CS3103 Computer Networks Practice

P2P Design Report

Team Members:

Benjamin Ang	A0156065E
Dennis Heng	A0121525R
Hoang Duong	A0161371M
Yuan Quan	A0160785X
Zhang Yuanhao	A0121832R

Table of Contents

1 UI Design

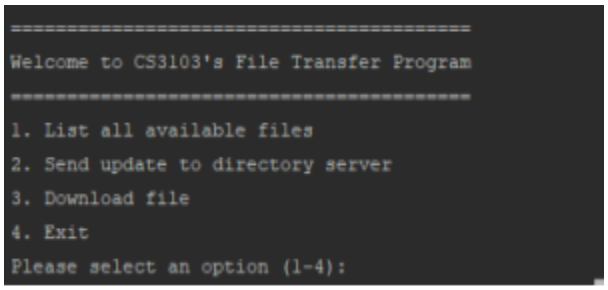

Figure 1. P2P Client initial command-line UI

Our UI for our client supports 4 actions. We decided to integrate querying the directory server & the download of a file together, as we wanted to give as much abstraction to the user as possible. The download option operates on the principle that a query command will be executed first internally to the directory server to obtain the file details first, only after the user has the file details can he then proceed to download a file.

2 Network Architecture

Assumptions

1. Peers are aware of the IP & port of the directory, relay, stun servers before they join the network.
2. All files are unique in the network.

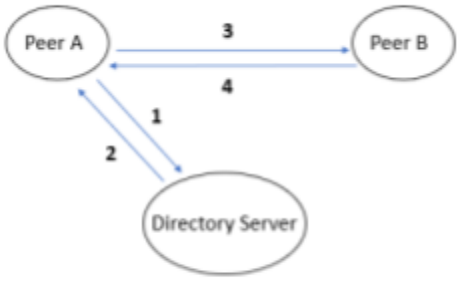

Figure 2. Network Architecture without NAT

When both sending and requesting peers are behind the same NAT, the network architecture is as above and all communications are done with TCP. In this sense, Peer A is the requesting peer and Peer B is the sending peer. Peer A would enter the download command which would result in a query message being sent to the directory server, the directory server responds to the query and sends back its reply. With the private IP and port of the peer who has the desired chunk, Peer A initiates a connection to B, who will be listening on a dedicated port for file requests. Peer A goes into receive mode to prepare to

receive data. Peer B commences the data transfer to A. Peer A now has the desired chunk and updates its internal memory that it now posses this chunk.

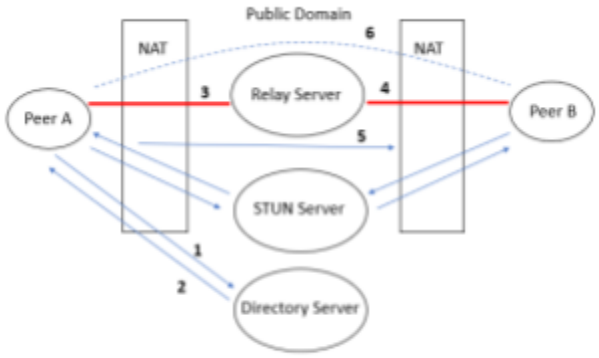

Figure 3. Network Architecture with NAT

When both the sending peer and receiving peer are behind different NATs, the network architecture is as above. In this scenario, communications are done with both UDP & TCP. In this scenario, Peer A is the requesting peer and Peer B is the sending peer. At the start, both parties obtain their public IP and port from the STUN server. Peer A would enter the download command which would result in a query message being sent to the directory server, the directory server responds to the query and sends back its reply. Having obtained B's public IP and port, Peer A will send the details of the desired chunk as well as B's public details to the relay server. With B's public details, the relay server can forward the desired file chunk details and A's public details to B. The relay server is able to forward the details to B as there is a persistent TCP connection between both parties, same goes for peer A and the relay server. This connection is stored and re-used whenever needed. The relay server knows how to re-use the correct connection as the server maintains a HashMap, the key being the public IP and port of the peer and the value being the socket connection from the relay to the peer. Peer A proceeds to do a UDP hole punch by sending a UDP packet to peer B's public IP and port, thereby creating the mapping in its NAT table. When Peer B receives the details from the relay server, it commences data transfer via UDP. If the hole punch was done successfully, the UDP packets from B should be able to reach A.

3 Message Formats

3.1 Notations and conventions used

1. All communication between client and server is done through strings.
2. The special characters “#’ & “,” has been chosen as message delimiters, with regard to 1.
3. “+” refers to the concatenation of something to a string

3.2 Program Operations

3.2.1 LIST

Client to server
Send → Command_LIST + ”#”
Command_LIST = “LIST_ALL_FILES”;

Server to Client
Send → “Index(1.,2.,3.) SPACE Filename : SPACE File size : ___Bytes”
Send →”EOP”
The number of such strings sent will be dependent on the number of file entries present on the directory server. If 5 file entries are on record, 5 such strings will be sent from server to client, each string for one file.
When all files have been listed server sends to the client a special string “EOP” symbolizing the end of data transmission from the server to the client.

3.2.2 EXIT

Client to server
Send → Command_Exit + “#” +Public IP + “#” + Public Port + “#” + Private IP + “#” + Private Port
Command_Exit = “Exit”;
To enable the directory server to be able to update it’s internal data entries. The exiting peer has to send over its own private and public details so that the server can delete all mappings associated with the peer.

