Project Report on Peer to Peer File Sharing System

CEN 502 - Computer Systems II - Fall 2015

Submitted To:
Dr. Violet R. Syrotiuk

Submitted By:
Ashish Aggarwal (1209260529)
Akash Goyal (1209535896)
Mengyuan Zhang(1209583385)

Part One: Survey of the summarizing existing P2P architectures for file sharing

According to (Peter, 2002), all peer-to-peer topologies, no matter how different they may be, will have one common feature. All file transfers made between peers are always done directly through a data connection that is made between the peer sharing the file and the peer requesting for it. The control process prior to the file transfer, however, can be implemented in many other ways. As stated by (Nelson, 2001), P2P fie sharing networks can be classified into four basic categories which are the centralized, decentralized, hierarchal and ring systems. Although these topologies can exist on their own, it is usually the practice for distributed systems to have a more complex topology by combining several basic systems to create, what is known now as hybrid systems. We will give a brief introduction to the four basic systems and later delve deeper into the topic of hybrid systems.

1. Centralized Topology

The concept of a centralized topology is very much based on the traditional client/server model. Please refer to Figure 1 for illustration. A centralized server must exist which is used to manage the files and user databases of multiple peers that log onto it (Peter, 2002). The client contacts the server to inform it of its current IP address and names of all the files that it is willing to share. This is done every time the application is launched. The information collected from the peers will then be used by the server to create a centralized database dynamically, that maps file names to sets of IP addresses. All search queries will be sent to the server, who will perform a search through its locally maintained database. If there is a match, a direct link to the peer sharing the file is established and the transfer executed (Kurose, 2003). It should be noted here that under no circumstances, will the file ever be placed on the server. Examples of applications that make use of such a network would be seti@home, folding@home, Napster which will be discussed in greater detail in the following sections, and many more.

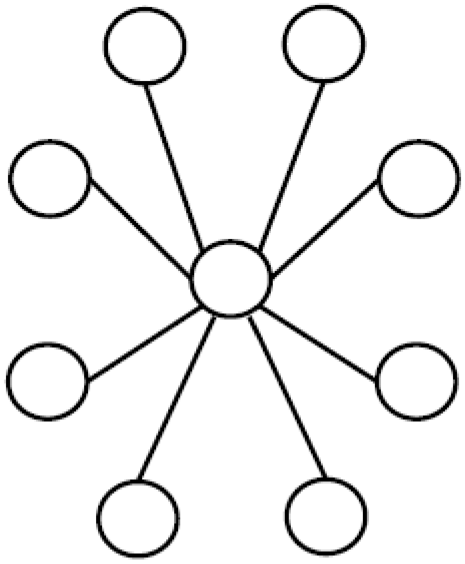


Figure 1: Illustration of the Centralized Topology

2. Ring Topology

It should be relatively clear that the drawback of a centralized topology is that the central server can become a bottle neck (when load becomes heavy) and a single point of failure. These are some of the main contributing factors as to why the ring topology came about. It is made up of a cluster of machines that are arranged in the form of a ring to act as a distributed server (Nelson, 2001). This cluster of machines will work together to provide better load balancing and high availability. This topology is generally used when all the machines are relatively nearby on the network, which means that it is most likely owned by a single organization; where anonymity is not an issue. Figure 2 shows a simple illustration of a ring topology.

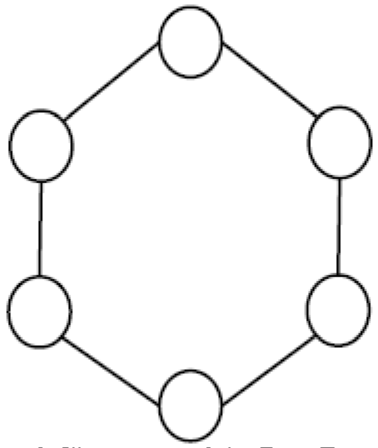


Figure 2: Illustration of the Ring Topology

3. Hierarchical Topology

Hierarchical systems have been in existence for a very long time. Even on the Internet, hierarchical systems have been about since the very beginning. Many Internet applications function in a hierarchical environment. The best example of a hierarchical system on the Internet would be the Domain Name Service (DNS) (Nelson, 2001). Authority flows from the root name servers to the servers of the registered name and so on so forth. This sort of topology is very suitable for systems that require a form of governance that involves delegation of rights or authority. Another good example of a system that makes use of the hierarchical topology would be the Certification Authorities (CA) that certify the validity of an entity on the Internet. The root CA can actually delegate some of its authoritative rights to companies that subscribe to it, so that those companies can, in turn grant certificates to those that reside underneath it. Figure 3 provides a brief illustration of how a hierarchical system looks like.

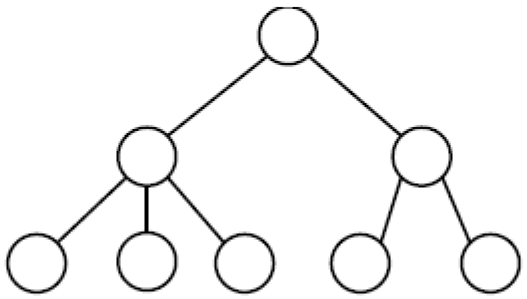


Figure 3: Illustration of the Hierarchy Topology

4. Decentralized Topology

In a pure peer-to-peer architecture, no centralized servers exist. All peers are equal, hence creating a flat, unstructured network topology (Peter, 2002). Please refer to Figure 4 for illustration. In order to join the network, a peer must first, contact a bootstrapping node (node that is always online), which gives the joining peer the IP address of one or more existing peers, officially making it part of the ever dynamic network. Each peer, however, will only have information about its neighbors, which are peers that have a direct edge to it in the network. Since there are no servers to manage searches, queries for files are flooded through the network (Kurose, 2003). The act of query flooding is not exactly the best solution as it entails a large overhead traffic in the network. A good example of an application that uses this model is Gnutella. Details of how it searches and shares files in a pure peer-to-peer network will be discussed later.

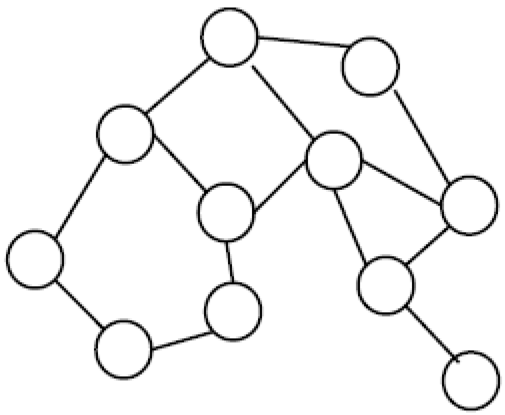


Figure 4: Illustration of the Decentralized Topology

5. Hybrid Topology

Having discussed the basic topologies of peer-to-peer networks, we now come to the more complex real world systems that generally combine several basic topologies into one system. This is known as the Hybrid architecture (Yang, 2002). We will discuss several of such examples in this section just to give a brief idea of this sort of architecture. In such a system, nodes will usually play more than one role.

