Raivat Alwar, Ritesh Patro
Project 4 Design Process

Chat Client:

To go about implementing the Chat Client, the very first thing that had to be done was to create a way that the node would be able to determine which “function”, “broadcast”, “whisper”, ie. it needed to be initiated. I initially planned on creating multiple python commands to deal with this but I decided to use one method instead which would then take the first few characters in the payload and cross check it with the characters in each function call. By function call, I am referring to “hello ”, “msg “, “whisper “, etc. In the case that the command is “hello”, I then created a helper method that would go to the end of the payload and extract each numerical character and convert it into an integer to store as the client port. For all the commands given I implemented them in a way that it would call my Transport layer with the payload contents and then get sent to the server.

Server:

When the server receives the request from the client, similar to the client, it needs to determine what function is being asked of it. This check occurs after the message has been fully received by the server in DataTimer.fired() by cross checking the first few characters of the received payload with the characters in each command. If the command happens to be “hello”, all the server does is extract the characters in between the command phrase and the client port then insert those into a 2D array. This 2D array holds the username with each row referring to a new user. This row of the username will then act as a key in a hashtable to get the node that each username belongs to. I also have a second hashtable created that uses the node as the key and the clients port number as its entry. Getting the client port from the payload is done in a similar way within the server as the client with the only difference being how I traverse the payload. When I get a payload in to my ChatClientP file, the payload comes in as bits hence I have to move my iterator by 8*(the index in the payload). In the case I receive the “broadcast” command, I have a timer created that will loop through the username’s nodes and transmit the message to the next node after the previous transmission is complete. For the command “whisper”, I get the username from the payload then look up its row number in my 2D array. This row number then lets me access the node and the client port of the user to get to allowing me to send them the whispered message. For the “listusr” command, I create a new payload with the value “listUsrRply” already stored in it. I then iterate through each user in the 2D array and append them to this new payload with a space in between each user. After adding all the known users, I then send it back to the same node I received this command from. Common across all commands except “hello”, one issue I had was that the server was unable to transmit packets to the clients. To get around this, instead of originally having my server open all its sockets, I have it only utilize half its sockets as a server and instead save the other half for any extra connections it needs to make when transmitting a message to a client. In essence, this causes my server in part to act as a client when it sends the message out while my clients in part act as servers when they receive the message.