Gnutella: Distributed System for Information Storage and Searching Model Description

Fernando R. A. Bordignon, Gabriel H. Tolosa
bordi@unlu.edu.ar, tolosoft@unlu.edu.ar
División Estadística y Sistemas
Departamento de Ciencias Básicas
Universidad Nacional de Luján

Abstract

This document describes a new application protocol named Gnutella, which is destined to information storage and searching in distributed environments, working with the peer-to-peer network concept.

Due to Gnutella was born as an end-user application program (and there's not a formal protocol specification). This paper pretends to formalize its operation modes and data structures to develop application programs based on this cooperative working model.

Introduction

Most Internet services are implemented in client/server processing model. Unlike this high-centralized scheme, Gnutella defines a network at the application layer, with the peer-to-peer network-working concept, where a node works as client and server at the same time. Conceptually, it's a distributed system for information storage and searching.

To understand the features and the state of development of the protocol, it turns out necessary to be sent to its origins. Gnutella arose like an independent project from the programmers Justin Frankel and Tom Pepper, founders of the Nullsoft company, outstanding to be the owner of the Winamp product (mp3 audio format player). On March 14, year 2000, in the Slashdot web site (http://www.slashdot.com) was published, in beta phase, a version of the Gnutella program. This program was described as "a software tool to file sharing, that can be more powerful than Napster". Immediately the page was put out of service, but too many copies were unloaded. Nullsoft never published the software again.

At the moment, a community of programmers carries out a continuation to the project. On the basis of techniques of reverse engineer and protocols analysis they decodificate and made several implementations of the Gnutella protocol. The main site of distribution is http://gnutella.wego.com.

It is important to emphasize two fundamental characteristics of Gnutella so that investigation groups and companies made studies: its decentralized feature and its cooperative spirit. From that, diverse projects based on the peer-to-peer model in wideworld networks are been developed [CLARK] [NAPSTER] [SX].

Although the actual protocol version is the 0.4, it's working in the new generation of it, studying the possibility of optimizations to the routing mechanics (message propagation), recovery and queries specification. The division Distributed Search Services of Clip2 company (http://dss.clip2.com) carried out studies on the evolution and present condition of the Gnutella network [DSS1] and the area Internet Ecologies of the Xerox Company made an analysis of the traffic in the network, describing the users behavior [ADAR].

Emergent Distributed Systems for File Sharing

A Distributed System for File Sharing (DSFS) allows to remote users – which operate in distributed machines - can interchange files. If - in this model - each machine can offer a portion of its own file system or ask for a remote file (each machine is equal in functionality), the system is considered like peer-to-peer. This allows the development of new applications and services based on end users.

The advance in this way will let the optimization of computer resources. For example, a company with an important number of computers can combine them to harness its availability of computing resources (storage space and CPU use) to carry out a certain task.

DSFS Examples:

Freenet [CLARKE] is a peer-to-peer network designed to allow information distribution on the Internet in an efficient way. It operates in decentralized way and it provides an important degree of anonymity to the users. In Freenet does not exist machines that offers centralized services and that its crash can attempt against the stability of the network. It uses propagation and cache policies better than Gnutella's. It implements replication strategies of frequently acceded archives, which allows to establish alternative routing towards mirrors with the objective of regulating the traffic and obtaining a more efficient information downloading.

Napster [NAPSTER] is a system to share mp3 files with centralized database of resources and users. A central server joins requests of the clients with its possible results. Once a client selects a file to download (from a listing given back by the central server) the client opens a connection with the remote machine that contains the file. The Napster central server has not stored files.

Scour Exchange [SX] is a system similar to Napster, except for it is not limited to operate with mp3 files. It let users to query and to recover information located in storage units of remote users.

Features of Gnutella

Gnutella works in a distributed environment model. An gnode X (host running Gnutella software) offers the files that wishes to share with other users, simultaneously the user owner of this gnode Y can be recovering, as client, files from gnode X. Usually, the server and client features are always active in a Gnutella node, but nothing prevents that some can not be activated.

Gnutella network is made up of a great number of gnodes distributed throughout the world. Its topology does not indicate hierarchy some, since each gnode is equal to the others in functionality. One gnode feature is the offered bandwidth for downloading files, the model undergoes certain variations. Those gnodes of greater bandwidth are preferred to others, making hubs or central ring of connection to the network.

Each Gnutella gnode only knows about the gnodes with which it directly connects. The other gnodes are invisible, unless they announce themselves answering a message like ping or answering a query. This feature provides a high degree of anonymity.

Of summarized form, the Gnutella network operates under the model known like "viral propagation". A client sends a message to a gnode, and this one send it again to all gnodes to which it is connected. Of this form, with single network address of a connected gnode, someone can enter to the network.

The access to the network is made indicating IP address and TCP port of any gnode that already is connected. The Gnode that initiates the connection announces its presence by sending a message. The Gnode to which it was connected send this message again to all the gnodes to which it is directly connected, and so on. Each one of these gnodes replies this message indicating how many archives share and total size of those.

An example of the high degree of stability and independence that presents this model can be seen in the following case: Suppose that gnode-1 connects gnode-2, all it offers gnode-1 can be taken by gnode-2 and visceversa. Now gnode-3 connects to gnode-2 and this one informs what gnodes it is directly connected (the messages of gnode-3 can arrive at gnode-1 since gnode-2 send them again). Next, gnode-2 crashes (or is out of service). The network continues working normally since gnode-3 previously has alternative network addresses of gnodes (given by gnode-2), that will allow him to connect itself to alternative gnodes to follow into the network.

In Gnutella implementations a parameter to define is the amount of simultaneous connections that a gnode must try to maintain. At the most connections have, a greater query space will be obtained and it will give a high availability of the presence into the network.

Message Structure

A gnutella message is transported on TCP protocol. Its head always have 23 bytes with the following fields: gnode identifier (16 bytes), function (1 byte), TTL or Time To Live (1 byte), Hops (1 byte) and payload length in bytes (4 bytes). The function field indicates the actions to do by gnodes that receive the message. The head structure is fixed for any type of function, but the second part (or payload) depends of the function type.