5.1. Centralized topology and Ring topology

This hybrid topology is a very common sight in the world of web hosting (Nelson, 2001). As mentioned previously in the ring topology section, heavy loaded web servers usually have a ring of servers that specializes in load balancing and failover. So, the servers themselves maintain a ring topology. The clients however are connected to the ring of servers through a centralized topology (i.e. client/server system). Therefore, the entire system is actually a hybrid; mixture between the sturdiness of a ring topology with the simplicity of a centralized system. Figure 5 gives a simple illustration of such a topology.

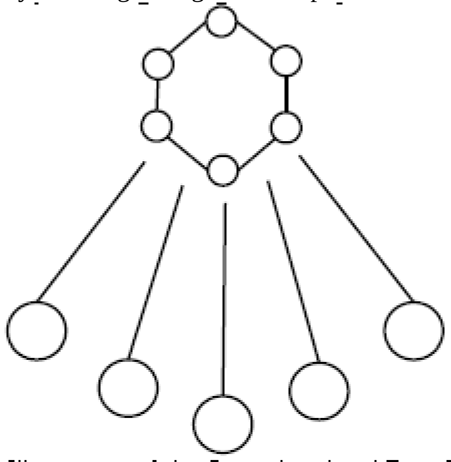


Figure 5: Illustration of the centralized and Ring topology

5.2. Centralized Topology and Centralized Topology

It is often the case where the server of a network is itself a client of a bigger network (Nelson, 2001). This sort of hybrid topology is a very common practice in organizations that provide web services. A simple example that will help illustrate this point would be, when a web browser contacts a centralized web server (Refer to Figure 6). The web server may process and format the results so that they can be presented in HTML format and in the process of doing that, these servers might themselves contact other servers (e.g. Database server) in order to obtain the necessary information (Nelson, 2002).

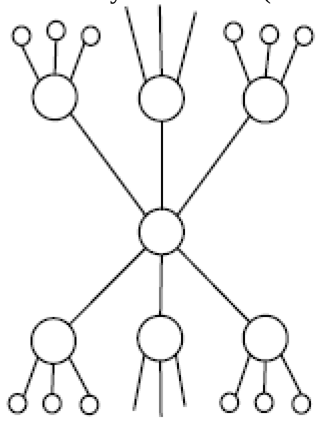


Figure 6： Illustration of the Centralized and Centralized Topology

5.3. Centralized Topology and Decentralized Topology

In this topology, peers that function as group leaders are introduced (Kurose, 2003). They have been known by many names. Some call them Group Leader Nodes, Super Nodes or even Ultra Nodes. To keep things simple and consistent with the following sections about Kazaa, we will refer to them as Super Nodes from here onwards.

These Super Nodes will perform the task of a centralized server as in the centralized topology, but only for a subset of peers. The Super Nodes themselves are tied together in a decentralized manner. Therefore, this hybrid topology actually introduces two different tiers of control. The first is where ordinary peers connect to the Super Nodes in a centralized topology fashion. The second is where the Super Nodes connect to each other in a decentralized topology fashion. Please refer to Figure 7.

As with the centralized topology, the Super Nodes maintains a database that maps file names to IP addresses of all peers that are assigned to it (Yang, 2002). It should be noted here that the Super Node's database only keeps track of the peers within its own group. This greatly reduces the scope of peers that it needs to serve. So, any ordinary peer with a high speed connection will qualify to be a Super Node. The best example of a peer-topeer application that utilizes such a topology would be Kazaa/FastTrack.

Another good example of such a topology would be the common Internet email. Mail clients have a decentralized relationship to specific mail servers. Like the Super Nodes, these mail servers share emails in a decentralized fashion among themselves.

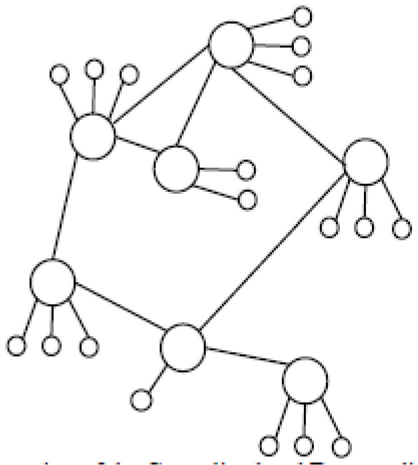


Figure 7: Illustration of the Centralized and Decentralized Topology

Part two: Design document

Abstract

The P2P system designed by our group consists of one centralized directory server and 3 peers in test. Each peer combines a functionality of a P2P client and a transient P2P server. As a P2P client, a peer makes requests to obtain information about available files from the directory server, and to obtain files from a peer serving files. As a transient P2P server, a peer serves files requested from a P2P client. In this P2P system, the directory server and each peer is uniquely identified by the name of its host computer and port number; each host has a unique IP address. Therefore, the main work of this design consists of two major parts:

1. The implementation of a communication protocol between the directory server and a P2P client.
2. The implementation of a communication protocol between a P2P client and a transient P2P server.

In the practical considerations, moreover, multiple tread control is designed to realize that every thread could operate independently and simultaneously.

Design Details

As what is introduced above, the main work of this project is divided into two part. The first part involves the communication between the directory server and a P2P client, in which UDP transport protocol and similar-to- HTTP message format are used.

The directory server stores a directory list which contains information such as filename, file size, file path, the IP address and host port of the host which stores the source file. When the directory server to P2P client directory build a connection, the client could implement three request commands for the directory server:

1. “Inform and update.”
2. “Query for content.”
3. “Exit.”

For each request command, different operations between directory server and P2P client will come. If the P2P client requests the directory server “inform and update”, the file name is required to be attached in the request message so that in the end of the directory server the directory list could update the new file with the file information such as the file name, file size, file path and IP address, host port of the peer who has stored this file. If “Query for content” is requested by the P2P client, the Filename is also required to be attached in the request message, and then directory server will return the File information as showed above to the client, which facilitates the client to build a connection to a destination host stored the file. Finally, if “Exit” is requested and when the request message arrives at the directory server, all the information of file stored in the request host will be deleted from the directory list of directory server.

The second part involves the communication between P2P client and a transient P2P server, in which TCP transport protocol is used. If a P2P client wants to download the file from another transient P2P server, on the P2P client side, firstly, it must get every information of the file to be downloaded from the directory server after “Query for content” request, then it calls for connection with the transient P2P server, request for the destination file download and receive the data stream of this file, on the transient P2P server side, firstly, its server process must be in progress and waiting for the calling from the P2P client to build the connection, then it sends the data stream to the P2P client. After the P2P client receive all the file content from transient P2P server, file download is finished and the connection will be closed.

The architecture of the system is to be similar to that of the original Napster; see Figure 3. However unlike Napster, which was used to share MP3 files. This file sharing system will be used to share text files.