Server to Client
No message will be sent in reply from the server back to the client.

3.2.3 UPDATE

Client to server

Send → Command_Update+ "#" + Public IP + "#" + Public Port + "#" + Private IP + "#" + Private Port + "#"+ Number of files to update + "#" FILE_UPDATE_1 + "#" + FILE_UPDATE_2...
Command_Update = "Update";
Number of files to update = The number of file records the peer currently has
FILE_UPDATE_X = Filename + "," File size(in bytes) + "," + File size(in chunks) + "," + ChunkMap
ChunkMap Format = "1,1,0....."
1 represents the chunk is available, 0 means otherwise. Assume that it is in order meaning ChunkMap starts from the first chunk to the last.

Server to client

Send → "Processed a total of XX entries"

Send →"EOP"

XX is the total number FILE_UPDATEs send to the server by the client.

When all files have been listed server sends to the client a special string "EOP" symbolizing the end of data from the server to the client.

3.2.4 QUERY

Client to server

Send → Command_Query + "#" File_Name

Command_Query = "QUERY_FILE";

Server to client

Send → "D#"+ Number of Chunks making up the file + "," + File Size + "," + "Public IP Public_Port Private IP Private_ Port" + "," + "Public IP Public_Port Private IP Private_Port",.....

Send →"EOP"

The public and private details after the file size each represents a peer who currently possess that particular chunk. This reply is in order, meaning the first public and private details after the "," represents the first chunk of a file.

When all files have been listed server sends to the client a special string "EOP" symbolizing the end of data from the server to the client.

4 Instructions/File Descriptions

4.1 P2P Client

1. P2PClient.java
   This is the main client program, this is the only file that needs to be compiled for the client. To compile, enter the command 'javac P2PClient.java'
2. Listener.java
   This java program is a thread which is started from P2PClient.java. It listens for data requests and it is responsible for the data transmission between peers when there is no NAT.
3. NATListen.java
   This java program is a thread which is started from P2PClient.java. It listens for data requests from the relay server and it is responsible for the data transmission between peers when there is NAT involved.

4.2 Directory Server

1. DirectoryServerListen.java
   This is the main server program for the directory server. It listens for incoming connections and will assign individual threads to deal with the connection. Hardcoded values are inserted in this program to test the P2P network. To compile, enter the command 'javac DirectoryServerListen.java'.
2. DirectoryServerThread.java
   This java program is a thread that contains the actual server logic.
3. Peer.java, Chunk.java, FileEntry.java
   Class files that are used by DirectoyServerListen and DirectoryServerThread.

4.3 NAT

1. StunAndRelayServer.java
   This java program contains both the STUN and Relay servers. They are multi-threaded as well. To compile, enter the command 'javac DirectoryServerListen.java'.

5 Program Screenshots

5.1 P2P Client Connection to Directory Server

```
YuannyPro:cs3103 Yuanhao$ java P2PClient
Private IP (port): 172.25.99.145(12345).
Public IP (port):  137.132.181.1(21562).
Please enter IP of directory server:
178.128.118.155
Please enter port of directory server:
7777
Attempting to connect to directory server at 178.128.118.155 (port 7777)...
Successfully established connection to directory server.
Starting client listener on: 3001
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4):
```

Output from P2P Client

```
root@cs3103-group1:~/test_folder/DirectoryServer# java DirectoryServerListen
Connection from: /137.132.181.1:58625
1 client(s) are connected !
```

Output from Directory Server

5.2 LIST

```
Please enter port of directory server:
7777
Attempting to connect to directory server at 178.128.118.155 (port 7777)...
Successfully established connection to directory server.
Starting client listener on: 3001
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4): 1
Files available:
1. Filename: vid1.3gpp  File size: 300000 Bytes

========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4):
```

Output from P2P Client

5.3 EXIT

```
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4): 4
Closed connection to directory server.
```

Output from P2P Client

5.4 UPDATE

```
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4):
Message sent to server: 'UPDATE#137.132.181.1#47467#172.25.99.145#3001#1#vid1.3g
pp,300000,6,1,1,1,1,1,1,#'.
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4):
```

Output from P2P Client

5.5 QUERY + DOWNLOAD

```
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4): 3
Please enter the name of the file to download: vid1.3gpp
Total number of chunks in vid1.3gpp: 6
Attempting to download chunk #1 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 50000 bytes).
Attempting to download chunk #2 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 50000 bytes).
Attempting to download chunk #3 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 50000 bytes).
Attempting to download chunk #4 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 50000 bytes).
Attempting to download chunk #5 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 50000 bytes).
Attempting to download chunk #6 directly from peer's private IP 137.132.181.1...
 complete! (downloaded 48162 bytes).
Successfully downloaded vid1.3gpp (total chunks: 6)!
========================================
Welcome to CS3103's File Transfer Program
========================================
1. List all available files
2. Send update to directory server
3. Download file
4. Exit
Please select an option (1-4):
```

Output from P2P Client

```
QUERY_FILE#vid1.3gpp
D#6,300000,137.132.181.1 4000 172.25.102.118 3001,137.132.181.1 4000 172.25.102.118 3001,137.132.181.1 4000 172.25.102.118 3001,137.132.181.1 40
37.132.181.1 4000 172.25.102.118 3001,137.132.181.1 4000 172.25.102.118 3001,
```

Output from Directory Server