CMPSC 176B Project Checkpoint #B: Large Scale Chat Server with File Transfer

Andrew Doan and Richard Boone

Project Description:

We are implementing a chat server for multiple clients to connect to and actively chat with each other. We would like to host multiple chatrooms on one server which clients can connect to individually. Upon connection, clients will be able to send encrypted text messages to other clients within the chatroom. They will also be able to share files with other clients who are in the chatroom. Each separate chat server instance will be able to host 10 individual clients, 5 separate chat channels, and store up to 100 unique usernames. At this time, no changes have occured in the specification of our project.

Original Timeline:

Milestone	Deadline
Initial server and single client programs with text echo	1/25/2018
Server and single client programs with file transfer echo	2/1/2018
Single chatroom support for multiple clients	2/8/2018
Single chatroom support for multiple clients with encryption	2/15/2018
Multiple chatroom support (no encryption, file transfer)	2/22/2018
Compile report B	2/24/2018
Multiple chatroom support with encryption and usernames	3/1/2018
Multiple chatroom support with file transfer and encryption	3/8/2018
Final Report:	3/10/2018
Stretch Goal:	Simple GUI/web page

Progress

As shown above, our original timeline showed us completing both single chatroom support with file transfer and encryption, and multiple chatroom support without file transfer or encryption at this point. However, we have worked ahead of schedule and have completed multiple chatroom support with file transfer, usernames, and encryption at this point. This means that we have completed the full goal of our project.

Implementation

Server Side:

On the server side, our server program listens from three different ports. Most interaction occurs through the first port, which acts as the chatroom and initial server. If the client wants to send a file to the chatroom, they are redirected to another server on the second port, where they upload the file to the server. If the client wants to download a file which another user has sent to the chatroom they are redirected to the third port, where they download the file.

The server we use uses a multithreaded approach to networking with multiple chatrooms. On startup of the server, we spawn two extra threads, one for the file receiving server and one for the file sending server. On each of these threads, and also on the main thread, our server listens for new connections. When a new connection comes in, the server spawns a new thread for the client according to which port the connection is coming over. If the connection occurs on the chatroom port, the incoming connection is asked for a username and initially assigned to the first chatroom. The server assumes all incoming traffic is encrypted. If the server receives unencrypted traffic, it will read it as garbage and close both the socket and the thread.

Client Side:

The client program is started by giving it the IP address and port number of the server's chat server, as well as the encryption key. The same AES-128 encryption key is needed on both the client and the server. The client connects to the server's chat room, and is assigned to the first chatroom after selecting a username. If the client wants to send a file to the chatroom, they type in the command "send file". The client is prompted for a file input. If the file input is valid, the client sends the command to the server. The server will send the client the port number of the file receiving server. The client will connect and upload the file, then close the connection. When the file is sent, the server announces to all clients that a file has been uploaded with its respective filename. If the client wants to receive a file, they will type the command "get file", and then the file they want after a prompt. If the file is unavailable, the server will tell the client so. If it is, the server will send the client the port number for the file sending server. The client will then connect to the file sending server and download the file.

Future Timeline

As we have worked ahead of schedule and finished the minimum viable product of our project, we will try to transition our program from using text input for various commands (typing make chatroom to make a chatroom, send file to get a file, ect.) and to use a rudimentary GUI. The GUI could possible be tkinter or wxpython, but as of now, nothing is set in stone while we research which framework to use.

We will also get performance statistics on our program as stated in our initial project proposal. These performance statistics will include RAM and CPU usage both while our program is running with many clients and while our program is transferring files.

Milestone	Deadline
Research and design GUI	3/1/19
Implement GUI, Get performance statistics	3/8/19