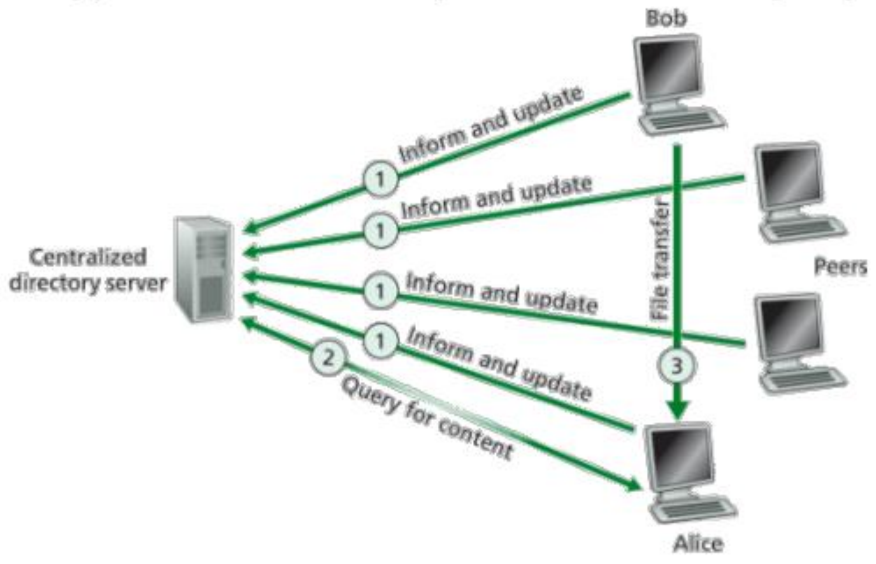


Figure 3. Architecture of P2P file sharing system in this project

1. P2P Client to Directory Server Communication

UDP Protocol

UDP, defined in [RFC 768], does just about as little as a transport protocol can do. Aside from the multiplexing/demultiplexing function and some light error checking, it adds nothing to IP. In fact, if the application developer chooses UDP instead of TCP, then the application is almost directly talking with IP. UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other small fields, and passes the resulting segment to the network layer. The network layer encapsulates the transport-layer segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process. Note that with UDP there is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be connectionless.

The UDP segment structure, shown in Figure 4, is defined in RFC 768. The application data occupies the data field of the UDP segment. The UDP header has only four fields, each consisting of two bytes. The port numbers allow the destination host to pass the application data to the correct process running on the destination end system (that is, to perform the demultiplexing function). The length field specifies the number of bytes in the UDP segment (header plus data). An explicit length value is needed since the size of the data field may differ from one UDP segment to the next. The checksum is used by the receiving host to check whether errors have been introduced into the segment. In truth, the checksum is also calculated over a few of the fields in the IP header in addition to the UDP segment. But we ignore this detail in order to see the forest through the trees. The length field specifies the length of the UDP segment, including the header, in bytes.

In the application layer, the UDP server and UDP client build the interaction by UDP sockets. Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet. After the packet passes through the sender's socket, the Internet will use this destination address to route the packet through the Internet to the socket in the receiving process. When the packet arrives at the receiving socket, the receiving process will retrieve the packet through the socket, and then inspect the packet's contents and take appropriate action. The detail process of UDP communication is showed in Figure 5.

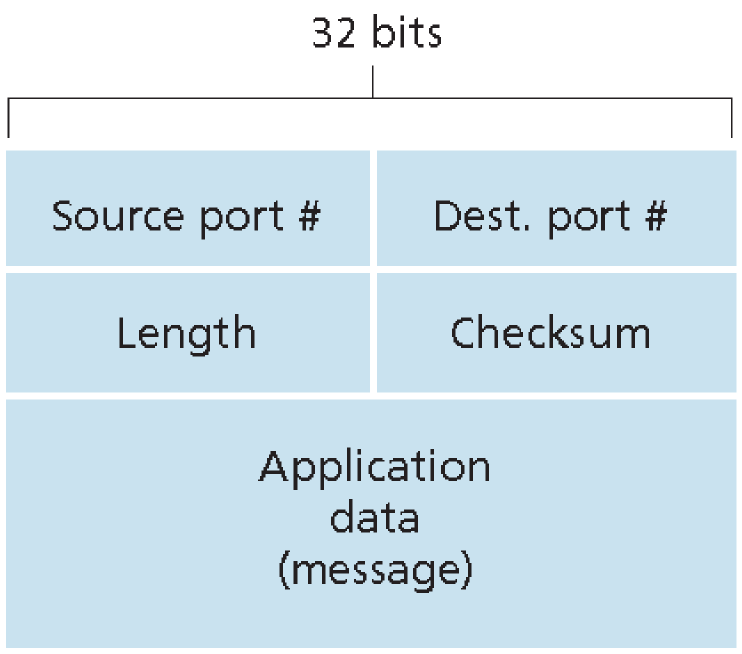


Figure 4. UDP segment structure

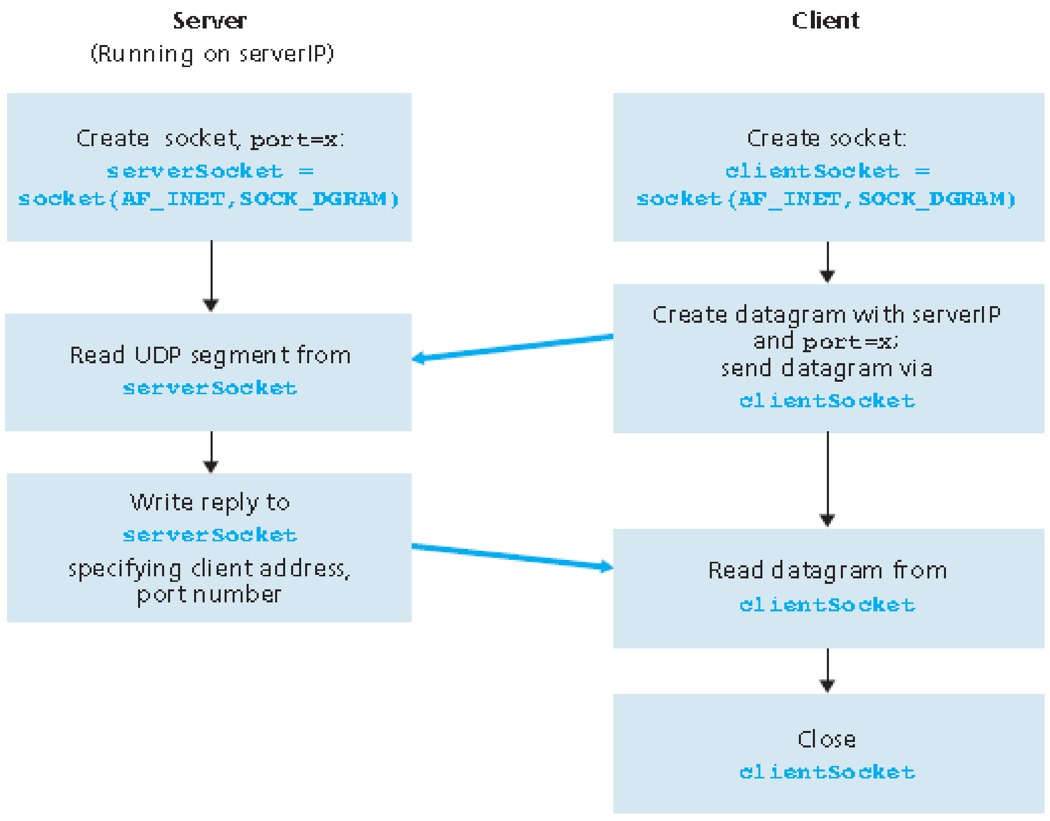

Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket = socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket = socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP and port=x; send datagram via clientSocket
Read UDP segment from serverSocket
Write reply to serverSocket specifying client address, port number
Read datagram from clientSocket
Close clientSocket

