Peer-to-Peer Networking with Firewalls

Lu Yan

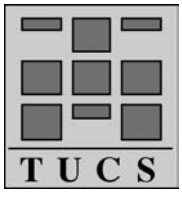
Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland.

Kaisa Sere

Department of Computer Science, Åbo Akademi University and
Turku Centre for Computer Science (TUCS),
FIN-20520 Turku, Finland.

Xinrong Zhou

Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland.



Turku Centre for Computer Science
TUCS Technical Report No 554
September 2003

ISBN 952-12-1218-7
ISSN 1239-1891

Abstract

A lot of networks today are behind firewalls. In peer-to-peer networking, firewall-protected peers may have to communicate with peers outside the firewall. This paper shows how to design peer-to-peer systems to work with different kinds of firewalls within the object-oriented action systems framework by combining formal and informal methods. We present our approach via a case study of extending a Gnutella-style peer-to-peer system to provide connectivity through firewalls.

Keywords: Firewall, peer-to-peer, object-oriented, action systems, Gnutella.

TUCS Laboratory
Distributed Systems Design

1 Introduction

The idea of peer-to-peer networking, in a sense that nodes on the network communicate directly with each other, is as old as internet itself. Internet used to be a peer-to-peer network if we go back to those early days in the 70's when internet was limited to researchers in a few selected laboratories. Nowadays internet has developed into a non peer-to-peer network, in the sense that most exchanges rely on mediation through gateways and servers. Moreover, most networks today employ firewalls, for security reasons, which impede direct communication by filtering packets and limiting the port numbers open to bi-directional traffic.

The goal of peer-to-peer networking is to remove the distinction between client and server. Instead of running web browers that can only request information from web server, users can run peer-to-peer applications to contribute contents or resources in addition to requesting them. As a vision of peer-to-peer networking, it is necessary for peer-to-peer applications to work in most environments, whether home, small business, or enterprise.

Previously [1] we have specified a Gnutella-like peer-to-peer system. When implementing such a system in Java, we realized that a lot of networks today are behind firewalls. In peer-to-peer networking, firewall-protected peers may have to communicate with peers outside the firewall. Thus a solution should be made to create communication schemes that overcome the obstacles placed by the firewalls to provide universal connectivity throughout the network. This motivates us to conduct a study of firewalls in peer-to-peer networking and achieve a way to traverse firewalls.

The rest of the paper is organized as follows. We start by defining our problem in Section 2. In Section 3 we give an overview of the Gnutella network. A solution to uni-directional firewalls is derived in Section 4. In Section 5 we provide a solution to another kind of firewalls which limit the open port numbers. We end in Section 6 with related work and concluding remarks.

2 Problem Definition

As firewalls have various topologies (single, double, nested, etc.) and various security policies (packet filtering, one-way-only, port limiting, etc.) [6], our problem has multiple faces and applications have multitude requirements. A general solution that fits all situations seems to be infeasible in this case. Thus we define the problem as shown in Fig. 1: How to provide connectivity between private peers and public peers through a single firewall?

We select the object-oriented action systems framework with UML diagrams as the foundation to work on. In this way, we can address our problem in a unified

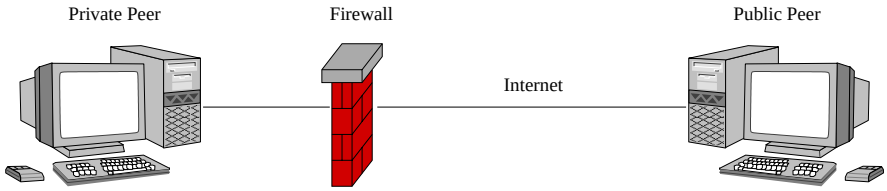


Figure 1: Problem definition

framework with benefits from both formal and informal methods.

Action systems is a state based formalism. It is derived from the guarded command language of Dijkstra [2] and defined using *weakest precondition* predicate transformers. An *action*, or a guarded command, is the basic building block in the formalism. An action system is an iterative composition of actions. The action systems framwork is used as a specification language and for the correct development of distributed systems.

OO-action system is an extension to the action systems framework with object-oriented support. An OO-action system consists of a finite set of classes, each class specifying the behavior of objects that are dynamically created and executed in parallel. The formal nature of OO-action systems makes it a good tool to build reliable and robust systems. Meanwhile, the object-oriented aspect of OO-action systems helps to build systems in an extendable way, which will generally ease and accelerate the design and implementation of new services or functionalities. Furthermore, the final set of classes in the OO-action system specification is easy to be implemented in popular OO-languages like Java, C++ or C#.

In this paper, however, we skip the details of semantics of action systems and its object-oriented extension, which can be found [3, 4].

3 Gnutella network

Gnutella [6] is a decentralized peer-to-peer file-sharing model that enables file sharing without using servers. To share files using the Gnutella model, a user starts with a networked computer A with a Gnutella *servent*, which works both as a server and a client. Computer A will connect to another Gnutella-networked computer B and then announce that it is *alive* to computer B. B will in turn announce to all its neighbours C, D, E, and F that A is alive. Those computers will recursively continue this pattern and announce to their neighbours that computer A is alive. Once computer A has announced that it is alive to the rest of the members of the peer-to-peer network, it can then search the contents of the shared

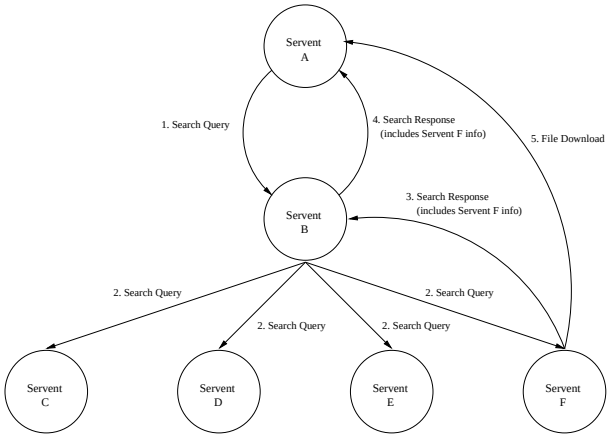


Figure 2: Gnutella peer-to-peer model [6]

directories of the peer-to-peer network.

Search requests are transmitted over the Gnutella network in a decentralized manner. One computer sends a search request to its neighbours, which in turn pass that request along to their neighbours, and so on. Figure 2 illustrates this model. The search request from computer A will be transmitted to all members of the peer-to-peer network, starting with computer B, then to C, D, E, F, which will in turn send the request to their neighbours, and so forth. If one of the computers in the peer-to-peer network, for example, computer F, has a match, it transmits the file information (name, location, etc.) back through all the computers in the pathway towards A (via computer B in this case). Computer A will then be able to open a direct connection with computer F and will be able to download that file directly from computer F.

4 Uni-directional firewalls

Most corporate networks today are configurated to allow outbound connections (from the firewall protected network to Internet), but deny inbound connections (from Internet to the firewall protected network) as illustrated in Fig. 3.