The actual version of Gnutella operates with five functions [DSS2] that will allow staying the network, to query by resources and to download files.

1. **Ping Function (code 0x00)**

   It is used by gnodes to announce its presence in the network. It does not use message payload, so that the payload length field is set to zero. Each gnode that receives a Ping message must respond generating Pong messages (they can be n) where announce its own presence or tell about gnodes that it knows. A Gnode will generate a Ping, in addition, to obtain information on neighboring gnodes.

   Example. Ping message

```
27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
00 07 00 00 00 00 00
```

Message

Gnode identifier	: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
Function	: 00
TTL	: 07
Hops	: 00
Payload length	: 00 00 00 00

2. **Pong Function (code 0x01)**

   A Gnode sends a Pong message like an answer to a Ping message. The payload of a Pong is made up of the following fields: port (2 bytes), IP address (4 bytes), number of shared files (4 bytes) and Kbytes shared (4 bytes). It's interesting to emphasize that a gnode when receiving data like this can open new connections that allow it to extend their action field and, simultaneously, to give a greater stability to its presence in the Gnutella network. A Gnode can send more than a Pong message like answer to a single Ping, since it informs about other gnodes that it has stored in its internal caches.

   Example. Pong message

```
27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
01 07 00 0E 00 00 00 CA 18 AA D2 62 95 03 00 00
00 00 00 00 00
```

Message

Gnode identifier	: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
Function	: 01
TTL	: 07
Hops	: 00
Payload length	: 0E 00 00 00

Payload

Port	: CA 18
IP address	: AA D2 62 85
Number of Shared files	: 03 00 00 00
Shared Kbytes	: 00 00 00 00

3. **Push Function (code 0x40)**

   It is used by gnodes that access the network crossing a firewall system, and cannot download a resource located into a gnode at the other side of that. A Gnode that requires the resource will send a Push message to gnode which has it, with the following fields: gnode identifier (16 bytes), index (4 bytes), IP address (4 bytes) and port (2 bytes). Once obtained the message, gnode server will establish a connection with the client gnode and it will send to it, using the IP address and port given. An answer to the Push function is not defined, since a gnode can send the resource or not to answer the request.

   Example. Push message

```
27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
40 07 00 1A 00 00 00 22 11 3F 71 49 B2 D4 11 88
```

```
                      23 00 80 AD 40 22 11 02 00 00 00 AA 12 62 AA CA
                      22
```

Message	
Gnode identifier	: 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
Function	: 40
TTL	: 07
Hops	: 00
Payload length	: 1A 00 00 00
Payload	
Gnode identifier	: 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11
Index	: 02 00 00 00
IP address	: AA 12 62 AA
Port	: CA 22

4. Query Function (code 0x80)

It is used by a gnode to query other gnodes by a resource. The payload used by this function is made up of two fields: minimum speed required for downloading (2 bytes) and search criteria of variable length. A Gnode when receiving a Query message and stating that there are no coincidences in its file structure will not generate any message in answer to it.

Example. Query message

```
                      28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
                      80 07 00 06 00 00 00 00 00 2A 2E 61 00
```

Message	
Gnode identifier	: 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
Function	: 80
TTL	: 07
Hops	: 00
Payload length	: 06 00 00 00
Payload	
Minimum speed	: 00 00
Search criteria	: 2A 2E 61 00 (*.a + null character)

5. Query_Hit Function (code 0x81)

It is used like answer to a Query message. The fields that compose a Query_Hit are: amount of hits or coincidences (1 byte), port (2 bytes) and IP address for downloading (4 bytes), operation speed (4 bytes), results (variable length) and gnode identifier that answers (16 bytes). The results (that can be n) are structured like a list, of the following syntax: index or number of answer (4 bytes), file size (in Kbytes) (4 bytes), file name (variable length) and register delimiter (0x0000). A Gnode includes in the answer its identifier to be used if the client must send a Push message.