Figure 5. The client-server application using UDP

Message format

Client side

a)

For the case of “Query for content” request, the File Name could be certain file name the P2P client want to search. Also, if the P2P client want to get all directory information of the directory server, it is practicable if the File Name content is “all”.

seqNum	“%”	method	“%”	Local IP	“%”	Local Port	cr	Lf
File Name			cr	lf				

Table 1: Message one sent by P2P client: where method’s content is “Query for content”.
The size of this message <= 128 bytes

b)

For the case of “inform and update” request, actually the data P2P client going to send might large than 128 bytes, then each message fragment will add “File Name” and “File Size” part as many as possible to reach the message size of 128 bytes.

seqNum	“%”	method	“%”	Local IP		“%”	Local Port	cr	Lf
File Name		“%”	File Size	cr	lf				
File Name		“%”	File Size	cr	lf				
…		…	…	…	…				
File Name		“%”	File Size	cr	lf				

Table 2. Message two sent by P2P client, where method content is “Inform and update”
The size of this message <=128 bytes

c)

Method	“%”	Local IP	“%”	Local Port	cr	lf

Table 3: Message two sent by P2P client, where method content is “Exit”.

Directory server side:

a)

SeqNum	“200- ok”

SeqNum	“200- ok”

….

SeqNum	“200- ok”

Table 4. Message one sent by directory server to respond request “Inform and update”

b)

To respond the “Query the content”, if the client queries the whole directory, the message format as table 5.a will be responded to include all file information in the directory list when the directory list is not empty, or the message format as table 5.b will be respond when the directory list is empty. If the client queries certain file, then the message format as table 5.a (only one row) will be responded when the searched file is in the directory list, or message format as table 5.b will be responded when the searched file is not in the directory list.

Destination host IP	Destination host port#	File name	File Size	cr	Lf
Destination host IP	Destination host port#	File name	File Size	cr	lf
…	…	…	…	…	…
Destination host IP	Destination host port#	File name	File Size	cr	lf

Table 5a: Message two sent to respond request "Query the content"
The size of message <= 128 bytes

"400_ File not found"

Table 5b: Message three sent to respond request "Query the content"

c)

"Socket closed"

Table 6: Message four sent to respond request "Exit"

P2P Client to Transient P2P Server Communication

TCP protocol

Unlike UDP, TCP is a connection-oriented protocol. This means that before the client and server can start to send data to each other, they first need to handshake and establish a TCP connection. The detail process of handshake and connection establishment is showed in Figure 6.

The One end of the TCP connection is attached to the client socket and the other end is attached to a server socket. When creating the TCP connection, we associate with it the client socket address (IP address and port number) and the server socket address (IP address and port number). The segment structure of TCP is showed in Figure 7. With the TCP connection established, when one side wants to send data to the other side, it just drops the data into the TCP connection via its socket. This is different from UDP, for which the server must attach a destination address to the packet before dropping it into the socket. The detail process of the application-layer TCP communication is showed in Figure 8.

