UCMerced Fall 2024 - CSE160 - Computer Networks
Project 4 - Chat Client and Server
Project Team: CJ Cox, Aaron Nam
Professor Cerpa

Design Process & Decisions

In Project 4, For our design process, the goal was to work with what we already had from Project 3. We added a new struct and variables to our ‘tcp_flag’ and ‘socket_store_t.’ We decided to add two new flags ‘MSG_START’ and ‘MSG_END.’ They are used in our switch cases in our Receive function that helps flag when the message starts and signal the end of the message. We added the use of ‘enum msg_type sendType’, and ‘rcvType’ from our ‘socket_store_t.’ We use these to allow us to know what type of message is either being sent or received. We decided to use three new command functions: serverStart(), clientStart(), and send(). The first function, serverStart(), will create a server-side socket and the destination will be set to itself. In the clientStart() function, we set the destination to the server address and we make itself the source. This would initiate the handshake. Once that’s done with, the client can use send(), which will start by sending a tcp_pack with the MSG_START flag, that contains the message type and destination. Once the server receives this and sends an ACK, the client will send a message to the server as normal, and depending on the message type (BROADCAST, UNICAST, GET_LIST), it’ll send the appropriate information to the appropriate source. In order to track all this extra information, we also created a cache that will keep track of all this information.

On the python side, we created three functions: broadcast(), unicast(), and getList() to correspond to each of the new send types. For testing, we’re using 1 server node, and 3 client nodes. We can ensure our functions work when broadcasting sends a message to node, unicast sends a message to only one node, and getList will return the users lists to only the node that requested it.

UCMerced Fall 2024 - CSE160 - Computer Networks
Project 4 - Chat Client and Server
Project Team: CJ Cox, Aaron Nam
Professor Cerpa

Discussion Questions

Discussion Questions

1. The chat client and server application as described above uses a single transport connection in each direction per client. A different design would use a transport connection per command and reply. Describe the pros and cons of these two designs.

For single transport connections in each direction per client, the pros are being a fewer amount of sockets needed. It would be more efficient for real-time chatting because it won't need to establish new connections. The cons for single transport connections would be a single point of failure can cause the connection to drop being centralized and it would probably require constant retransmission.

The other design transport connection per command and reply, the pros is that there is a simpler state of management for each connection. If a scenario occurs where one of the commands failed to execute it wont effect the other transport connections. The con of this design is that since transport connections per command and reply

2. Describe which features of your transport protocol are a good fit for the chat client and server application, and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?

Our transport protocol is great for the chat client and server application because it implements functions to handle the 'hello', 'msg', 'whisper', and 'listusr' commands. Currently the features that are problematic would be the static sliding window that might cause limitations of data transfers. We have a fixed buffer size that could shorten the length of messages being sent and received.

3. Even if you did not implement the extra credit application, read its protocol specification. Describe which features of your transport protocol are a good fit for the web server application, and which are not. Are the features that are not a good fit simply unnecessary, or are they problematic, and why? If problematic, how can we best deal with them?

The features of our transport protocol that would best fit a web server application would be the basic work flow of how our Transport Layer handles sockets and transfers data. There is a reliable delivery that would retransmit if packets are lost and handshake is very reliable due to having checks and retransmissions. The not good parts of our transport protocol are having one sliding window which is not dynamic and currently no congestion control. The best way to deal with this problem is to create a dynamic sliding window.

4. Describe one way in which you would like to improve your design.

To improve our design overall would be to handle more robust errors, due to some bugs within the transport layer sometimes there would be a bug that causes the transport layer to not send correctly. For example, “Send SYN,” would be the biggest bug we have right now and that could be due to a loop associated with the handshake. This could happen due to the socket states not being set correctly.