These corporate firewalls examine the packets of information sent at the transport level to determine whether a particular packet should be blocked. Each packet is either forwarded or blocked based on a set of rules defined by the firewall administrator. With packet-filering rules, firewalls can easily track the direction in which a TCP connection is initiated. The first packets of the TCP three-way handshake are uniquely identified by the flags they contain, and firewall rules can use this information to ensure that certain connections are initiate in only one di-

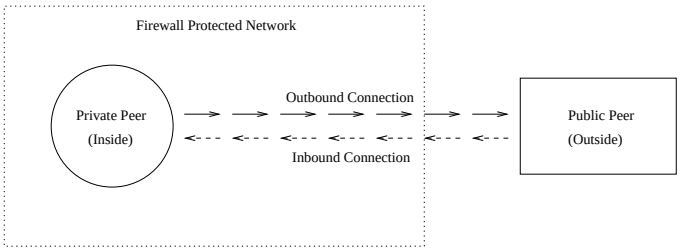


Figure 3: Uni-directional Firewall

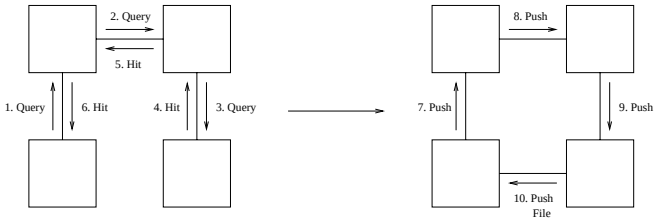


Figure 4: Push routing [5]

rection. A common configuration for these firewalls is to allow all connections initiated by computers inside the firewall, and restrict all connections for computers outside the firewall. For example, firewall rules might specify that users can browse from their computers to a web server on Internet, but an outside user on Internet cannot browse to the protected user's computer.

In order to traverse this kind of firewalls, we introduce a *Push* descriptor and routing rules for servents: Once a servent receives a QueryHit descriptor, it may initiate a direct download, but it is impossible to establish the direct connection if the servent is behind a firewall that does not permit incoming connections to its Gnutella port. If this direct connection cannot be established, the servent attempting the file download may request that the servent sharing the file *Push* the file instead. i.e. A servent may send a Push descriptor if it receives a QueryHit descriptor from a servent that doesn't support incoming connections.

Intuitively, Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptors as illustrated in Fig. 4. A servent that receives a Push descriptor with $ServentID = n$, but has not seen a QueryHit descriptor with $ServentID = n$ should remove the Push descriptor from the network. This ensures that only those servents that routed the QueryHit descriptors will see the Push descriptor.

We extend our original system specification [1] to adopt uni-directional firewalls by adding a Push router *Rf*, which is a new action system modeling Push

routing rules as shown in Table 1. We compose it with the previous two action systems [1] *Rc* modeling Ping - Pong routing rules and *Rl* modeling Query - QueryHit routing rules together, to derive a new specification of router

$$R \;=\; |[\; Rc \;/\!/\; Rl \;/\!/\; \mathit{Rf} \;]|$$

where on the higher level, we have components of the router

$$\{< Router, R >, < PingPongRouter, Rc >,$$
$$< QueryRouter, Rl >, < PushRouter, \mathit{Rf} >\}$$

A servent can request a file push by routing a Push request back to the servent that sent the QueryHit descriptor describing the target file. The servent that is the target of the Push request should, upon receipt of the Push descriptor, attempt to establish a new TCP/IP connection to the requesting servent. As specified in the refined file repository in Table 2, when the direct connection is established, the firewalled servent should immediately send a HTTP GIV request with *requestIP*, *filename* and *destinationIP* information, where *requestIP* and *destinationIP* are IP address information of the firewalled servent and the target servent for the Push request, and *filename* is the requested file information. In this way, the initial TCP/IP connection becomes an outbound one, which is allowed by uni-directional firewalls. Receiving the HTTP GIV request, the target servent should extract the *requestIP* and *filename* information and construct an HTTP GET request with the above information. After that, the file download process is identical to the normal file download process without firewalls. We summarize the sequence of a Push session in Fig. 5.

5 Port-blocking firewalls

In corporate networks, another kind of common firewalls are port-blocking firewalls, which usually do not grant long-time and trusted privileges to ports and protocols other than port 80 and HTTP/HTTPS. For example, port 21 (standard FTP access) and port 23 (standard Telnet access) are usually blocked and applications are denied network traffic through these ports. In this case, HTTP (port 80) has become the only entry mechanism to the corporate network. Using HTTP protocol, for a servent to communicate with another servent through port-blocking firewalls, the servent has to *pretend* that it is an HTTP server, serving WWW documents. In other words, it is going to mimick an *httpd* program.

When it is impossible to establish an IP connection through a firewall, two servents that need to talk directly to each other, solve this problem by having SOCKS support built into them, and having SOCKS proxy running on both sides. As illustrated in Fig. 6, it builds an HTTP-tunnel between the two servents.

Table 1: Specification of Push router

```
Rf = |[ attr   servantDB := φ; cKeyword := φ; filename := φ;
        target := φ; pushTarget := φ;
        obj    receivedMsg : Msg; newMsg : Msg; f : FileRepository
        meth   SendPush( ) = (newMsg := new(Msg(Push));
                   newMsg.info.requestIP := _this_IP_;
                   newMsg.info.filename := receivedMsg.info.filename;
                   newMsg.info.destinationIP := receivedMsg.info.IP;
                   _outgoing_message_ := newMsg);
               ReceiveMsg( ) = receivedMsg := _incoming_message_;
               ForwardMsg(m) = (m.TTL > 0 →
                   m.Transmit( ); _outgoing_message_ := m)
        do
            true →
                ReceiveMsg( );
                if receivedMsg.type = QueryHit →
                    servantDB := servantDB ∪ receivedMsg.servantID;
                    if receivedMsg.info.keyword = cKeyword →
                        target := receivedMsg.info.filename@
                            receivedMsg.info.IP;
                        if f.firewall →
                            SendPush( )
                        fi
                        cKeyword := φ
                    [] receivedMsg.info.keyword ≠ cKeyword∧
                        receivedMsg.descriptorID ∈ descriptorDB →
                            ForwardMsg(receivedMsg)
                    fi
                [] receivedMsg.type = Push →
                    if receivedMsg.info.destinationIP = _this_IP_ →
                        pushTarget := receivedMsg.info.requestIP®
                            receivedMsg.info.filename@
                            receivedMsg.info.destinationIP
                    [] receivedMsg.info.destinationIP ≠ _this_IP_∧
                        receivedMsg.servantID ∈ servantDB →
                            ForwardMsg(receivedMsg)
                    fi
                fi
        od
    ]|
```