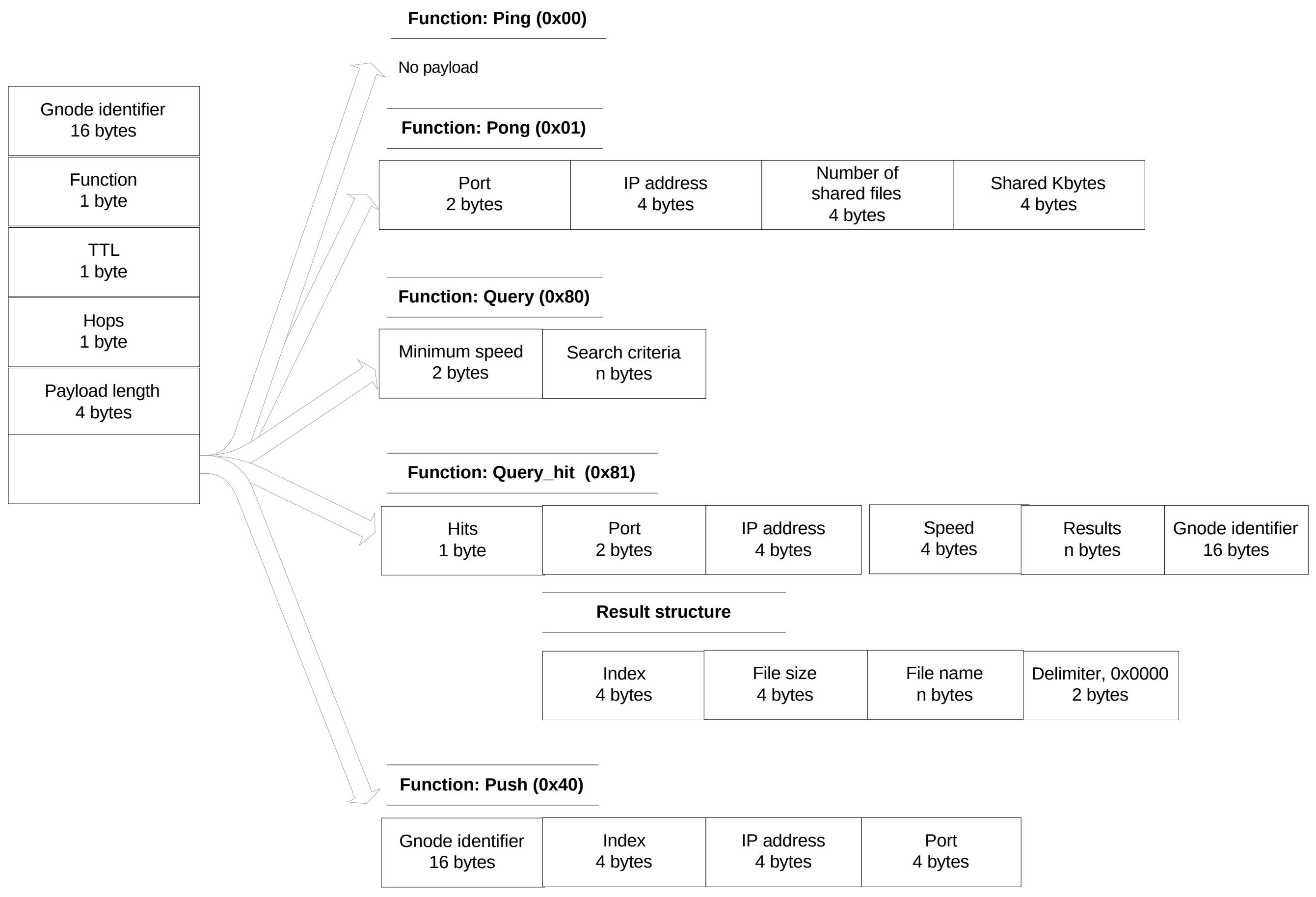
Example. Query_Hit Message

```
                      28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
                      81 07 00 57 00 00 00 03 CA 18 AA D2 62 95 1C 00
                      00 00 00 00 00 00 70 00 00 00 31 34 39 61 72 63
                      68 69 2E 61 00 00 01 00 00 00 70 00 00 00 31 34
                      39 61 72 63 68 69 2E 62 00 00 02 00 00 00 70 00
                      00 00 31 34 39 61 72 63 68 69 2E 6D 00 00 A0 3D
                      8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66
```

Message	
Gnode Identifier	: 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62
Function	: 81
TTL	: 07
Hops	: 00
Payload length	: 57 00 00 00
Payload	
Hits	: 03
Port	: CA 18

```
        IP address          : AA D2 62 95
        Speed               : 1C 00 00 00

Result structure
        Gnode  identifier   : 3D 8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66
Result structure
        1st element
                Index               : 00 00 00 00
                File size           : 70 00 00 00
                File name           : 31 34 39 61 72 63 68 69 2E 61 (149archi.a)
                Delimiter           : 00 00
        2nd element
                Index               : 01 00 00 00
                File size           : 70 00 00 00
                File name           : 31 34 39 61 72 63 68 69 2E 62 (149archi.b)
                Delimiter           : 00 00
        3st element
                Index               : 02 00 00 00
                File size           : 70 00 00 00
                File Name           : 31 34 39 61 72 63 68 69 2E 6D (149archi.m)
                Delimiter           : 00 00
```

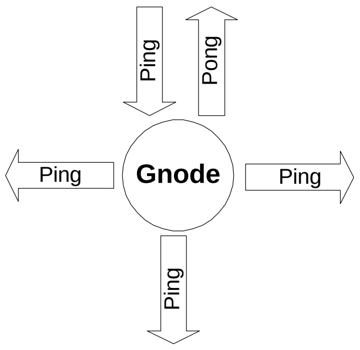


Gnutella protocol, messages data structures, version 0.4

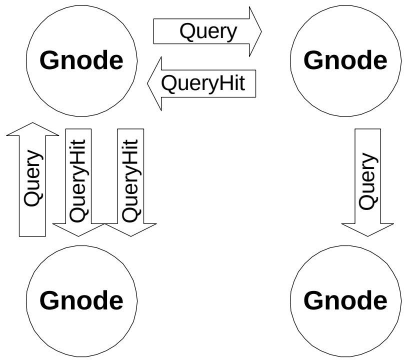
Propagation Rules

Peer-to-peer model used by Gnutella requires that gnodes have the ability to propagate received messages (Ping, Pong, Query, Query_Hit and Push) through their open connections. To operate in an efficient way, (considering network topology, the nonhierarchy and the existence of loops) gnodes must implement several propagation rules [DSS2], such as:

Rule a. A gnode will propagate Ping and Query messages to all gnodes which it's directly connected, except the one that gave this message.



Ping Propagation



Query Propagation

Rule b. Pong, Query_Hit and Push messages must be propagated by the same way by which the initial associated message (Ping or Query) traveled. This is possible because before propagating a message, gnodes review in internal tables by which generated such answer, and thus obtain the connection by where it must send this again.

Rule c. A gnode will decrement the value of field TTL in one unit and will increase the value of the Hops field in one unit before propagating this message by the pertinent connections. If when decrementing the value of TTL the obtained result is zero, it must reject the message.

Rule d. If a gnode receives a message with identical gnode identifier and same function code that one received previously (in a brief period of time, not formally specified) it would have to avoid to propagate it

At a moment, it can have thousand gnodes in the network, but for one gnode its dominion of action will be restricted to the amount of established connections and within reach of their messages (before the value of field TTL reaches zero). Due to the dynamism, the topology and the existence of temporary users, are possible that the answers to a same consultation be different in the time. Whenever a user enters in the network can do it to a part different from the same one, according to the present state of its neighboring gnodes.

File Downloading

Once gnode receives a result (Query_Hit) to a previous query, it can choose to select a file for its downloading from the remote gnode. Files recover directly, without intervention of intermediate gnodes and therefore either of the network Gnutella. The downloading is made using the HTTP version 1.0 protocol [BERNERS].