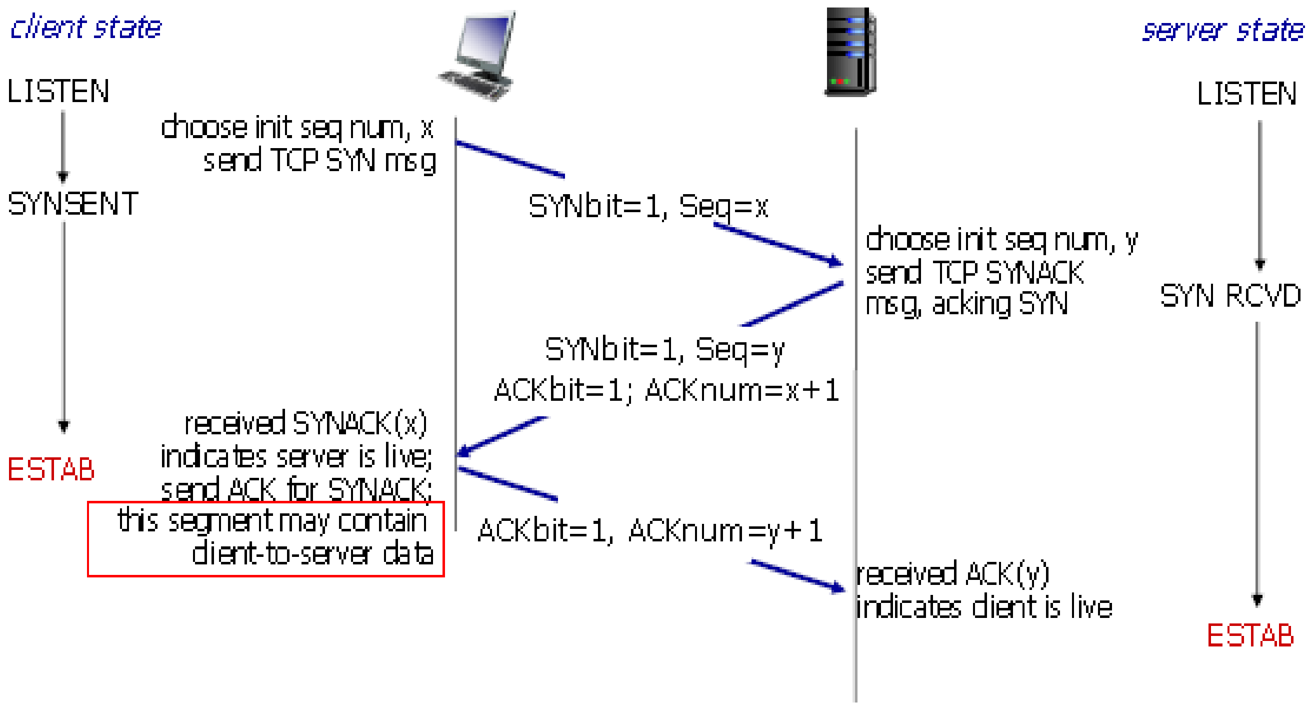


Figure 6. TCP 3 way handshake

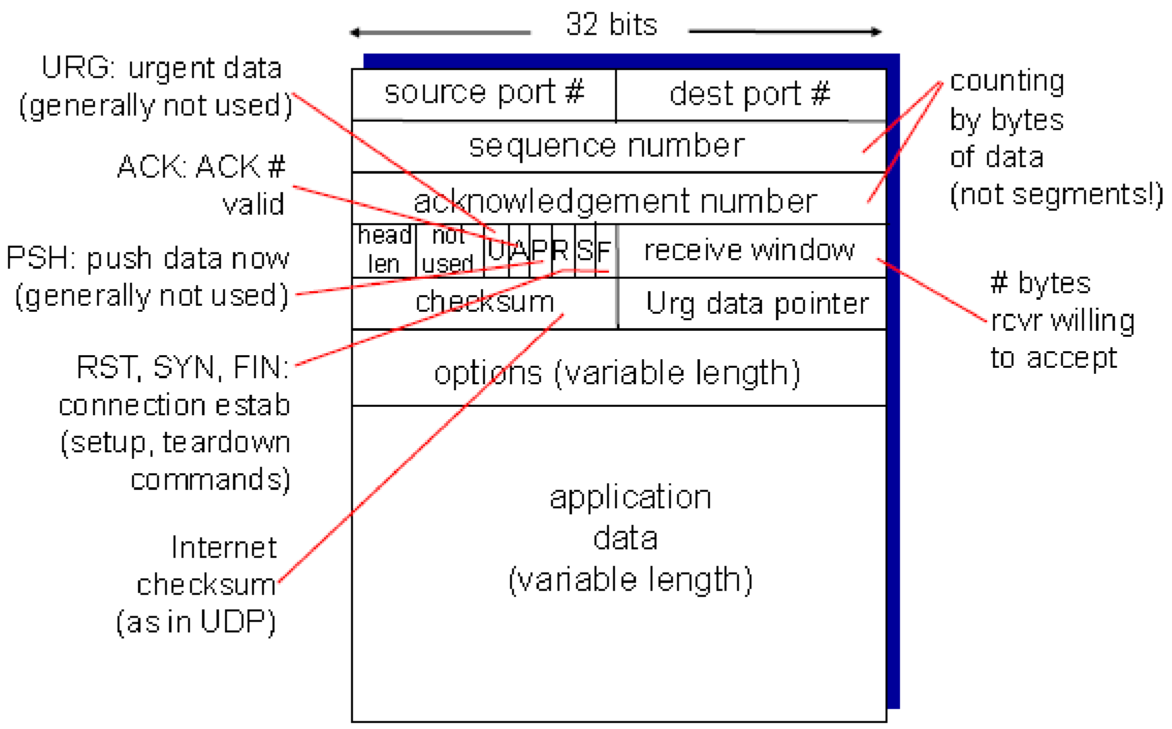

32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
U A P R S F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
URG: urgent data (generally not used)
ACK: ACK # valid
PSH: push data now (generally not used)
RST, SYN, FIN: connection estab (setup, teardown commands)
Internet checksum (as in UDP)
counting by bytes of data (not segments!)
# bytes rcvr willing to accept

Figure 7. TCP Segment structure

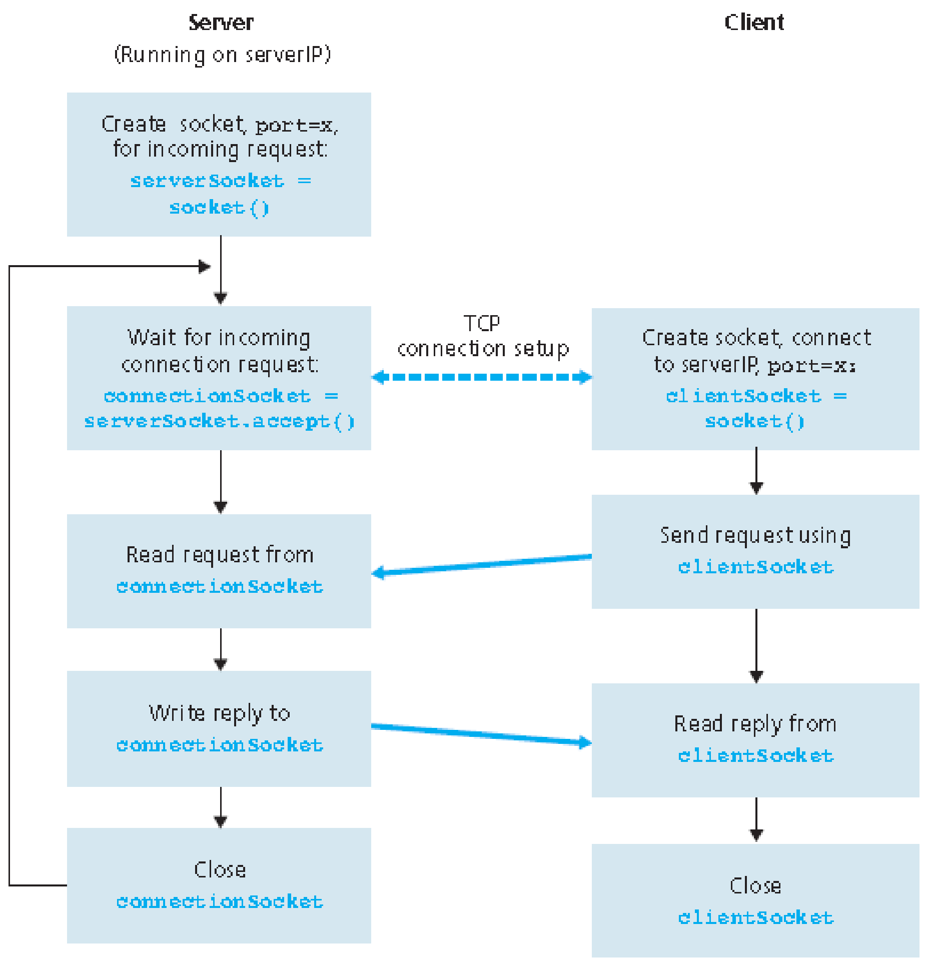

Server
(Running on serverIP)
Client
Create socket, port=x, for incoming request:
serverSocket = socket()
Wait for incoming connection request:
connectionSocket = serverSocket.accept()
TCP connection setup
Create socket, connect to serverIP, port=x:
clientSocket = socket()
Read request from connectionSocket
Send request using clientSocket
Write reply to connectionSocket
Read reply from clientSocket
Close connectionSocket
Close clientSocket

Figure 8. The server-client application using TCP

Message format

P2P client side:

"GET /"	"/" + File Name	"HTTP/ 1.1"	lf

Table 7. Message sent by P2P client to request download the file

P2P server side:

In this message, if the status code is "200", then the respond content is "OK", which means the Download is with no problem to process. If the status code is "400", then the respond content is "Bad Request". If the status code is "404", then the respond content is "Resource not found". If the status code is "505", then the response content is "version not supported".