Table 2: Specification of file repository

$$
\begin{array}{lll}
F = |[& \textbf{attr} & \mathit{firewall}^* := \mathit{false}; \mathit{fileDB} := \_\mathit{fileDB}\_; \mathit{cFileDB}; \\
 & & \mathit{filename} := \phi; target := \phi; pushTarget := \phi \\
 & \textbf{meth} & SetTarget(t) = (target := t); \\
 & & PushTarget(t) = (pushTarget := t); \\
 & & Has(key) = (\{key\} \in dom(\mathit{fileDB})); \\
 & & Find(key) = (\mathit{filename} := \mathit{file} \wedge \{\mathit{file}\} \in ran(\{key\} \lhd \mathit{fileDB})) \\
 & \textbf{do} & \\
 & & target \neq \phi \rightarrow \\
 & & \quad \textbf{cFileDB} := \textbf{fileDB}; \\
 & & \quad HTTP\_GET(target); \\
 & & \quad target := \phi; \\
 & & \quad \mathit{Refresh}(\mathit{fileDB}); \\
 & & \quad \textbf{if fileDB} = \textbf{cFileDB} \rightarrow \\
 & & \quad\quad \textbf{firewall} := \textbf{true} \\
 & & \quad [] \ \textbf{fileDB} \neq \textbf{cFileDB} \rightarrow \\
 & & \quad\quad \textbf{firewall} := \textbf{false} \\
 & & \quad \textbf{fi} \\
 & & [] \ pushTarget \neq \phi \rightarrow \\
 & & \quad HTTP\_GIV(pushTarget); \\
 & & \quad pushTarget := \phi; \\
 & & \quad \mathit{Refresh}(\mathit{fileDB}) \\
 & \textbf{od} & \\
]| & &
\end{array}
$$

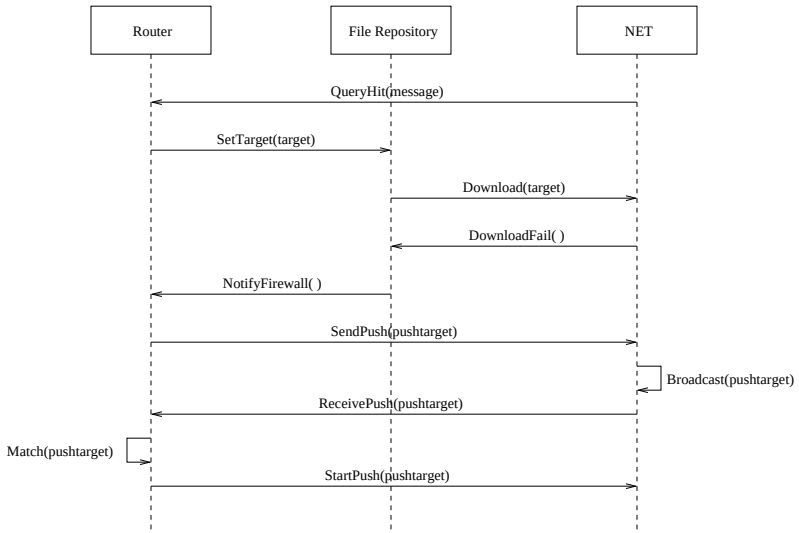


Figure 5: Sequence diagram of a Push session

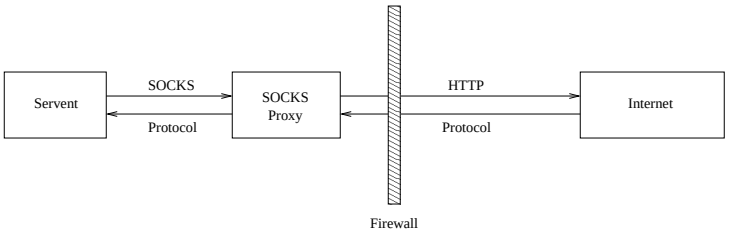


Figure 6: Firewall architecture and extendable socket

After initializtion, the SOCKS proxy creates a *ProxySocket* and starts accepting connections on the Gnutella port. All the information to be sent by the attempting servent is formatted as a URL message (using the GET method of HTTP) and a *URLConnection* via HTTP protocol (port 80) is made. On the other side, the target servent accepts the request and a connection is established with the attempting servent (actually with the SOCKS proxy in the target servent). The SOCKS proxy in the target servent can read the information sent by the attempting servent and write back to it. In this way, transactions between two servents are enabled.

We extend our original system specification [1] to adopt port-blocking firewalls by adding a new layer to the architecture of servent in Fig. 7. This layer will act as a tunnel between servent and internet.

As specified in Table 3, after receiving messages from the attemping servent and encoding them into HTTP format, the SOCKS proxy sends the messages to internet via port 80. In the reverse way, the SOCKS proxy keeps receiving mes-

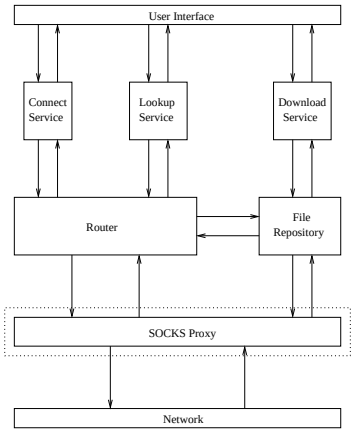


Figure 7: Refined architecture of servent

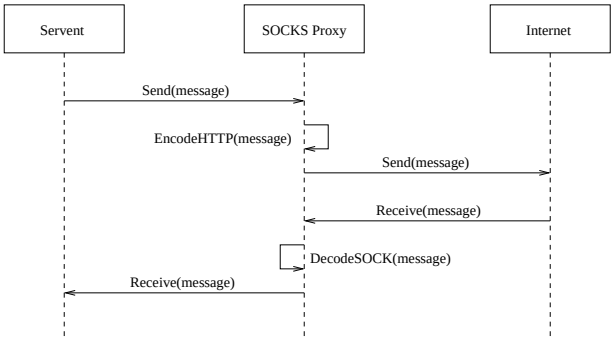


Figure 8: Sequence diagram of a SOCKS proxy session

sages from HTTP port and decoding them into original format. With this additional layer, our system can traverse port-blocking firewalls without any changes in its core parts. We summarize the sequence of a SOCKS proxy session in Fig. 8.

6 Related work and conclusions

The corporate firewall is a double-edged sword. It helps prevent unauthorized access to the corporate Web, but may disable access for legitimate peer-to-peer applications. There has been protocols such as PPTP (Point-to-Point Tunneling Protocol) [7], UPNP (Universal Plug and Play) [8], RSIP (Realm Specific IP) [9] and Middlebox protocol [10] to address the firewall problems in peer-to-peer networking. A recent protocol, JXTA [11] from Sun has provided an alternative

Table 3: Specification of SOCKS proxy

$$
\begin{array}{lll}
S = [\!| & \textbf{attr} & listenPort := \_Gnutella\_port\_; \\
 & & destinationPort := 80 \\
 & \textbf{obj} & ProxySocket : Socket; \\
 & & HTTPSocket : Socket; \\
 & & imsg : Msg; omsg : Msg \\
 & \textbf{init} & ProxySocket = new(Socket(listenPort)); \\
 & & HTTPSocket = new(Socket(destinationPort)) \\
 & \textbf{do} & \\
 & & \_incoming\_request\_ \neq \phi \rightarrow \\
 & & \quad imsg := EncodeSOCK(DecodeHTTP(HTTPSocket.Read())); \\
 & & \quad \_incoming\_message\_ := ProxySocket.Write(imsg) \\
 & & [] \ \_outgoing\_request\_ \neq \phi \rightarrow \\
 & & \quad omsg := EncodeHTTP(DecodeSOCK(ProxySocket.Read())); \\
 & & \quad \_outgoing\_message\_ := HTTPSocket.Write(omsg) \\
 & \textbf{od} & \\
|\!] & &
\end{array}
$$

solution to address the uni-directional firewalls problem by adding a publicly addressable node, called "rendezvous server", which firewalled peer can already talk to. The scheme is that peers interact mostly with their neighbors who are on the same side of the firewall as they are and one or a small number of designated peers can bridge between peers on the different sides of the firewall. But the problem posed by firewalls still remains when configuring the firewalls to allow traffic through these bridge peers.

