Student Name: Luyang Ye
Student ID: z5280537

COMP3331 Assignment Report

How the system work

Basucally, it is a multi-threaded code. There are mainly recv_thread, send_thread for UDP connection, tcp_thread for TCP connection and input_thread used to deal with input from terminal. Ideally, these threads will not effect each other. The two threads used for UDP connection is mainly used to deal with the ping messages. The input_thread will take the input from the terminal as command and send message through TCP connection based on these command. The tcp_thread will act based on the message send to it. I referenced the multi_thread code from the course website to design their basic structure.
I use ‘Subscribe’ and ‘Unsubscribe’ messages through the UDP connection to control the ping message, and the peer join and quit command also used these 2 kinds of messages. All other messages are send through TCP connection. By default, all the messages should be strings, and each argv separate by a space like ‘ ‘, so I can split the message and simply send more information through one message.
The data retrieval command must request a exist file in the directory, otherwise nothing will be transmit. The peer who hold the file will read the data and send all the data to the peer who request the file, then the request file will open a new file called ‘received_’ + the original filename and write the data received to the new file. Due to the time reason, I just test the report.odt file, other types file may be failed.

Possible improvements

There are mainly 3 possible improvements I can imagine now.
First, since the ping message, ‘subscribe’ message, and the ‘Unsubscribe’ message is sent by UDP connection, sometimes the message may be lost. If I had plenty of time, I will try to seperate the ping message and the ‘Subscribe message and use TCP connection to send the ‘Subscribe’ and ‘Unsubscribe’ message.
Second, for peer joining, the program now can just show the required command in the terminal and ask the user to type it in another Xterm by hand. The main reason is that, although I can ask the terminal who shows the command to implement the command directly, since the ping message must be sent for every ping interval, I’m afraid the terminal who print and run the command will have to hold the Xterm. Then the terminal cannot work as a peer. If I have more time, I will try to write a script which can take the printed command as input and let the new peer join. If so, I can just run the script, and close the script after finish peer join.
Third, I choose to handle the KeyboardInterrupt and SystemExist of the thread to implement peer departure(abrupt). However, sometime other errors will cause problem of that. If I have more time, I will try to handle more exceptions to avoid this.

Particular circumstance

Sometimes ‘Address already’ in use will happened, I try to free the port or change the peer name and peer id to solve this problem, but that not always work.
Sometimes ‘connection refused’ will happen, I try to free the port or change the peer name and peer id to solve this problem, but that not always work.
Sometimes ‘UnicodeDecodeError’, I try to edit the messages in the program, but there’s no guarantee this will not happen any more. I also try to restart from the peer initialisation, and that should work.
After the quit command, if user do not close the window, indexErro and badfile descriptor will happen. But since all the sockets is already closed, that should not influence other peers.