CS3002F: NETWORKS ASSIGNMENT 2018 (STAGE 1)

Tony Guo, Martin Flanagan, Anran Chen

GXXHEN001, FLNMAR011, CHNANR001

Design:

The server application running on the server was implemented so that it should always be on – listening for any incoming client connections. Once a client connects, the server spawns a thread with the accepted socket to handle input and output with that specific client. This way with multithreading, the server will be able to manage multiple clients and listen for new incoming ones.

Once a client is validated and logged into the chat client application, the chat client GUI runs and a separate thread for handling input from server is spawned. This implementation of the chat client makes it so that the input stream's operations are separate to that of the output stream's operations. This makes the chat app responsive and interactive.

When a client starts their chat application, they are presented with a login screen. If they are new, they can click the label "I'm New" to sign up with a unique username and any password. Once they have a username, they can sign in on the login screen. The server will check their credentials and send a message back to the client, telling it whether the client's credentials were valid or not. If it wasn't valid, they will be notified (with a suitable error message) and will remain on the login screen. If their credentials were valid, they will be presented with the chat client. Here they will be able to send text messages and send files to any client online, however, they must know the client's username. They can also broadcast a message to all clients online.

The way transferring a file was implemented was to create a new socket with the server to transfer the file to the server, after which the connection was closed. For each client that accepted the file, a new socket connection was established between them and the server to transfer the file. Another socket was used to not halt operations for sending and receiving of messages. This makes the sending of the file from the server to the relevant client a background operation.

Two constraints that were taken into consideration were limited bandwidth, and authentication and confidentiality issues. With respect to limited bandwidth, clients have the option of accepting or declining the receiving of a file. With respect to authentication and confidentiality issues, a login system was implemented. A client will need a username and a password to be able to login.

The communication framework for our protocol specifies that communication speed must be that of real-time for instant interaction, and that the server must handle all messages sent over the internet first before sending it to another client. This way the reliability of exchanges will be reliable, and constant feedback from server to client will let to user know what is going.

Functionality:

Sign-up and Login with authentication:

Sign-up:

When a new user signs up they have to create a unique username (in this case we used our student numbers as they are unique) as well as a password. If all is in order, the new user's credentials are stored on the server with the other users' credentials.

Login:

The user logs in with their unique username and password and the server checks whether the credentials are correct.

Sending a text message:

Unicast:

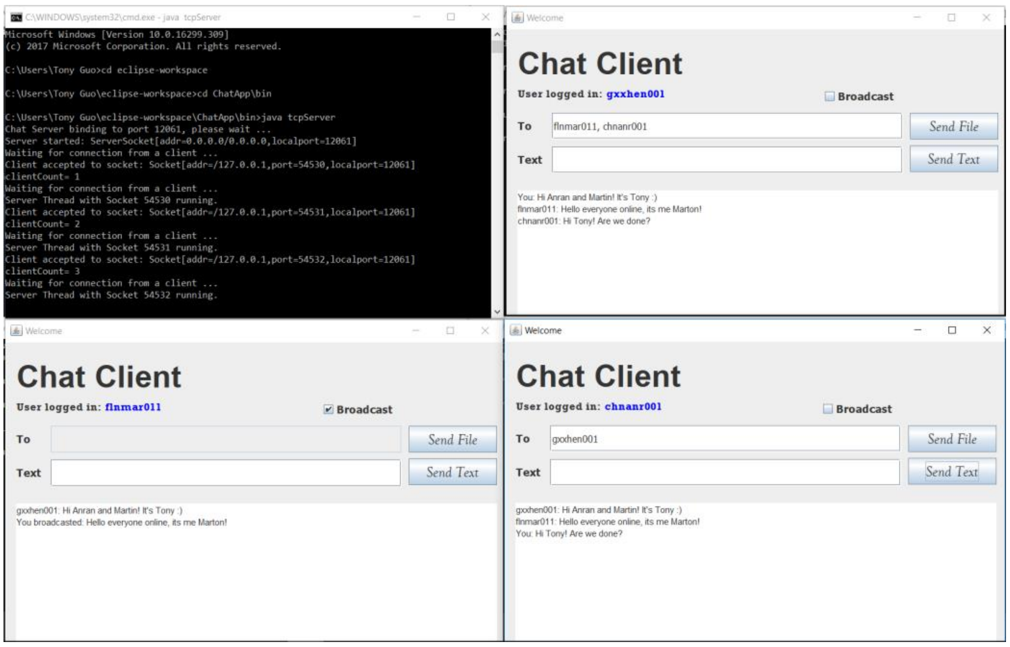
This is when a text message is sent to another user specified by the user (in the To: text field) sending the message.

Multicast:

This is when a text message is sent to other users specified by the user (in the To: text field split by commas) sending the message.