When a gnode receives a Query_Hit, it obtains the following data: IP address and port where gnode that has the resource is in passive open, index, file size and file name. To begin the download, the client gnode will open a connection TCP against remote gnode in the port previously obtained and - using the primitive GET of protocol HTTP - it will ask for the selected file, indicating index and file name.

Example

```
GET /<index>/<file name>/  HTTP/1.0
Connection: Keep-Alive
Range: bytes=0-
```

When the remote Gnode receives such request and validating that the asked for file is available, will transmit the file, sending first a standard HTTP header with the following format:

HTTP 200 OK
Server: Gnutella
Content-type: application/binary
Content-length: <file size in bytes>

It is important to emphasize that it's possible the inclusion of the "resume" functionality in the GET header, to ask for a file from a certain byte number to the end. This is useful in case of previous interrupted downloads.

Operation of a Gnutella Network

To describe a Gnutella network behavior, it was made a laboratory experience in order to obtain data for a protocol analysis. Four computers were used running Gnutella software version 0.56 (http://gnutella.wego.com), so that each one of these acted like a gnode. Next, those were described:

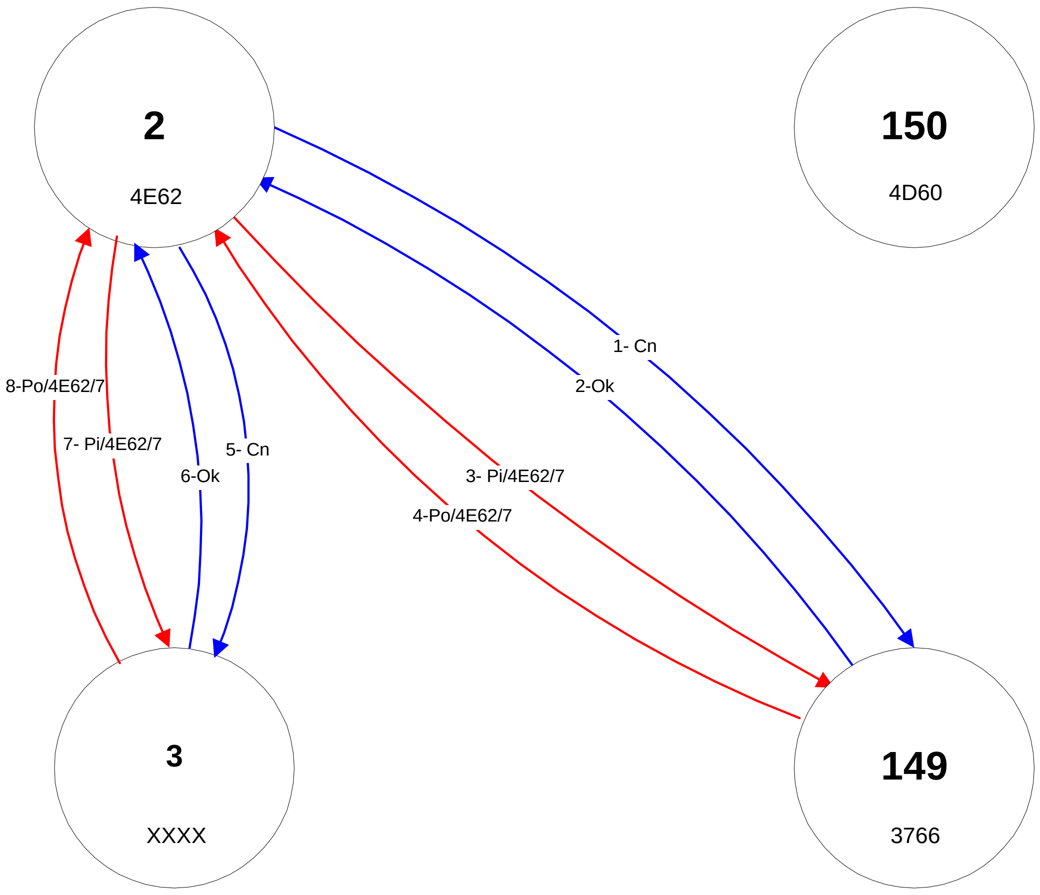
Gnode IP address:	170.210.98.2 (only last byte in graphs)
Gnode Identifier:	4E62 (four last bytes)
Shared Files:	2archi.a, 2archi.b y 2archi.z
Gnode IP address:	170.210.98.3
Gnode Identifier:	XXXX
Shared Files:	3archi.a, 3archi.b y 3archi.i
Gnode IP address:	170.210.98.150
Gnode Identifier:	4D60
Shared Files:	150archi.a, 150archi.x y 150archi.p
Gnode IP address:	170.210.98.149
Gnode Identifier:	3760
Shared Files:	149archi.a, 149archi.b y 149archi.m

In graphs, each arrow indicates the message direction, and its identification consists of four elements: message number, message type, source gnode identification and TTL. The heavy arrow indicates an opened connection and its direction (from the opening gnode). The message types are codified in the following way:

Cn:	GNUTELLA CONNECT	– Connection opening at application level
Ok:	GNUTELLA OK	– Connection opening accepted
Pi:	PING	– Ping
Po:	PONG	– Pong
Qy:	QUERY	– Query
Rs:	QUERY_HIT	– Response