We have specified a Gnutella-like peer-to-peer system within the OO-action systems framework by combining UML diagrams. In this paper, we have presented our solution to traverse firewalls for peer-to-peer systems. We have extended a Gnutella-style peer-to-peer system to adopt uni-directional firewalls and port-blocking firewalls using OO-action systems. The modular architecture of our system makes it easy to incorporate new services and functionalities without great changes to its original design.

Peer-to-peer computing is currently attracting lots of attention, spurred by the surprisingly rapid deployment of some peer-to-peer applications like Napster, Gnutella and Kazaa. Firewalls have become a great challenge to peer-to-peer computing of the Internet. A new technology, SOAP [12], has been developed to provide safe and reliable access through firewall protection. In the future work, we plan to explore this new standard and incorporate it into the development of

peer-to-peer systems in firewall environments.

Acknowledgements

We are grateful for Nayyar Iqbal's contribution to this project; and he wrote his Master thesis on peer-to-peer technology.

References

[1] L. Yan and K. Sere: *Stepwise Development of Peer-to-Peer Systems.* Proceedings of the 6th International Workshop in Formal Methods (IWFM'03), Dublin, Ireland, July 2003. Electronic Workshops in Computing (eWiC), British Computer Society (BCS).

[2] E.W. Dijkstra: *A Discipline of Programming.* Prentice-Hall International, 1976.

[3] R.J.R. Back and K. Sere: *From Action Systems to Modular Systems.* Software - Concepts and Tools. (1996) 17: 26–39.

[4] M. Bonsangue, J.N. Kok and K. Sere: *An approach to object-orientation in action systems.* Proceedings of Mathematics of Program Construction (MPC'98), Marstrand, Sweden, June 1998. Lecture Notes in Computer Science 1422. Springer Verlag.

[5] Clip2 DSS: *Gnutella Protocol Specification v0.4.*
Online. http://www.clip2.com/GnutellaProtocol04.pdf.

[6] I. Ivkovic: *Improving Gnutella Protocol: Protocol Analysis and Research Proposals.* Technical report, LimeWire LLC, 2001.

[7] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little and G. Zorn: *Point-to-Point Tunneling Protocol (PPTP).* RFC 2637, July 1999.

[8] *Understanding Universal Plug and Play.* Microsoft Corporation, WhitePaper, 2000.

[9] M. Borella and G. Montenegro: *RSIP: Address Sharing with End-to-End Security.* Proceedings of the Special Workshop on Intelligence at the Network Edge, California, USA, March 2000.

[10] B. Reynolds and D. Ghosal: *STEM: Secure Telephony Enabled Middlebox.* IEEE Communications Special Issue on Security in Telecommunication Network, October 2002.

[11] L. Gong: *JXTA: A network programming environment.* IEEE Internet Computing, 5(3): 88–95, May/June 2001.

[12] W3C: *Simple Object Access Protocol (SOAP).* Online. http://www.w3c.org.

Gnutella Client Development With Formal Specification

D1: Requirements (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. Introduction:

Gnutella is a decentralized system. It allows the users to share resources such as hard disk, CPU power etc. from their system and locate the resource shared by others on the network.

Every user of the Gnutella network has to download a Gnutella program, which will connect the user to the network and then user can reach and download any type of files.

Since there is a much discussion about formal specification like the benefits, code optimization and reliability that's why we are trying to implement Gnutella client with formal specification strategy to experiment the usage of formal specification.

The aim of this development is to implement a text based Gnutella client with formal specification strategy so that we can learn about the usage of formal specification in real time peer-to-peer systems.

2. Functionality:

There are many Gnutella clients in the market but there are very few clients that are highly robust and efficient. We are now trying to make a text based Gnutella client that follows formal specification style approach and that we think should be robust and reliable.

Although our client will be text based but it will give all the Gnutella functionalities like connection, forward messages, query, query reply and download.

If we will have enough time we will also try to implement the push functionality that is used if the client is behind firewall but still this push implementation is optional in our project.

We will make the client with open source so that everybody could come and implement his own ideas and make the client works better.

3. Future Extensions:

Our goal is to make a fully functional java based Gnutella client for every platform. We will try to implement all the basic and advance functionality.

We have many things to do in the future like adding security, lowering the network traffic and increasing the download speed. Also we will try to implement the media library so that the user can listen and check the file before downloading it fully.

Because the code will be based on GNU open license so everybody could be able to contribute and implement his own ideas in the future.

Gnutella Client Development With Formal Specification

D2: Project Management (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. Team Members:

The team for the project “Gnutella Client Development with Formal Specification” consists of

- Nayyar Iqbal
- Kaisa Sere
- Lu Yan

We have decided the responsibilities of each member of the group and also milestones.
Kaisa Sere is responsible for the project management; her task includes the management, assign the milestones and coordination of the project members. Lu Yan is responsible for providing the formal specification of the project and manages the documentation also providing the feed back of the development process. Nayyar Iqbal is responsible for the development and coding of the project.

2. Time Table

Milestone	Start Date	End Date	Notes
1	01-May-2003	20-May-2003	
2	15-May-2003	18-May-2003	
3	16-May-2003	02-July-2003	
4	15-June-2003	01-July-2003	
5	26-June-2003	05-July-2003	
6	05-July-2003		
7	20-July-2003		
8	04-August-2003		
9	15-Sep-2003		
10			

- **Milestone 1:**
  Reading
  Checkout and study carefully the documentation of Gnutella Protocol V0.4 and v0.6. In this part Nayyar Iqbal and Lu Yan coordinate each other about the protocol specifications. Because Lu has already done a research in Gnutella, he transferred his ideas to Nayyar Iqbal so that both can communicate in the same metal wavelength.

- **Milestone2:**
  Deciding the Gnutella programming language.
  Lu Yan and Nayyar Iqbal both agreed to write the client code in Java, because java is the most popular language now a days and it is available in all platforms.

- **Milestone 3:**
  Writing the first part of thesis.
  In this part Nayyar Iqbal has to write his first part of thesis i.e. to give the introduction of Formal Specification, Napster and then to write down the details of Gnutella Protocol.

- **Milestone4:**
  Start Implementation
  We have set the target to complete it in September 2003, because of deadline constrains, the implementation responsible Nayyar Iqbal has decided to utilize the Limewire classes.

  Creating a main menu.
  This part is the very basic in the development of Gnutella. In this part we agreed to decide the menu of the program. We will try to implement the menu in the main function which will be in the first class that has to be invoked when the java program start.

