Program Design:

In my system,I have two side Server side and Client side.I use UDP and TCP protocol to deal with the command and file transmission.I code by using python.The UDP socket is used to deal with all the command except UPD and DWN.The TCP socket is used to do the transmission.The TCP socket would be closed once the sever finished up the transmission of the file.The server uses select to control whether to use UDP socket or TCP socket.I write the handle function to deal with the UDP command, and function validate_message_owner_and_format to validate the whether the command is a valid request and validate the belonging of each message.

I use dictionary (active_users, client_states, file_user_container) to store the status of the user and progress of authentication .

```
active_users = {('127.0.0.1', 50001): “yona”}
client_states = {('127.0.0.1', 50002): {"stage": "waiting_password", "username": "daddy"}}
file_user_container = {
    IP: {
        "username": username,
        "thread_name": thread,
        "file_name": file,
        "type": "upload" or "download",
        "udp_address": (ip, port)
    }
}
```

And I deal with the all the format normalization in client side to make sure that all of the command message which sent to the server side is normalized and in correct format.In this case,the server could focus on dealing with the command itself.

Application layer and how my system works:

First of all, all of the command message would be sent by client side.And the client side would make sure that the message is in the correct format such as the number of the parameter or whether the string is legal or not.

Then the client side would send UDP socket to the server and the handle timeout function would deal with the retransmit mechanism.If the server didn’t response in 2 seconds.The client side would retransmit the request and wait for the response from the server.But if it still not works.We would restart the connection.

When the server receive the UDP message,it would split it and use the thread name,username,IP address,message number as the parameter in different functions to deal with each situation.

I record the IP address and username in active_users.And client_states is used to store the stage of each active user. And file_user_container is used to record the

key info of the UDP message.So the TCP connection would use the info from the UDP socket to help build connection with server side.

For the file transmission mechanism.The UDP client would first send the request for upload or download.When the server get the request.It will store the information of the file name and thread name into the file_user_container.Then the server would start TCP connection with client side and receive the file from client side or send the existing file to the client side to satisfy the needs of download.

And for each command or transmission,either the client side or the server side would receive the prompt statement of each step.The server would have the record of each step of users.And the user side would receive the response message from the server to know about the situation.And I use if and else to deal with each exception case.

Tradeoffs and Multiple Concurrent Clients

I use threading.thread to help server server multiple users.The select function could decide whether it is a UDP or TCP socket.And I design the function for handle both UDP and TCP socket.And I embedded quite a lot of functions in the server.This can result in code that is somewhat redundant and not well encapsulated.It reduces code clarity and reusability, making maintenance more complex in larger-scale deployments.

To ensure thread safety during concurrent file access, I implemented a lock mechanism using Python’s threading.Lock. Each thread (Concurrency_control_locks, which is itself protected by Concurrency_control_locks_lock to manage concurrent access to the dictionary. This mechanism guarantees multiple user: but only one user can read or write to a thread file at one time. While this prevents data corruption, it sacrifices multiple users readability, which limits scalability and responsiveness in a high-read workload scenario.

In total ,I think I have finished up all the function which required in the task.