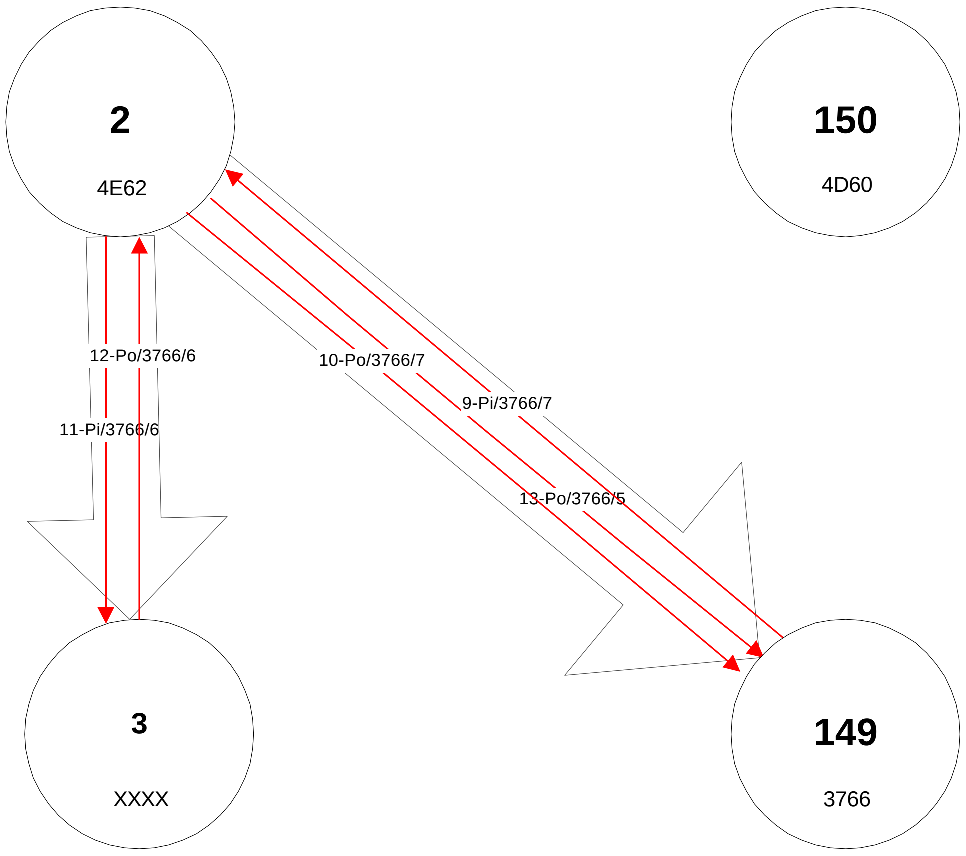
Operation sequence was the following:

First, Gnode 2 was asked to open a communications with gnode 149 and gnode 3. Once accepted the connection opening at the application level, gnode 2 sent a Ping message to each one of them, obtaining a Pong message in answer.



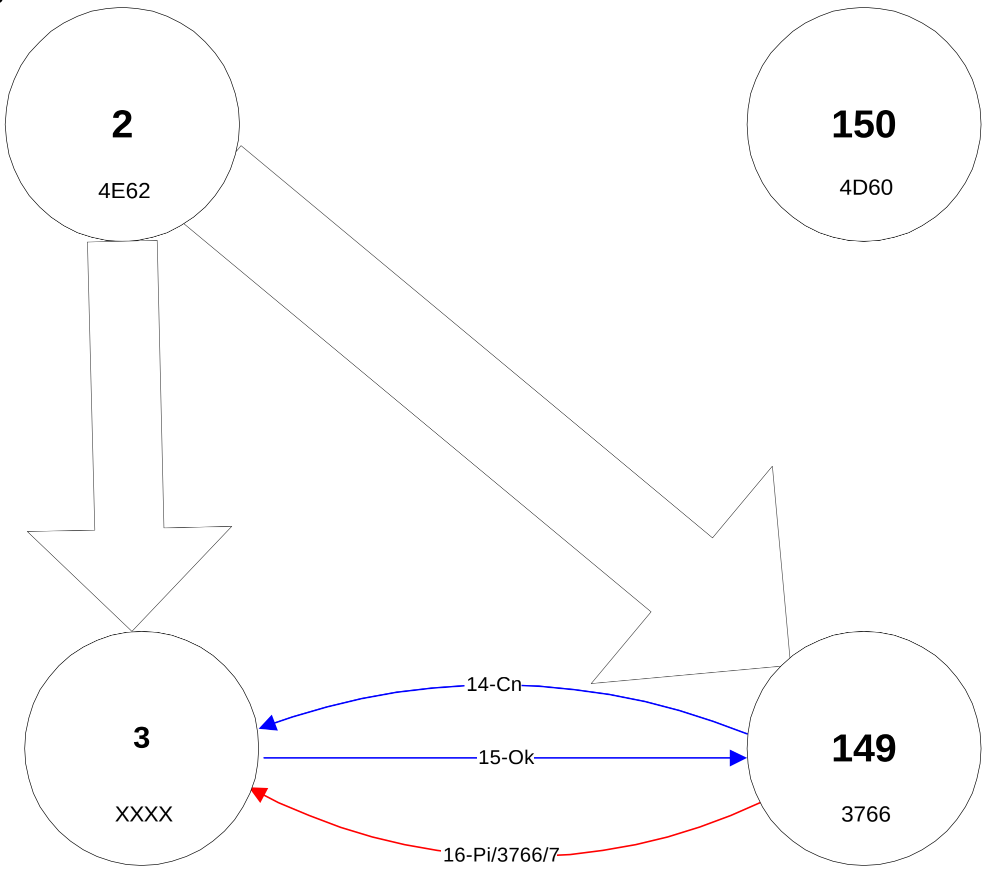
Graphic 1

Next, was asked to gnode 149 the execution of a “update” application command, which generated the shipment of a Ping message, by all its open connections. It sent the message to gnode 2, which first answered it (with its Pong message) and soon it propagated it by all his opened connections, except by where it received the Ping (in this case with gnode 3), decrementing TTL field. Gnode 3 answered to gnode 2 to the Ping and this one redirected it to gnode 149, that it was that originated the requirement.