Gnutella Class:
Java program always start in the class, which has a main method and then goes on further. In Gnutella class we will implement the main method. We think our menu could be

1. Help
2. Connection
3. Search
4. Download

1. Help: is used to checkout the help documentation and print out the menu.
2. Connection: is used to checkout the connection. If the connection is linked with neighbors it will show some details about the neighbors. If the connection is not set up it will show the message to wait for the connection.
   e.g. connection
   Connection is not established; please wait for the connection to establish.
3. Search: After checking the connection the user can query the Gnutella network with search "string". This will return the results also with the details of files for downloading. The detail is IP address, port number, the name of file and file index.
   e.g. search songs
4. Downloading: Because our client is text based so this seems to be rather difficult process. The user should know about the details of file to download like IP address, Port number, file name and file index. The detail can be taken from the query part of our program.
   e.g. Download 135.168.2.7 6346 song 85
   After creating the menu we will implement the background details. This include to implement the classes such as Ping, Pong, Query, QueryHit and Router.

- **Milestone 5:**
  Creating a backend functionality
  Router Class:
  This will be the core class that will be called from the main Gnutella class. This class is responsible for calling the threads of the other classes for the backend functionality.

- **Milestone 6:**
  Creating a connection
  The connection class will be responsible for the connection and handshaking. This class will call some other classes like sockets and connection manager for developing packets and leaf or ultrapeer capabilities.

- **Milestone 7:**
  Creating a Ping class:
  This is the basic class for the ping message functionality. This will be called from the ping thread class that is responsible for the thread generation and this will constantly send a ping message to other servents for updates.

- **Milestone 8:**
  Creating a Pong class:
  This is the main class for the ping reply message. It will have a functionality to create a pong message and send it to the other servents.

- **Milestone 9:**
  Creating the Query class:
  This will be the basic class for the query message. This will make a query packet and send it to the network for the searches.

- **Milestone 10.**
  Creating a Query Hit Class:
  This class will be the response of the query from other hosts.

Gnutella Client Development With Formal Specification

D3: Design (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. System Overview:

The objective of this system is to create a Gnutella client by using the Formal Specification so that the client could be highly efficient and error free.

We choose Java programming language for our project because java provides reusable class hierarchy and it is available in every platform.

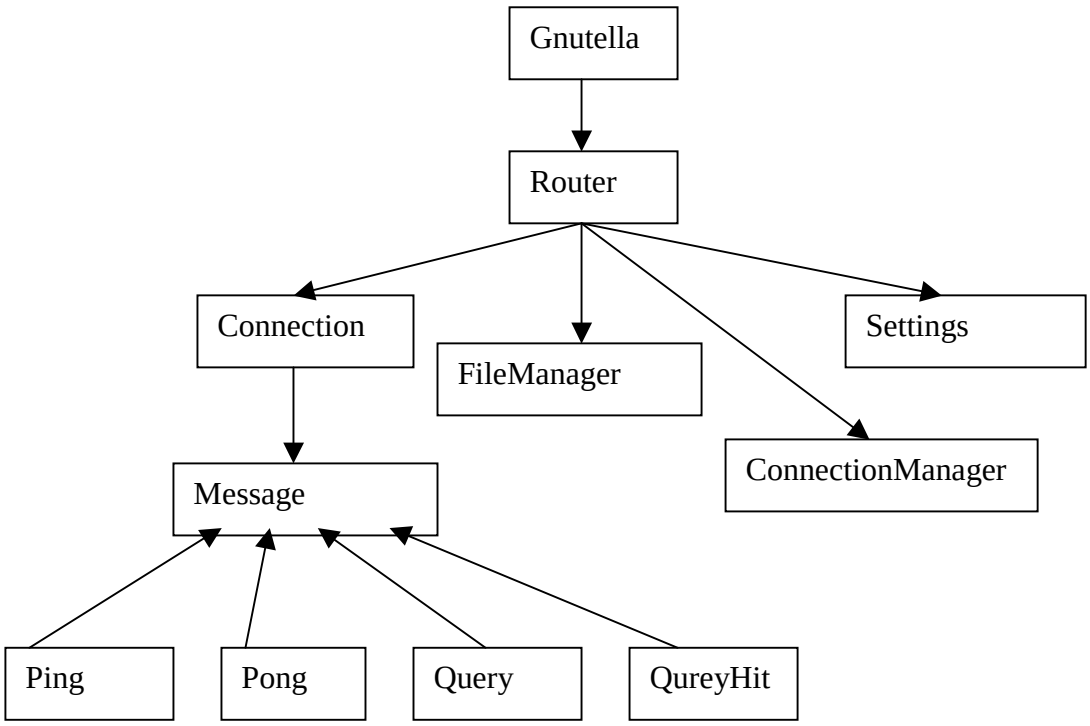
We will try to implement all the Gnutella client functionalities in our client like connection, ping, pong, query and query hit. We will also try to implement the push functionality if we will have enough time.

The main class of our client name is Gnutella.java, it has the functionalities of creating a backend thread for the class name router. Router class is the core of this client, it creates a connection, doing file management functionality and also some kind of backend setting.

Testing the functionality to be both working and in accordance with the java specification is done using a suite of tests. A complete set of tests can't however be guaranteed.

Further improvements would perhaps include a fast search engine and implements security feature.

2. Class Design:



Picture 1: Context Diagram of the Classes

3. Description:

The above picture shows the class hierarchy and calling of other classes. The main class of our program is Gnutella which also has a main function. So this is the first class that has to be instantiated first time when the user start the program. It then call the other classes for connection and management. This class is the basic class that has a menu functionality.

The menu is simple text based menu that has five commands. When this class starts it shows the welcome message, the project documentation website where the user can change the code under the GNU public license. After that it show the prompt of our client. The prompt is simple and DOS like prompt. In the prompt the user can type five different commands i.e., help, connection, search, download and exit. The connection command is used to check the status of the connection, if the connection is not established it shows the message that the user has to wait for the connection to established, if the connection is established it shows the status of the connection. We include only the basic things in the reply that will be print out and they are like “connection is established” and then the detail of the connection. The detail include the number of hosts that our client is connected to, the total number of files on the connected hosts, and the size of shared files in kilobytes.
The search command is used to search the string from the Gnutella network. It takes one argument of type string. The user can give the file name and also the file extension for differentiates between files. The search result include the IP address of the host, the port number where the other user is responding to, the name of the file and its index number.
Our client will show the result as soon as it gets the results from the hosts.

The download command is little complex because it takes four arguments that are necessary for the downloading of file. All these four arguments can be taken from the search results. The user has to type the download command, the IP address of the remote host, the port number through which the user is responding, the name of the file and the index number.
If the host is busy it will show the user like host is busy. If there are many users connected to the host and downloading the files and host cannot open more connection, it will then shows the message like too many uploads. If the host is not busy then it will respond with the HTTP OK message also shows the name of the client and then our client can download the particular file from the host. The user can see the downloading process because we print the data this is being transferring. The data is in binary form. The last menu option name is Exit and it is just for simply exit the application. It also shutdown all the backend threads and return to the DOS prompt.