"HTTP/ 1.1"	Status Code	respond	cr	lf

Table 8. Message sent by P2P server

Data Structure

The remarkable data structure used in this project is ArrayList, mainly used for the directory list. The ArrayList class extends Abstract List and implements the List interface. ArrayList supports dynamic arrays that can grow as needed. Standard Java arrays are of a fixed length. After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

The Directory Server is maintaining an ArrayList<FileInfo> to store the complete information of files. FileInfo is a POJO class that is having the information:

- Client port
- Client Address
- File Name
- File Size

Only unique elements will be present in the ArrayList, so that if single client inform and update multiple times, it should discard the same file that is already present in ArrayList.

Design Choice

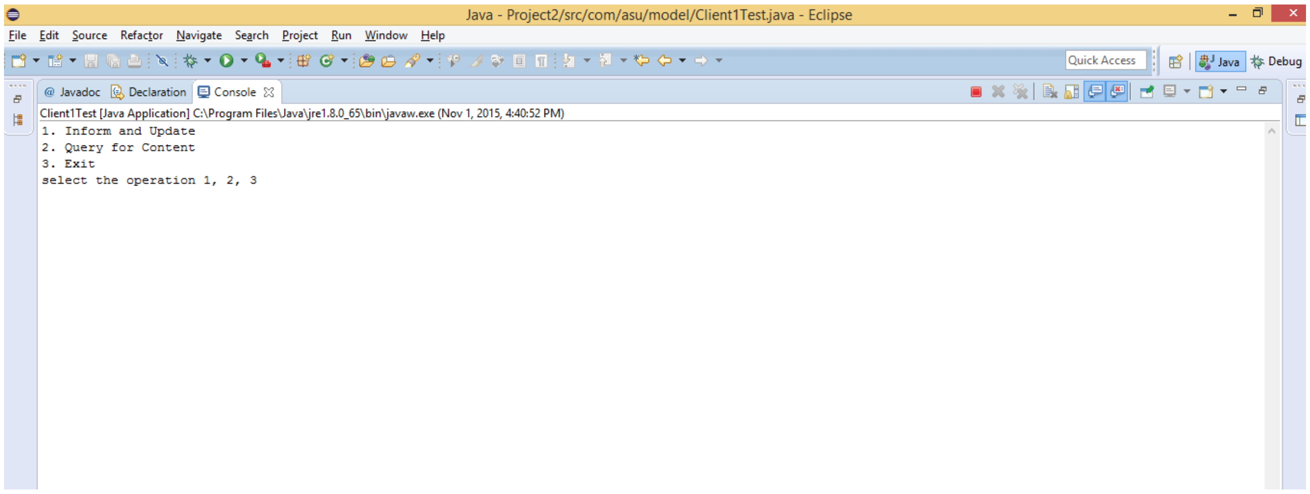
In this project, we use both UDP and TCP protocol for P2P client to directory server and P2P client to transient P2P server respectively. Since the UDP is an unreliable transfer protocol, we implemented the "Stop and Wait" model for the communication between P2P client and directory server. The P2P client sends a packet to the directory server, and stop there and wait for the response from the directory server. If the acknowledgement shows the transfer is successful, then it will send the next packet. Otherwise, it will resend the same packet again and wait for the acknowledgement. We also implemented timeout functionality in the "stop and wait" model.

Part 3. Usage Guide

The usage instruction is described by graph step by step:

Step One. Start the client

As the P2P client program starts, the client has three choices as Figure 8, the user could input the number correspond to different request. If the user inputs “1”, then the client will request to update and inform the client’s file information to the directory server and the client will head to step two. If the user inputs “2”, then the client will request to query one certain file or all directory of the directory server and the client will head to step two. If the user inputs “3”, then the client will quit the P2P system, and all the file information related to this client host will be deleted in the directory list of the directory server.

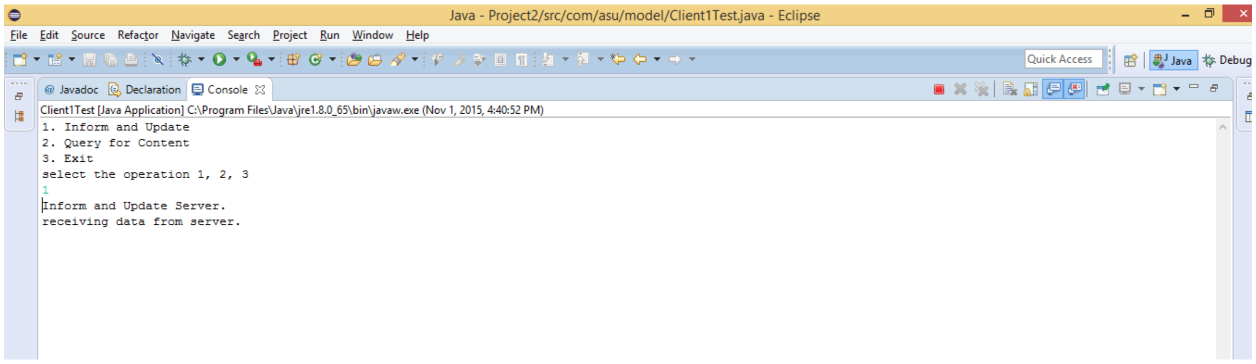

Java - Project2/src/com/asu/model/Client1Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Javadoc
Declaration
Console

```
Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 4:40:52 PM)
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
```

Figure 8. Start Window

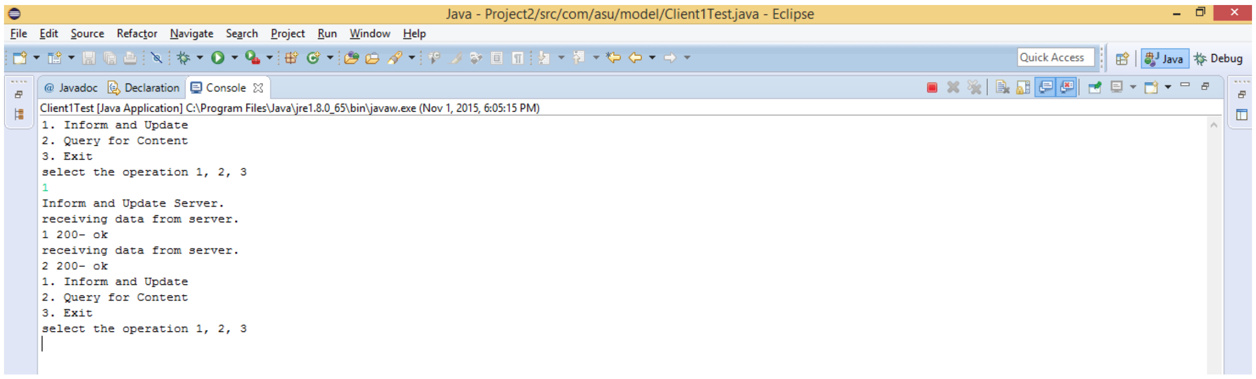
Step Two. Case 1: when “1” is input.