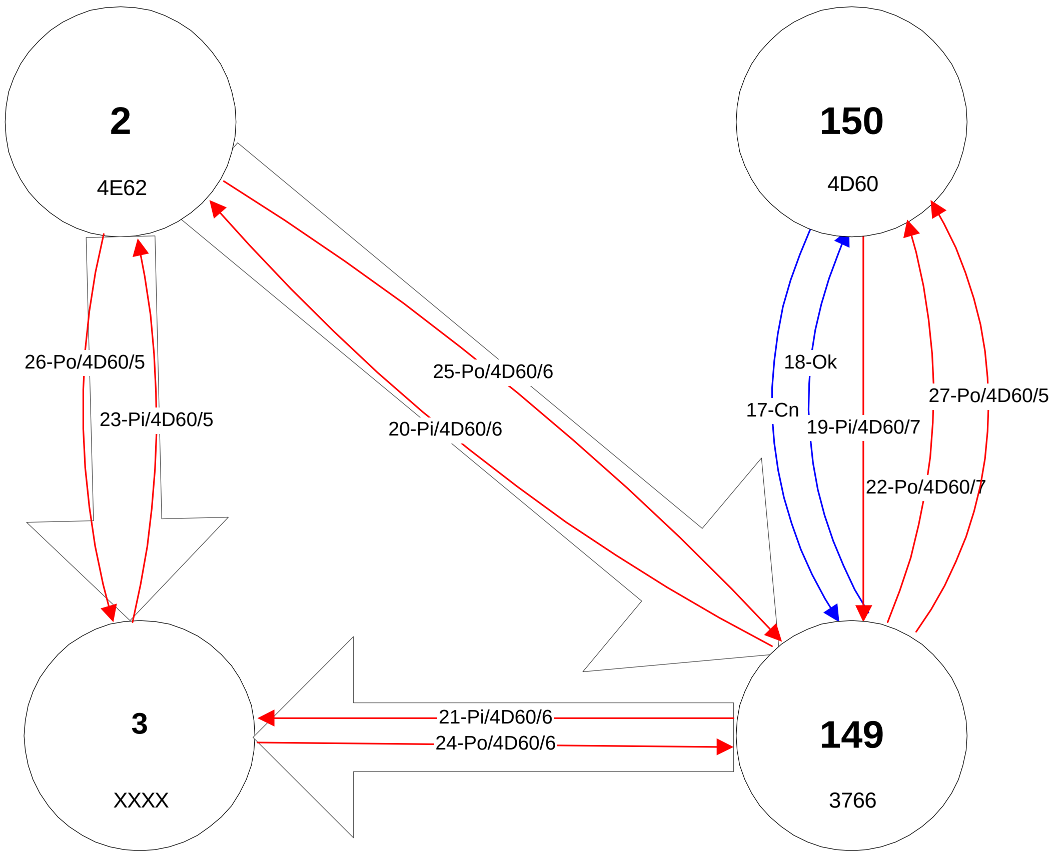
Graphic 2

As receiving information of the presence of Gnode 3, gnode 149 opened a connection with it, and sent a Ping message.



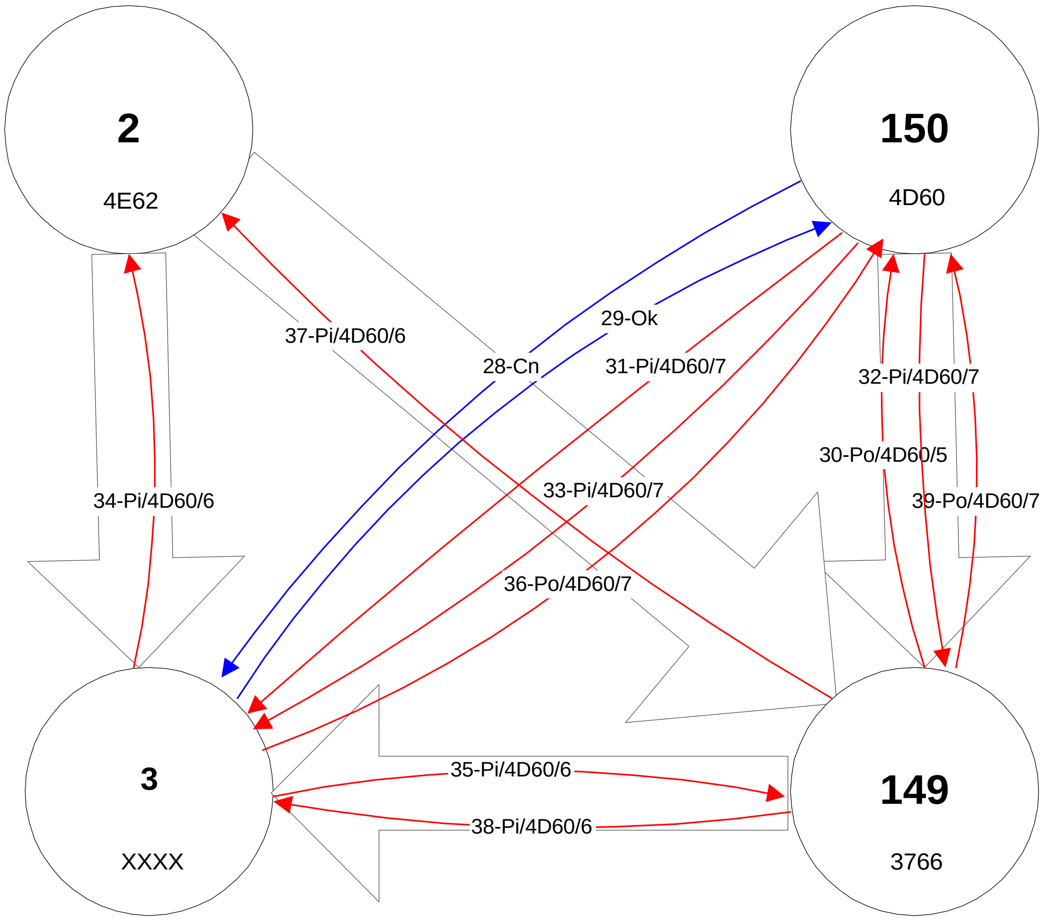
Graphic 3

Soon, the opening of a connection with gnode 149 was asked for to gnode 150. Then, it sent a Ping message to gnode 149, which was answered (with a Pong) and propagated by all its open connections (with gnodes 2 and 3, those did the same as well). The answers traveled by the same ways where the Ping message circulated and, finally, they arrived at gnode 150 (with the message TTL field decremented according to the gnodes by they were routed).