The Gnutella class will call the Router class and start its thread. The Router class is the core of the client because it encapsulate all the basic functionality necessary for the backend events. It also create and destroy the connection and also provide some functionality of downloading file. The Router class instantiate the objects of other classes for the backend

functionality. It call the Connection Manager class, Host Catcher, Settings Manager, Listen Port and File Manager. It instantiate a threads for the back end functionality. So this class works with the connection, File Management and Settings classes for the initialization of connection and file Management.

The connection class is the basic class for messaging connection. The constructors in this class performs the socket functionality and reading and writing of messages. The connection class is the main class and Managed connection is the sub class of connection class. Managed connection has the responsibility for buffering. Also this class has a loop that reads the messages. This is also called message dispatch thread. The Managed Connection maintains a list of all ManagedConnection instances. The Managed Connection works with the Host Catcher class for cached addresses. It has also the responsibility of fetching the outgoing connections.

Message is the main class responsible for the creation of messages. The class stores the data that is common to the Gnutella message for Ping, Pong, Query and QueryHit. It is also the super class of these four classes. The class is responsible for reading and writing messages to input and output stream. This class also checks the packets from the network.

Ping class is the sub class of Message class. This class's main responsibility is to create a ping packet and send it to the output stream. This class defines some data structures that are necessary for the creation of ping packets.

Pong class is also a sub class of Message class, it has a main responsibility to create a reply to a ping message and send it to the corresponding host. It defines certain data structures that are necessary for the pong packet creation. This class defines many methods for the pong functionality. It also defines the creation of payload data packet and write it to the output stream.

Query class is also the sub class of Message class. This class is responsible for the generation of Query Message. First this class defines the data structures that are necessary for the query generation messages. This class defines four constructors for the generation of query message either from the scratch or from the network. This class works with the message class by calling the constructors of message class to generate the query packet. This class has one method that check for the illegal characters in the query. Also it has some accessor functions to get the payload and speed etc.

QueryHit is the reply of the query message that is generated by other host. This class contains information about the responding host in addition to an array of responses. These responses are not parsed until the getResponses method is called. This class is highly efficient and bad query reply packets may not be discovered until the getResponses method are called. These

methods may throw BadPacketException if the metadata cannot be extracted.

The FileManager class describes the list of all shared files. This class provides operations to add and remove individual files, directories and set of directories. This class also provides a method to efficiently query for files whose names contain certain keywords.

The ConnectionManager class lists all of Managed Connection by our client. This class provides methods for creating user-requested outgoing connections, accept incoming connections and fetches outgoing connection as needed. This class also creates a thread for handling these connections when appropriate. Because this is the only list of all connections, it plays an important role in message broadcasting. This class has methods for the downstream.

Gnutella Client Development With Formal Specification

D4: Implementation (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. Introduction:

Here we will describe the details of the main classes that are used in the our program. Also we will describe the main methods of the classes that are necessary to describe. The description of all the classes are beyond the scope this document. For the list of all the classes and methods the viewer can see the project website, which is www.abo.fi/~niqbal/gnutella. We also described some of the details of the classes in the previous document i.e. D3 Design.

2. Gnutella Class:

This is the main class with text menu functionality. This class is responsible for calling the backend functionality class called Router. We made five type of menus in this class. The help menu is responsible for generating help messages and also the list of menus. The status menu show the user about the status of the connection i.e. whether the user is connected to Gnutella network or not. If the user is connected to Gnutella then it will call the method of router class to show the status of connection. It also shows how many hosts the user is connected, the number of shared files and the total size of files.

The search menu is responsible for searching the files from the hosts. It takes one argument of type “string” and then called the router method name “search” to search the hosts for the particular file and show the results to the user. The result of query include the IP address and number of the port the user is connected to, the name of the file and the index number of the file. All these are necessary to download the file from the hosts.

The download menu is responsible for the downloading of a particular file. It takes four arguments as specified in the search results. The user has to give all the four arguments to download a file. The downloading process will be shown to the user by using a print function that will print the binary data that is being transferred. The exit menu simply shutdown all the backend threads, prints the bye message and quits the application.

Method Details:

Main Method:

This is the main method of Java which should be in the class that call the first time. It has the functionality of creating the text menu for the program. It uses some “if” checks for checking the user input command and then calling the particular functions.

String Method:

This method is used to parse the user input command

3. Router Class:

This class is the main backend class of the application. This class is responsible for making the threads related to connection and file handling. First this class will instantiate the objects related to connection and file management then it call the start method to start the backend thread. The start method call the other class methods to make threads, these includes the connection initialization thread that try to connect with the other hosts in Gnutella network, also this call the file management thread for the file management functionality like uploading and downloading. This class includes the three methods to show the results of the connection status as describe in Gnutella "connection" menu. This class has a one major function name shutdown that is used to shutdown all the threads and quits the application. This class instantiate the object of "HostCatcher" that is used for the update of the list of hosts by using the pongs. It also checks the number of super node availability and also whether the node is shielded or not. This class also has some methods related to get the IP and port number of the hosts, also the number of files shared and their size.

Method Details:

Start Method:
This method starts the various threads and tasks.

IsStarted Method:
This method is used to check for whether the backend threads have been started or not.

Shutdown Method:
This method shutdown the backend threads.

QueryDownload Method:
This method is the implementation of downloading.

Connect Method:
This method uses the object of connection manager to connect to the network.

getNumHosts Method:
This method return the number of hosts. This method is used in Gnutella class to print the number of users connection with our client.

getNulFiles Method:
This method return the number of files that are sheared by other users. This method is also used in Gnutella class to print the number of shared files.

getTotalFileSize Method:

This method return the size of total files that are shared. The value return is in kilobytes. This method is also used in Gnutella class to print the total size of all the shared files by other hosts.

query Method:
This method is used to search the network for files with the given data.

4. Connection Class:

This is the main connection class that is responsible for the connection to Gnutella network. It describe all the data structure necessary for connection like "GNUTELLA CONNECT/0.6" as describe in the Gnutella protocol specification. Other constants like TTL, CONNECT and 200 OK etc. This class also describe the availability of ultrapeer because ultrapeer functionality is the core in Gnutella version 0.6. This class also describes the socket functionality that is used to open the connection with remote host Also describe the compression scheme that is used to download the files.

This describe several methods for connection to Gnutella network. Also describe some kinds of exceptions like IO Exceptions and Null Pointer Exceptions.
This also describe two kind of constructors, one takes the argument of IP and host and is used for the un-initialized outgoing Gnutella 0.6 connection and the other one takes two arguments of type Socket and HandShakeResponder, this one is used for the un-initialized incoming Gnutella 0.6 connection, if the client is attempting to connect using 0.4 handshake it will reject it. This class is also responsible for the receiving of IO packets and handshake of the connection, also describe two types of initialization, one is initialize without timeout and the other one is initialize the connection by using handshake. This class includes some functions that respond to the other host by using handshake response. This class strictly define only Gnutella version 0.6 and that's why the client can only communicate with the other client that is using the same Gnutella version 0.6, if the other client is using version 0.4 then our client just drop the connection and try to connect to some other clients that are using Gnutella 0.6 version, also describe the functions for the read and write to the output stream. This class is responsible for the getting of IP address, port number of the host and also checks for the stability of the connection. Finally this class close the connection with remote host.

