Trung Nguyen
Yat Shing Pang

Network Programming Project Milestone 5
Report

Introduction

For this milestone, we have to build upon the previous milestone. Tasks include adding delay option to server, LEAVE function which remove Peer and a mechanic that forget peer who have not seen in 2 days.

Pang handled most of the user interface include forget peer who have not seen in 2 days mechanics while Trung handled most of the server and client part of the code and LEAVE function for this milestone. All the code is written in Java. The scripts are written in shell.

Technologies Used

Eclipse, Linux command line, xterm in linux cmd, various Linux text editors, Java 8, and Apache Commons CLI 1.3.1 library, ASN1 library

Formats For Data Files

For storing the data (gossips and peers), we use two simple texts files gossip1.txt and peer1.txt as default.

The format of the gossip text file is just 3 lines, as follows:
SHA_256 encoded message
Date
Message
For example:
mBHL7IKilvdcOFKR03ASvBNX//ypQkTRUvilYmB1/OY=
2017-01-09-16-18-20-001Z
Tom eats Jerry
The format of the peer text file is also 3 lines, as follows:
Name
Port
IP address
For example:
John
2356
163.118.239.68

Any additional entries would just be appended from the previous entry, in the consecutive line.

Architecture

Implementing Forget Peer

In order to forget the peer, the server will record the exact time and date the peer was added, and writes that time and date into the peer1.txt file which stores all the peers. Whenever the server starts, it checks whether or not there are existing peers, and at the same time, checks whether or not that recorded date, when compared to the current date, is 2 days more. If it is the case, then the peer is deleted from memory, and the peer1.txt is also edited such that the peer record is deleted. Since the server cannot be idle for more than 20 seconds, it is safe to assume that the peers date will only need to be checked when the server starts, because otherwise it would mean the client will have to communicate with the server every 20 seconds for 2 days straight. A script can easily achieve that, but that possibility is simply rejected because no one is going to do it anyways.

Implementing The Server

Since the server and client are encapsulated in 1 program, the server will receive the address and port from the UI.java class and also include a delay time, which delay the start of the server.

TCPThread: this class extends thread. It initializes SocketServer with passed in port. When it runs, a Socket is created to listen for connection to be made and accepts it. It will then create a thread type TCPServer which process the message that get send to the port and reply. Thread.join() is called to make sure that only one thread run at a time.

TCPServer: this class extends thread, read input using InputStreamReader and write using OutputStreamWriter. Once the server get the input, it call P_Input and pass the input and the path to the data file. P_Input process the message and return a string. The thread will send this string to the client. If the GOSSIP message get return call UDPBroadcast to broadcast the message. If the server receive “break” as a message, it will close the socket.

UDPServer: this class extends thread. It initializes DatagramSocket with passed in port. Messages that send or receive are type DatagramPacket. When server receive the message it call P_Input to process it. If the GOSSIP message get return call UDPBroadcast to broadcast the message. Since UDP server does not know if a client connect to it until it receive a message, it needs the address and port of received message to return communication. Since UDP server is connectionless it does not required to close the socket.

UDPBroadcast: initializes DatagramSocket with passed in port. Messages that send or receive are type DatagramPacket. The broadcast message get pass from the server thread that call UDPBroadcast. P_Input get call to get the list of PEERS to get the address and port to send message. After broadcast the message close the socket.

All inputs coming to the server are ASN1 encoded. The server will decode the message then process the inputs. Any output from the server will also get encoded in ASN1

ASN1 Encoding

Used ASN1 library provided by Dr. Silaghi to encode and decode message to and from the server/client. In the format:

- Gossip ::= [APPLICATION 1] EXPLICIT SEQUENCE {sha256hash OCTET STRING, timestamp GeneralizedTime, message UTF8String}
- Peer ::= [APPLICATION 2] IMPLICIT SEQUENCE {name UTF8String, port INTEGER, ip PrintableString}
- PeersQuery ::= [APPLICATION 3] IMPLICIT NULL
- PeersAnswer ::= [1] EXPLICIT SEQUENCE OF Peer

- UTF8String ::= [UNIVERSAL 12] IMPLICIT OCTET STRING
- PrintableString ::= [UNIVERSAL 19] IMPLICIT OCTET STRING
- Leave ::= [APPLICATION 4] EXPLICIT SEQUENCE {name UTF8String}

Encapsulating Server and Client

Class UI process user's options passed in like port, address, or message using "GET_OPT_UI" and pass it into server and client thread. Whenever UI starts both the server and client will start and be connected, ready to use.

User Manual

Folders:

source: contain all the source code files
bin: contain all the compiled .class files
library: contain commons-cli .jar library
data: default data folder
script: script to compile, run or test the server

To **compile** the file, run

```
bash script/compile.sh
```

To **start the server**:

On a new terminal, run:

```
bash script/scriptServer1.sh [-p [port]] [-d [data directory]]
```

To **test TCP Client**:

Server ip and port are required options, while message and timestamp are optional.
On a new terminal, run:

```
bash script/scriptClientTCP.sh -s [Server ip] -p [port] [-m [Message] -t [timestamp]]
```

To **test UDP Client**:

Server ip and port are required options, while message and timestamp are optional.
On a new terminal, run:

```
bash script/scriptClientUDP.sh -s [Server ip] -p [port] [-m [Message] -t [timestamp]]
```

To **test Witness Sever**

Make sure that the server IP and port is in the PEER file. If not, added before testing

To run Witness Server:

```
bash script/scriptServerPeerWitness.sh -p port
```

To **test leave:**

```
bash script/checkLeave.sh -s [Server ip] -p [port] [-m [Message] -t [timestamp] -d [delay]]
```

To test individual classes, go into .../bin/ run "java filename"
Class NPServer and GET_OPT call external jar library Commons-CLI which requires adding a classpath when run. The jar file is in ~/library folder. To get the full path go in ~/library, open terminal can type command "pwd" then add "/commons-cli-1.3.1.jar" to the path
For example:

```
java -cp .:path/commons-cli-1.3.1.jar NPServer
```

Add “-p port” and/or “-d data/file” to specify which port and/or data folder file user wants to use. If not specify, default port is 3333 and data file located in ~/data folder.
For example:

```
java -cp .:path/commons-cli-1.3.1.jar NPServer -p 3562 -d /home/[User]/Desktop/Test
```

To run client call

```
java -cp .:path/commons-cli-1.3.1.jar -p port -S hostname [-m message] [-t timestamp]
```

For example

```
java -cp .:path/commons-cli-1.3.1.jar -p 3333 -S 127.0.0.1 -m “hello”
```

This will connect to server at address 127.0.0.1 port 3333 with message hello. Since no timestamp declared the system will use current time as timestamp.

Conclusion

Overall, the UI will create a TCP, UDP thread and a client thread, which will start a server that could accept both TCP and UDP connection and a client (can be either TCP or UDP) that already connect to it. The user can send message, add/check/remove peers using the client UI provided. By entering running scriptClientTCP or scriptClientUDP, the data is processed and sends its output back to the NPS server.

References

https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html
https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html
http://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html?is-external=true
https://docs.oracle.com/javase/7/docs/api/java/io/BufferedWriter.html
https://docs.oracle.com/javase/7/docs/api/java/io/OutputStreamWriter.html
http://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html
http://commons.apache.org/proper/commons-cli/javadocs/api-release/index.html
http://cs.fit.edu/%7Emsilaghi/Sp2017/2017sprNPG/ASN1_GUIDE.pdf