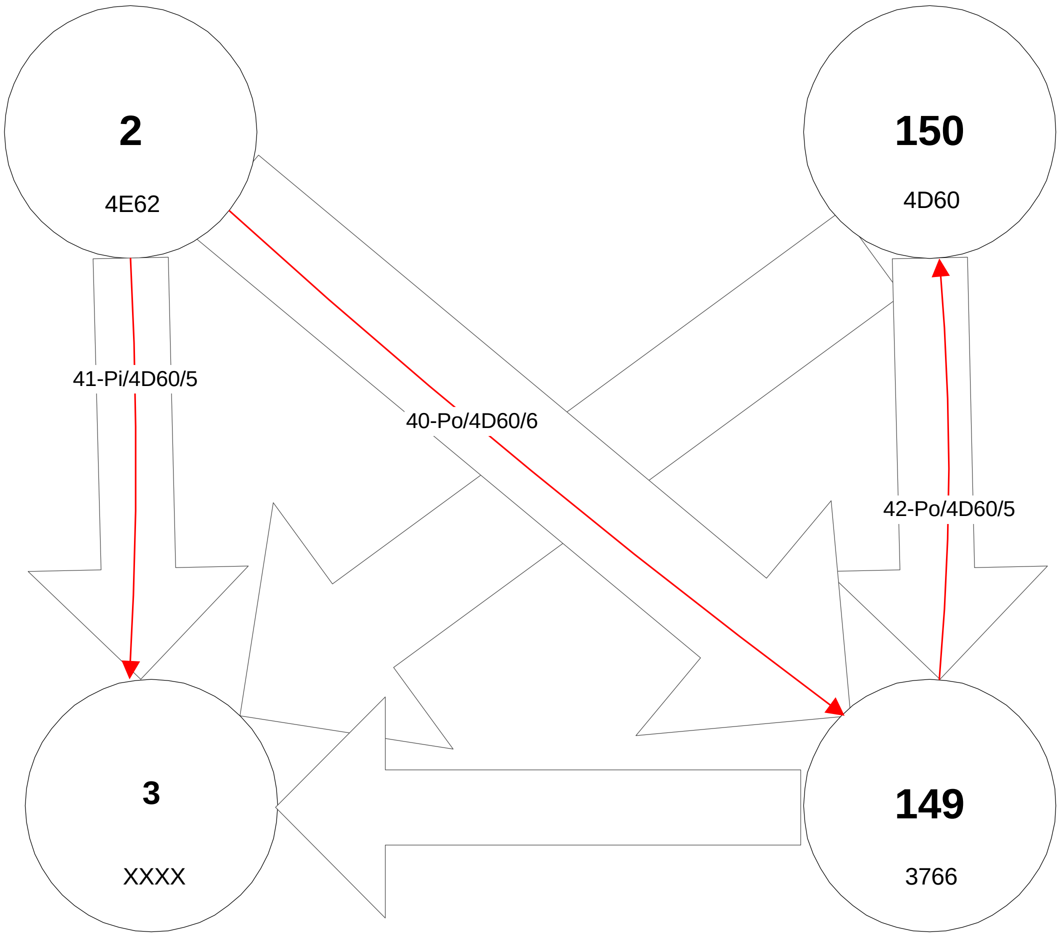


Graphic 4

When receiving information of the existence of gnode 3, gnode 150 opened a connection with this one and it sent a Ping message to it, which was propagated according to the same previous behavior.


2
4E62
150
4D60
3
XXXX
149
3766
37-Pi/4D60/6
29-Ok
28-Cn
31-Pi/4D60/7
32-Pi/4D60/7
30-Po/4D60/5
34-Pi/4D60/6
33-Pi/4D60/7
39-Po/4D60/7
36-Po/4D60/7
35-Pi/4D60/6
38-Pi/4D60/6