Methods Detail:

Initialize Method:
This method is responsible for the handshaking with the network.

Send Method:
This method is responsible for sending a message to the network.

Receive Method:
This method is responsible for receiving a message from the Gnutella network.

GetPort Method:
This method is responsible to return the port that the host is connected to. Normally the Gnutella port is 6346.

ReadLine and writeLine Method:
These methods are responsible for sending and receiving a handshaking line from the network.

Close Method:
This method close the socket connection.

postInit Method:
This method is called when the connection has been initialized and accepted.

concludeIncomingHandshake Method:
This is the main method for handshake. This responds to the handshake from the host on the other side. This involves multiple steps and "for" loop is used for the maximum handshake attempts. If the handshake will not be reached then it will throw the exception.

concludeOutgoingHandshke Method:
This method works like the concludeIncomingHandshake but this one respond to the hosts on the other side of the connection. This also involves multiple steps and for loop is used for the maximum handshake attempts. If the handshake will not reached then it will throw the exception.

initializeIncoming Method:
This method is also use for the handshake. This sends and receive the handshake string for the incoming connections. This method throws the exception if the handshake is not reached.

initializeOutgoing Method:
This method works same like initializeIncoming but this is used for the outgoing connection. This also throw the exception if the handshake is not reached.

notLessThan06 Method:
This method is used to check for the Gnutella client version number. If the version is 0.6 then it will try to connect, else the calling function of it will throws the exception.

5. Ping Class:

This class is the sub class of message. This is a main class for the Ping message. This class has the functionality of making the ping packet with the help of message class and send it to the network with the help of connection class.
This class has four different constructors for different functionality. The constructors will create a ping message for the Gnutella network and then call the main Message class to handle it.
The constructors can be of normal ping constructor that is used for the creation of ping from data read on the network, also to create a big ping request from data read from the network. The two other constructors are for the outgoing pings. One is for the creation of normal ping with new GUID and the last one is used to create a QueryKey request ping with a new GUID and TTL of 1.
This class also describe the functionality to write the payload to the output stream, the payload is the data structure of type byte array that is used to store the payload of the packet.
The message class is called from the constructors to generate a ping message and send to the network.

Method Details:

writePayload Method:
This method is used to write the payload to the output stream.

isQueryKeyRequest Method:
This method first check for the TTL, hops and payload, if the values are wrong then it will return the false, if everything is correct then it will return the parsed block.

6. Pong Class:

This is the main class for the ping reply functionality. This is also the sub class of Message class. This will create a pong packets with IP address, port number and the total number files shared.
This first describe the data structure that are necessary for the pong packet, these include the payload size, IP address, port number and number of files shared etc. This class has methods for the creation of Pong packet for specific host with the specific GUID and TTL. The two other methods creates a Pong for the host with the specific GUID, TTL and QueryKey. This class defines only one constructor that is used to establish all ping reply invariants. This class also describe the functionality of the ultrapeer, provides the accessors to return the IP address and port number, return the files that are sharing and the kilobytes of the shared files, all these are necessary for the pong packet creation.

Method Details:

getIP Method:
This method return the IP address of the user in a dotted decimal notation format.

getPortMethod:
This method return the port number of the user. Normally the port in Gnutella network is 6346.

getFile Method:
This method return the shared files to the host.

getKbytes Method:
This method return the size of files in Kilobytes

writePayload Method:
This method is used to write the payload to the output stream.

isUltrapeer Method:
This method checks for the message is marked with ultrapeer or not.

mark Method:
This method marks the given Kbytes fields.

stripExtendedPayload Method:
This return a message identical to its own but without any extended data.

7. Query Class:

This is the main class for creating query messages. This class also inherited from message class.
This class defines several methods related to query. First this class defines the data structure that is necessary for the query packet creation, this includes the payload data, query string, query key, minimum speed etc. This class describe four different types of constructors for the creation of query message. These includes creation of query from the scratch and it takes the necessary argument of GUID. This class also defines some other methods used to check about query string that whether or not it contain illegal characters. Some other methods used for the functionality to write the payload to the output stream, also some accessors functions for returning the payload and query etc.

Method Details:

getQueryMethod:
This method return the query string of this message.

newQueryGUID:

This return a new GUID for the query requests.

IsFirewalledSource Method:
This check that the query source is firewalled or not.

getPayloadMethod:
This method return the payload of the query.

getReplyAddress Method:
This return the IP address of the connection.

getReplyPort Method:
This return the port number of the connection.

hasIllegalChars Method:
This method checks whether the query string contains illegal characters or not.

desiresOutOfBandReplies:
This method is used to check whether the query source can accept out of band replies or not.

8. QueryHit Class:

This is also the sub class of message. This is the response of the query. This describes several data type that is necessary for the generation of QueryHit. The data structure include the payload, size of payload, vendor messages, support for browse host, busy and some other masks for the flag. This class provides constructor for the generation of QueryHit message. The other methods includes the creation of IP address from payload, getting the IP address, set the GUID for the message and writing the payload to the output stream. Also includes the checking whether the remote host is busy or not, the measuring of the speed of the QueryHit, parsing the result for the hit. Accessors include the getting of client GUID. Also this class calculate the quality of service of the given host for checking whether or not the host is behind the firewall or is it busy

Method Details:

setAddress Method:
This method is used to make the IP address from the payload.

setGUID Method:
This method set the GUID for this message. This method is necessary when we want to cache query replies or want to change The GUID.

getResultCount:
This method returns the number of results in this query.

getResultAsList:
This method returns the list that will yield the results.

getSupportBrowseHose:
This check for the client that whether it supports the browse host feature.

parseResult2 Methods.
This method parses the individual results for the hit.

getClientGUID Method:
This method returns the 16 byte client ID of the responding host.

writePayload Method:
This method is used to write the payload data to the output stream.

getIPBytes Method:
This method is used to return the IP address for the query hit as an array of bytes.

getIsBusy Method:
This method is used to check for the remote host is busy of not. This throws BadPacketException if the flag could not be extracted, either because it is missing or corrupted.

getIsMeasuredSpeed Method:
This method checks the speed in the QueryHit. This throws the BadPacketException if the flag of speed could not be extracted.

9. Message Class:

This is the main class for ping, pong, query and queryhit classes. This class define the data structure that will be used by all the classes like the ping, pong , query, query reply, vendor messages etc. This class describes three constructors with different arguments for creating the message from the scratch or from the network., also define the methods for reading the data packet from the network and also writing the data packet back to the network, defines the accessor functions for getting the network, checking for the network packets is from TCP or UDP, getting the GUID, TTL, hops, total length etc. It also defines the setter functions like setting the TTL, setting the GUID and hops

Method Details

Read Method:
This method is used to read the Gnutella messages from the specified input stream. The returned message can be of any recognize message like query, QueryHit, pings or pongs etc.