After the P2P client inputs “1” at first step, then one message will come to prompt the client that the inform and update is in process and waiting the response from the directory server as shown in Figure 9. After the server side update all the information in the directory list, it will send a message with the sequence number back as shown in Figure 10.


Java - Project2/src/com/asu/model/Client1Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Javadoc
Declaration
Console

```
Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 4:40:52 PM)
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
1
Inform and Update Server.
receiving data from server.
```

Figure 9. Prompt automatically



Java - Project2/src/com/asu/model/Client1Test.java - Eclipse

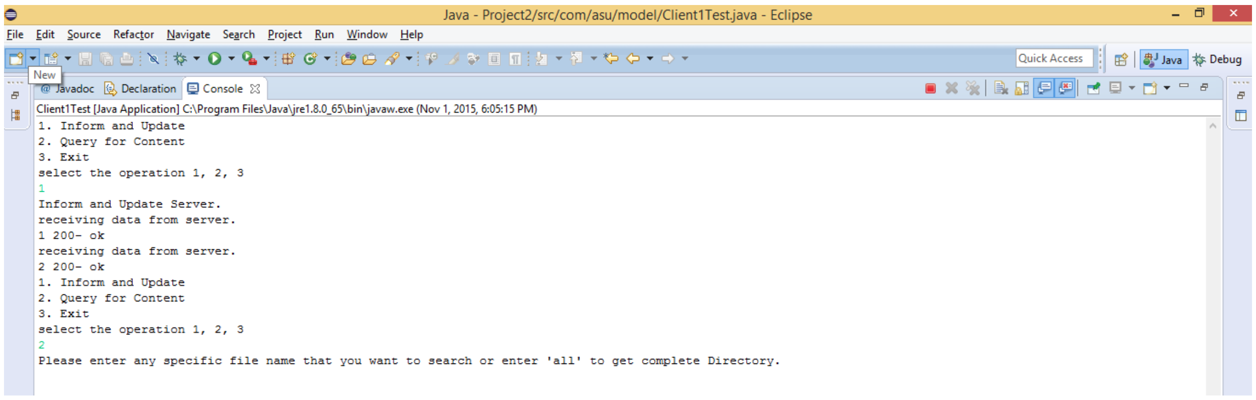
Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 6:05:15 PM)

```
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
1
Inform and Update Server.
receiving data from server.
1 200- ok
receiving data from server.
2 200- ok
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
```

Figure 10. Sequence number and "200- ok" are received

Step Two. Case 2: when "2" is input.

After the P2P client inputs "2" at the first step then a message will appear automatically(as shown in Figure 11.) to prompt the client to input the file name to query a certain file or input "all" to query the whole directory list of the directory server. Then, the directory server will respond back the IP address and host port number of the destination peer which stored the source file. Moreover, the program will prompt if the client want to build a connection to the destination peer and download the file. Yes or no could be selected (as shown in Figure 12 and Figure 13.).

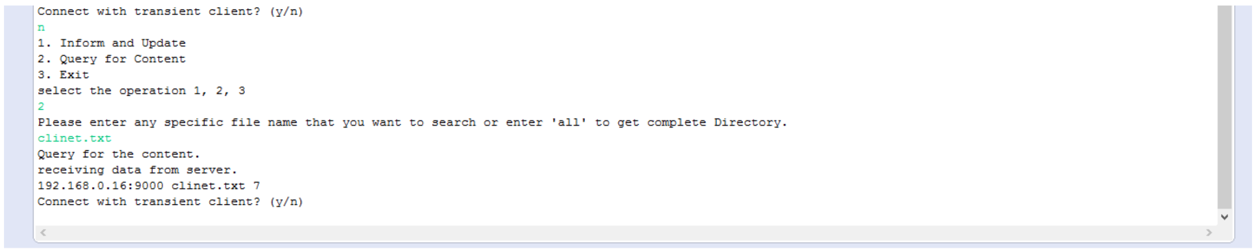


Java - Project2/src/com/asu/model/Client1Test.java - Eclipse

Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 6:05:15 PM)

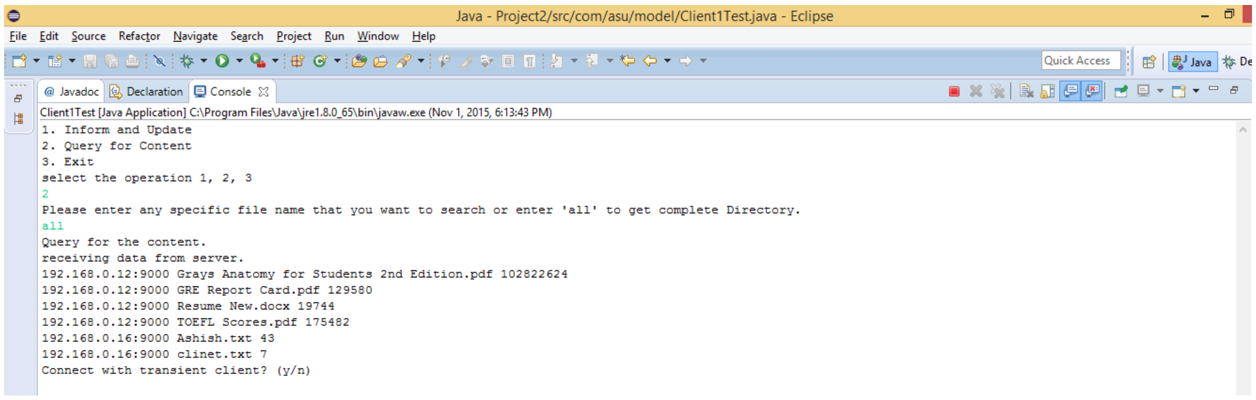
```
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
1
Inform and Update Server.
receiving data from server.
1 200- ok
receiving data from server.
2 200- ok
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
Please enter any specific file name that you want to search or enter 'all' to get complete Directory.
```

Figure 11: prompt automatically



```
Connect with transient client? (y/n)
n
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
Please enter any specific file name that you want to search or enter 'all' to get complete Directory.
clinet.txt
Query for the content.
receiving data from server.
192.168.0.16:9000 clinet.txt 7
Connect with transient client? (y/n)
```