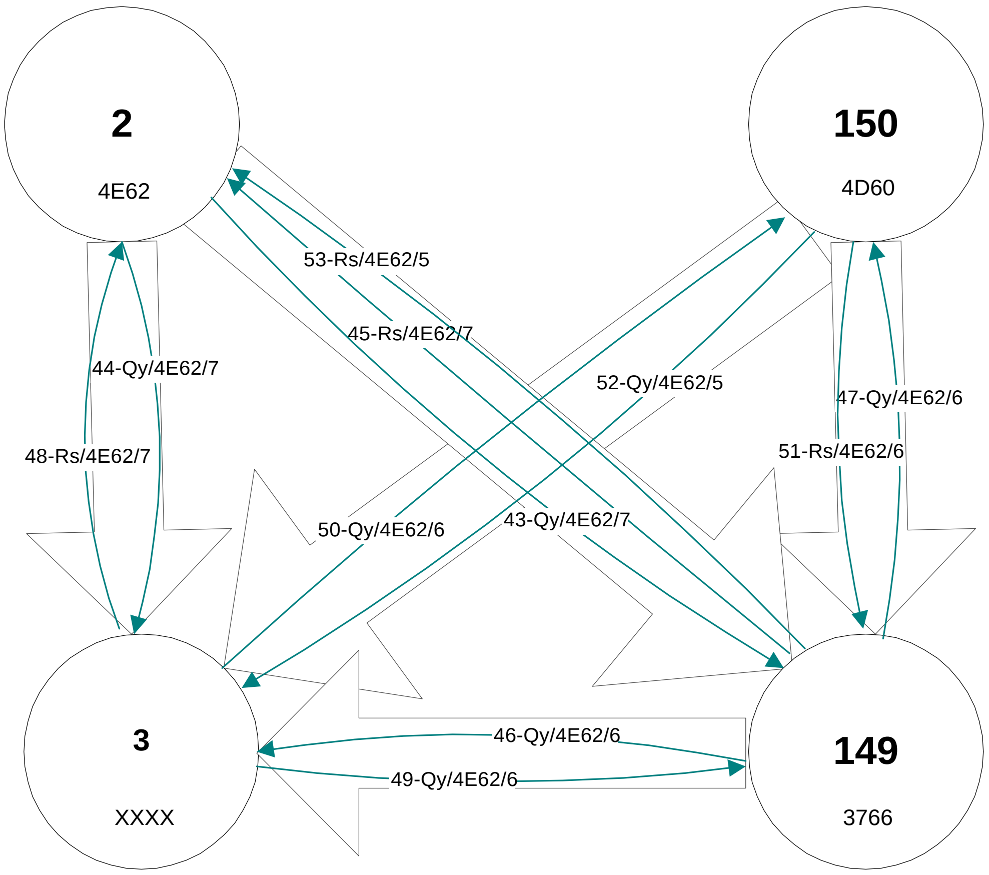
Graphic 5


2
4E62
150
4D60
41-Pi/4D60/5
40-Po/4D60/6
42-Po/4D60/5
3
XXXX
149
3766

Graphic 6

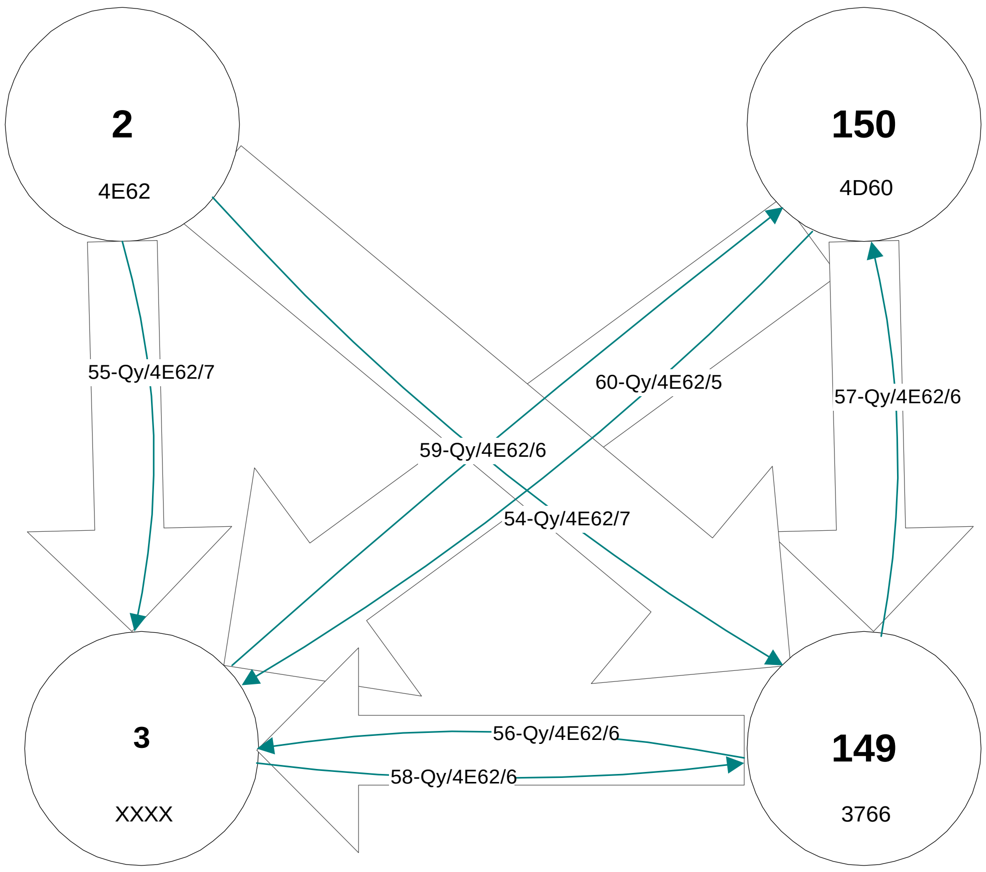
At this moment, the Gnutella network consisted of four gnodes, on which were executed the following queries:

In gnode 2 the query was made indicating the *.a pattern. This gnode sent the query by all its open connections (in this case with gnodes 149 and 3). Gnodes 149 and 3 - as well - propagated the query through all their connections (except by where it come). Those gnodes that have files which matched the search pattern (in this case gnodes 2, 149 and 150) answered the query sending a message by the same way of the query message. The behavior of the gnodes (propagating queries) was similar to Ping message propagation, happening the same with the answers (with Pong messages).



Graphic 7

Finally, in gnode 2, it was executed a query with “i.a” pattern. This gnode sent the query message of the same way that the previous one. Gnodes 149 and 3 propagated the query. No one gnodes had files whose name matched the pattern search, therefore they did not respond it.



Graphic 8

Conclusions

The massive use of the protocol on different platform implementations has demonstrated that Gnutella is valid beginning for distributed services development based on the peer-to-peer philosophy.

Due to his history and it short life, Gnutella is still an immature protocol, and it's been studied to evaluate efficiency and functionality improvements (to implement in later versions).

As an interesting feature, it defines a network at application level, oriented to be cooperative, and it offers heterogeneity, stability, redundancy and fault tolerant elements.

The propagation model can serve to generate distributed applications that are not only limited to file sharing tasks. For example, this model could be used, with smaller modifications, to obtain a network of distributed processing based on a cooperative model.

References

[ADAR]	Adar, E. y Huberman, B. Free Riding on Gnutella. Technical Report. Xerox PARC, August 2000.
[CLARKE]	Clarke, I.; Sandberg, O.; Wiley, B. y Hong, T. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, 2000.
[DSS1]	Gnutella: To the Bandwidth Barrier and Beyond. Distributed Search Services. http://dss.clip2.com. November 2000.
[DSS2]	The Gnutella Protocol Specification v0.4. Distributed Search Services. http://dss.clip2.com. September 2000.
[NAPSTER]	Napster, homepage http://www.napster.com

[ORAM]	Oram, A., "Gnutella and Freenet Represent True Technological Innovation," The O'Reilly Network, May 2000. http://www.oreillynet.com/lpt/a/208
[BERNERS]	Berners-Lee, T.; Fielding, R. y Frystyk, H. RFC 1945. Hypertext Transfer Protocol – HTTP/1.0. May 1996.
[SX]	Scour Exchange, homepage http://www.scour.com