Write Method:
This method write the encoding of the object to out but this does not flush out.

writeGemExtension Method:
This method write the given extension string to the given stream. This report whether next call should add delimiter.

getPriorities Method:
This method returns the user defined priority for flow control purposes.

setPriority Method:
This method set the priority for the flow control purposes this takes one argument name priority of integer type and set the value with this argument.

compareTo Method:
This method takes one argument of message and then compares it with its own message. This returns a negative value if this is lesser priority than the message or return positive value if the priority is higher or zero if same priority.

ReadNullTerminatedBytes Method:
This method is used to read null terminated byte from the stream.

SetTTL Method:
This method sets the TTL to the given value as provided in argument. If the TTL is less than zero then it throws illegalArgumentException.

setGUID Method:
This method sets the GUID of the message with the GUID argument. This is needed when we want to cache query replies or other messages and change the GUID as per request.

SetHops Method:
This method sets the hops with the given argument. If the hops is less than zero then it throws illegalArgumentException. This method is used when we want certain message to look as if they have traveled further.

getLength Method:
This method returns the length of the payload in bytes.

UpdateLength Method:
This method updates the length of the payload in bytes

getTotalLength Method:
This method returns the total length of the object in bytes.

hop Method:
This method increment the hop and decrement TTL, if the TTL is greater, then return the old value of TTL.

10. FileManager Class:

This class list all the shared files. This describes the data structure necessary for the functionality of files like index, shared directories, threads, size of files etc. This class describes the function name initialize for the loading of all the files, getting the size and number of files etc, provide the functions to add and remove files, directories or set of directories, also provides methods to efficiently query for files whose name contains certain key words and describe the updates to shared directories and shared files

Method Detail:

Initialize Method:
This method is responsible for loading all files asynchronously.

getSize Method:
This method returns the size of all files in bytes. The return type is integer so the size of the file can be up to 2GB but if the size is more than 2GB it will return the same.

getNumFiles Method:
This method returns the number of shared files.

getNulPendingFiles Method:
This method returns the number of pending files that are pending for sharing.

getSharedFilesImp Method:
This method scans through all the files and return true if the directory was shared.

loadThreadInterrupted Method:
This method check if the load thread has been interrupted and if its true then it stop loading files.

updateDirectories Method:
This method recursively adds this directory and all subdirectories to the shared directories and update the number.

updateSharedFile Method:
This method updates the shared files with the list of sharable files.

addFileIfShared Method:

This method adds the given file from argument if it exists in a shared directory and has a shared extension.

addFile Method:
This takes one argument of file and add it to its own object if the file has a proper extension and not too big, this return true if the method is successful.

hasExtensions Method:
This returns true if the file name has a shared extension.

IfFileSharable Method:
This return true if the file is sharable else false.

search Method:
This method search a file in the shared directory and return a set of indices of files matching the query string.

Gnutella Client Development With Formal Specification

D5: Test Report (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. Overview:

Testing has been used throughout the development process. We did not make any special test cases but we simply used the testing capability of our java editor, which is IntelliJ Idea. This editor helps us a lot in testing the program. We first use the test for the connection to Gnutella network. This includes some of the connection classes. The connection test is not just simply a connection to the Gnutella network but it also includes the testing of ping and pong. After that we move forward and test the query and queryhit. In all the testing phases we added some exceptions because we found out that in some places the exceptions were missing. In the end of writing the whole working code we again tested it and keenly see the working and bugs. We tried to eliminate the bugs as much as possible but still the users can check it out and sent the bug report to the website because testing and fixing will go on in the life time of a program. Even the most sophisticated program has always some bugs left.

2. Tested Functionality:

The goal of this development is to create a fully functional Gnutella client that has all the core features of general Gnutella clients that are available in the market. Testing of this application is very time consuming because of the connection to the Gnutella network sometimes drops without given any notice. We aimed at testing the different modules and the overall functionality.

The explicitly tested modules are.

- Gnutella: Test for the menu commands, implement both uppercase and lowercase commands.
- Router: Check for the threading and backend functionality.
- Connection: Connection to Gnutella network, exceptions handling.
- Ping: Test for the class is working correctly under the packet generation. Check for the payload
  data.
- Pong: Check for the payload size data, IP address and port number. Test for the ultrapeer functionality.
- Query: Check the generation of query packet. Test for the illegal characters.
- QueryHit: Test if the queryhit packet is correct and correctly send to the host.

3. Non-tested Functionality:

Details in the implementation has not been explicitly tested – only through more general tests. These include if all the exceptions required to be thrown when specific conditions are met, if all the rules are correct, and many other details. Most of them should function as expected and required, but not necessarily all, and they are very difficult to test other than by running test after test.

Gnutella Client Development With Formal Specification

D6: User's Manual (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

History:

This documentation has evolved as follows:

- Original version 1.0 by Nayyar Iqbal, Kaisa Sere and Lu Yan, July 2003

Contents:

1. Introduction:

The Gnutella client is an open source implementation of Gnutella protocol. It is based on Java programming language designed by Sun Microsystems. Since Java released it got interest from a wide audience with all type of applications. The basic feature of Java language is that it provides a common application interface regardless of underlying platform allowing a project to be moved to a new architecture with a minimal or even completely without change, as long as the Java Virtual Machine exists for that architecture.

Our Gnutella client is slightly different from other Gnutella clients that are available in market because out client is a text based. Our client has all the basic functionality like the other clients and has a complete implementation of Gnutella protocol. The Java version we use in the implementation is the latest version 1.4 but we suppose that the client could also work with Java version 1.2. We use java programming language because it provides all the features that are necessary for formal specification like robustness, maintainability and modularity.

2. Obtaining the Software:

The program and its associated documentation can be download from www.abo.fi/~niqbal/gnutella.

3. Installation:

Installation is quite easy. The user can just download the program and copy it to the directory. The file gnutella.bat is used to run the application. The classpath is necessary to set. The classpath of java programming language is set according to the subdirectory of java. The gnutella.bat file sets the classpath of our program library.

4. Uninstallation:

Since we do not use the windows registry and other things of operating system, so its easy to uninstall the software. The user can simply issue the command rm –rf <subdirectory name> in Unix operating system and deltree <subdirectory> command in Windows system.

5 Usage:

When the program installed on the machine its easy to run it. Change the current directory to the directory of the program. The user computer should be set to run the java programs. The user has to check that the correct java version is installed in the operating system and classpath is set to run the java application from any directory. The library that our program use is in the subdirectory name lib in the program directory. The classpath of our library is set in gnutella.bat file so user should not worry about the classpath

of our program. If everything is set, the user can then issue a command gnutella.bat in the command prompt and the program will then work. After that the user can see the prompt of program and can issue the command help for getting help about the system and further process.

Gnutella Client Development With Formal Specification

D7: License (V1.0)

Turku Centre for Computer Science
Åbo Akademi University
Department of Computer Science

Nayyar Iqbal
Kaisa Sere
Lu Yan

July, 2003

GNU General Public License

Version 2, June 1991

The whole text is available at http://www.gnu.org/copyleft/gpl.html.

Turku Centre for Computer Science
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

http://www.tucs.fi

University of Turku
- **Department of Information Technology**
- **Department of Mathematics**

Åbo Akademi University
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

Turku School of Economics and Business Administration
- **Institute of Information Systems Science**