Project 4 Document

By Devanshu Kumar and Keerthana Madadi

Design:

After completing project 3, we were able to send packets between multiple nodes at different ports. We also set up client and server to be able to create a connection and communicate with each other.

In project 4, we built on the implementation of project 3. To use the functionalities of chat, we extended on the TCP protocol to be able to communicate effectively between various clients connected to the same server.

All of our commands start in testA.py by passing them in "setClient()". It takes care to direct the commands to its appropriate component based on its functionality. All of the commands send a struct payload to the server. This struct contains information about nodes, ports, message, and usernames. The server can use this struct to set up the chat features.

For the client to connect to the chat server, we used the previously implemented three way handshake as the basis. We set up the client to connect to the server's particular port through a three way handshake. We also made sure that the chat server keeps track of all the connected clients when the connection is completed by creating a list of all the connected clients. This functionality can be used using the "hello" command.

For a client to broadcast a message to all the clients connected to the chat server, the client sends the message to the server. Then the server sends the same message and sender client's user name to all the clients connected to it. The message sent by the client is basically redirected to other clients instead of the chat server displaying it. The server acts as a transport for the message instead as a receiver of the message. This functionality can be used using the "msg" command.

For a client to unicast a message to a particular client connected to the chat server, the client has to send the message and the receiving client's username. The chat server then searches its list of connected clients for this particular client. If it finds this client in its list, the chat server sends the message and the sender client's username only to that specific client. This functionality can be used using the "whisper" command.

Lastly, a client can request the list of all the connected clients from the chat user with "listUsr" command. The client requests the chat user for the list of all its connected clients. Then the chat

server sends the list back to the client where the client can view the entire list. However, the server only sends the list of connected clients' usernames instead of their node and port information.

Discussion Questions:

1) An advantage of this would be that it would have simplified the logic for determining the outputs at the client's end but it would require more logic during startup and more memory allocation for supporting multiple sockets per client.
2) Our ways of passing the payload and storing client information are good fit for the chat client as we were able to pass multiple transport information of different data types and uniquely store information about the client in the server end. The feature that was not good was our timer layouts as in some occasions, we miss to retransmit due to run time. The best way to resolve them is to maintain a state of the retransmit state and provide implementations in the TCP.fired() function.
3) Our transport protocol is not ideal to be successful in HTTP because we have a limited range of sockets and we have hardcoded a few servers, port locations for transport purposes. To fix them, we need to spend time on making the server interaction dynamic. However, our transport layer is good for sending packets back and forth.
4) I would spend time working on placing the retransmission timers properly as this has been a major challenge for us to recover from lost packets. There are instances where due to packet lost, we are receiving incomplete data in packets and it causes our program to crash. This scenario, however, is rare given that we give it a good run time.
5)