing so every ultrapeer receives a hash table from its leaf nodes with all the query words potentially supported. So on a receive of a query the ultrapeer can process the routing without having any look at the actual resource by breaking the query into indivdual words and just sneaking throw this big query routing table. For the preparation of the hashed information every leaf node breaks its resource names into indivdual words. It also does elemination of plurals by just tailoring the last two or three letters. After that the ASCII words are hashed and the present flag is set. Finally after applying an optional compression step the data is split into small pieces and shipped with the regular Gnutella messages to the ultrapeers. Please mind that those arrays can become very so only the hash and a flag are written into the table and not the word itself.

6.3 Further extensions

Another feature of version 0.6 is HUGE which provides hash functions needed for building up the query routing tables. Additionally HUGE extends the file download between the peers to make remote peers aware of location changes of files, for example. The new protocol specification even consists of a pong caching scheme to reduce the traffic in exploring the network.

7 Summary

This report shows the evolution of the Gnutella protocol design from an unstructured to a structured peer-to-peer network. Gnutella was able to handle the dynamic nature of P2P environments in a scalable manner. The efficiency of the discovery of data among the network was improved further in the version 0.6. Nevertheless there are two important aspects that Gnutella did not accomplished very well. The first is the content distribution over the network. As explained in chapter 3 the content as well as the traffic is concentrated at a few nodes with very special qualities. The second point where the Gnutella protocol fails is in providing a concrete answer to a search request. Because of the flooding that is limited by the TTL values of the messages, the Gnutella network responds with 0 results in case no QueryHit message to a corresponding Query has been received. In the context of the network this does not mean that the resource is not available at a node. It rather says that the resource did not exist among the queried peers until the TTL expired. These two aspects of content distribution and search are definitely something that a designer of such a protocol would like to have control of in order to make it as efficient as possible. A solution is provided by the Distributed Hash Function of the centralized peer-to-peer approach that marks the next generation of P2P networks.

References

[1] The Gnutella Protocol Specification v0.4 Document Revision 1.2.

[2] Gnutella Protocol Specification v0.6.

[3] Matei Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network.

[4] Dennis Heimbigner. Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality.

[5] David G. Deschenes, Scott D. Weber and Brian D. Davison. Crawling Gnutella: Lessons Learned. figures: 7

[6] Jordan Ritter "Why Gnutella can't scale. No, really." February 2001.

[7] Demetrios Zeinalipour-Yazti "Exploiting the Security Weaknesses of the Gnutella Protocol". figures: 1, 8, 9, 10

[8] Kelsey Anderson "Analysis of the Traffic on the Gnutella Network" March 2001. figures: 2, 3, 4, 5, 6