Broadcast:

This is when a text message is sent to all the users who are currently online and does not depend on the user specifying the username.



```
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Tony Guo>cd eclipse-workspace

C:\Users\Tony Guo\eclipse-workspace>cd ChatApp\bin

C:\Users\Tony Guo\eclipse-workspace\ChatApp\bin>java tcpServer
Chat Server binding to port 12061, please wait ...
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,localport=12061]
Waiting for connection from a client ...
Client accepted to socket: Socket[addr=/127.0.0.1,port=54530,localport=12061]
clientCount= 1
Waiting for connection from a client ...
Server Thread with Socket 54530 running.
Client accepted to socket: Socket[addr=/127.0.0.1,port=54531,localport=12061]
clientCount= 2
Waiting for connection from a client ...
Server Thread with Socket 54531 running.
Client accepted to socket: Socket[addr=/127.0.0.1,port=54532,localport=12061]
clientCount= 3
Waiting for connection from a client ...
Server Thread with Socket 54532 running.
```

Chat Client
User logged in: gxxhen001 ☐ Broadcast
To: flnmar011, chnanr001

```
You: Hi Anran and Martin! It's Tony :)
flnmar011: Hello everyone online, its me Marton!
chnanr001: Hi Tony! Are we done?
```

Chat Client
User logged in: flnmar011 ☑ Broadcast
To:

```
gxxhen001: Hi Anran and Martin! It's Tony :)
You broadcasted: Hello everyone online, its me Marton!
```

Chat Client
User logged in: chnanr001 ☐ Broadcast
To: gxxhen001

```
gxxhen001: Hi Anran and Martin! It's Tony :)
flnmar011: Hello everyone online, its me Marton!
You: Hi Tony! Are we done?
```

Figure 1: All possible examples of send text.

Sending a file:

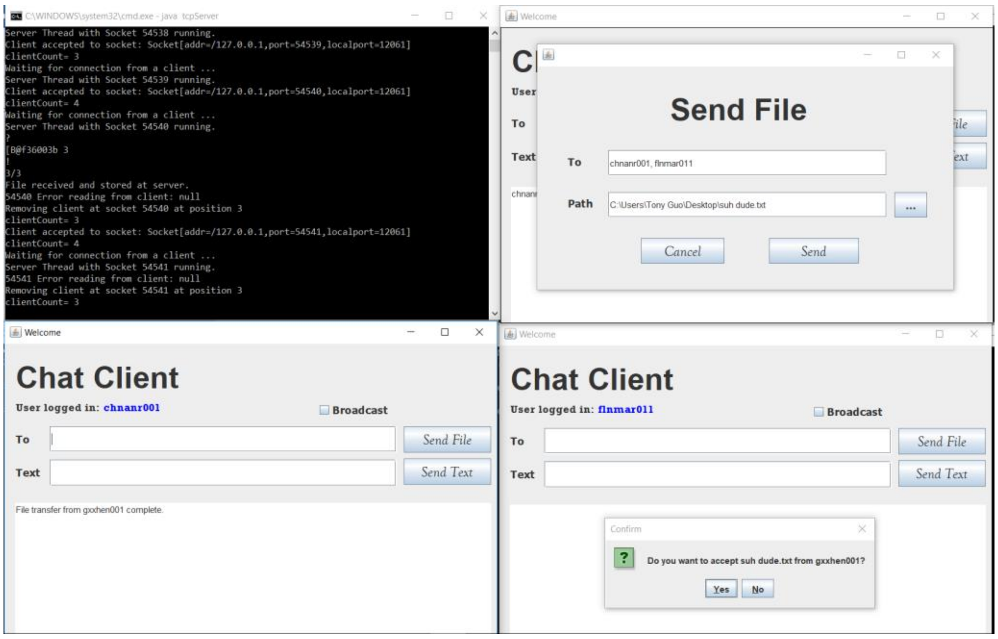
Files are sent to the Server from the sending client. The server then stores it in an array and then sends requests to all the users that the file is meant to be sent to. Once the users accept the file, the server then sends the file to all the users that have accepted to receive the file.

Unicast file sending:

This is when a file is sent to another user specified by the user (in the To: text field) sending the message.

Multicast file sending:

This is when a text message is sent to other users specified by the user (in the To: text field split by commas) sending the message.



```
Server Thread with Socket 54538 running.
Client accepted to socket: Socket[addr=/127.0.0.1,port=54539,localport=12061]
clientCount= 3
Waiting for connection from a client ...
Server Thread with Socket 54539 running.
Client accepted to socket: Socket[addr=/127.0.0.1,port=54540,localport=12061]
clientCount= 4
Waiting for connection from a client ...
Server Thread with Socket 54540 running.
?
[B@f36003b 3
!
3/3
File received and stored at server.
54540 Error reading from client: null
Removing client at socket 54540 at position 3
clientCount= 3
Client accepted to socket: Socket[addr=/127.0.0.1,port=54541,localport=12061]
clientCount= 4
Waiting for connection from a client ...
Server Thread with Socket 54541 running.
54541 Error reading from client: null
Removing client at socket 54541 at position 3
clientCount= 3
```

Send File

To: chnanr001, flnmar011

Path: C:\Users\Tony Guo\Desktop\suh dude.txt

Chat Client

User logged in: chnanr001

File transfer from gxxhen001 complete.

Chat Client

User logged in: flnmar011

Do you want to accept suh dude.txt from gxxhen001?

Figure 2: Sending a file (Multicasted).