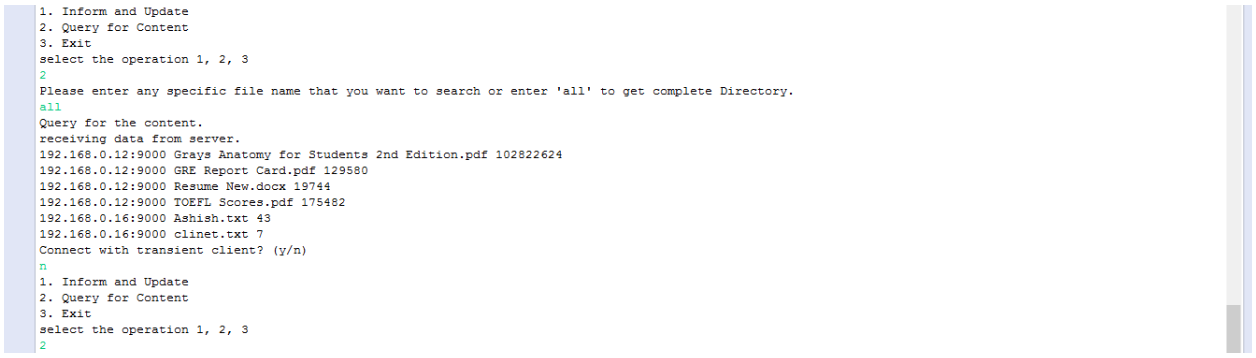
Figure 12: The respond comes from the server with the information of one certain file and prompt for the next selection


Java - Project2/src/com/asu/model/Client1Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Javadoc
Declaration
Console

```
Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 6:13:43 PM)
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
Please enter any specific file name that you want to search or enter 'all' to get complete Directory.
all
Query for the content.
receiving data from server.
192.168.0.12:9000 Grays Anatomy for Students 2nd Edition.pdf 102822624
192.168.0.12:9000 GRE Report Card.pdf 129580
192.168.0.12:9000 Resume New.docx 19744
192.168.0.12:9000 TOEFL Scores.pdf 175482
192.168.0.16:9000 Ashish.txt 43
192.168.0.16:9000 clinet.txt 7
Connect with transient client? (y/n)
```

Figure 13: The respond comes from the server with the information of all file in directory list and prompt for the next selection

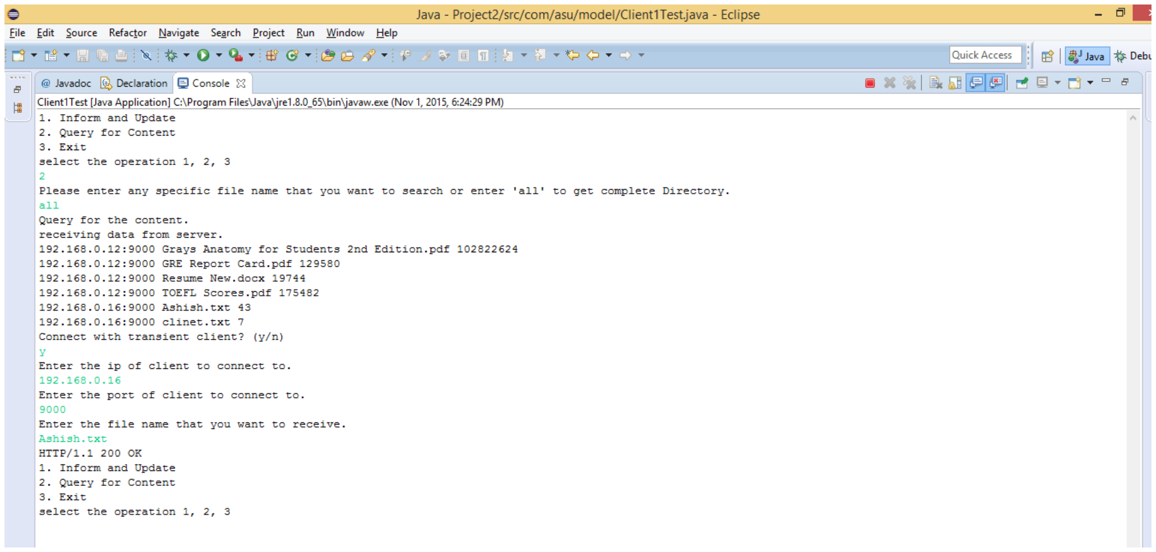
If the P2P client input "n", then the operation window will return to the beginning window as shown in Figure 14:



```
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
Please enter any specific file name that you want to search or enter 'all' to get complete Directory.
all
Query for the content.
receiving data from server.
192.168.0.12:9000 Grays Anatomy for Students 2nd Edition.pdf 102822624
192.168.0.12:9000 GRE Report Card.pdf 129580
192.168.0.12:9000 Resume New.docx 19744
192.168.0.12:9000 TOEFL Scores.pdf 175482
192.168.0.16:9000 Ashish.txt 43
192.168.0.16:9000 clinet.txt 7
Connect with transient client? (y/n)
n
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
```

Figure 14: If "n" is selected

If the P2P client inputs "y", then the operation window will appear message step by step to prompt the client to input the IP address and host port of the destination P2P server as well as the file name, if everything goes right, then the P2P client will successfully download the file it wants and get a respond "HTTP/1.1 200 OK" From the P2P server.
However, the message "400- Bad Request", "404- Resource not found", "505- version not supported" will come from P2P server if there are some problems of the download, the problem message describes the problem literally.


Java - Project2/src/com/asu/model/Client1Test.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debu
Javadoc
Declaration
Console

```
Client1Test [Java Application] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Nov 1, 2015, 6:24:29 PM)
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
2
Please enter any specific file name that you want to search or enter 'all' to get complete Directory.
all
Query for the content.
receiving data from server.
192.168.0.12:9000 Grays Anatomy for Students 2nd Edition.pdf 102822624
192.168.0.12:9000 GRE Report Card.pdf 129580
192.168.0.12:9000 Resume New.docx 19744
192.168.0.12:9000 TOEFL Scores.pdf 175482
192.168.0.16:9000 Ashish.txt 43
192.168.0.16:9000 clinet.txt 7
Connect with transient client? (y/n)
y
Enter the ip of client to connect to.
192.168.0.16
Enter the port of client to connect to.
9000
Enter the file name that you want to receive.
Ashish.txt
HTTP/1.1 200 OK
1. Inform and Update
2. Query for Content
3. Exit
select the operation 1, 2, 3
```

Figure 15: If "y" is selected and download successfully.

References

1. Authors: Choon Hoong Ding, Sarana Nutanong, and Rajkumar Buyya, P2P Networks for Content Sharing http://arxiv.org/ftp/cs/papers/0402/0402018.pdf
    1.1. Peter B.,Tim W.,Bart D. and Piet D. (2002), A Comparison of Peer-to-Peer Architectures, Broadband Communication Networks Group (IBCN), Department of Information Technology (INTEC), Ghent University, Belgium, 1-2.
    1.2. Nelson Minar (2001), Distributed Systems Topologies: Part 1, Oreilly Network, http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html
    1.3. Nelson Minar (2002), *Distributed Systems Topologies: Part 2*, Oreilly Network, http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html
    1.4. J. F. Kurose and K. W. Ross (2003), Computer Networking: A Top-Down Approach Featuring the Internet, Addison Wesley, Boston.
    1.5. B. Yang, H. Garcia-Moline (February 2002), *Designing a Super-Peer Network*, Standford University.