Group Member Name(s): Tyson-Tien Nguyen

Group Member Student ID(s): 921031514

Initial Attempt and Outcome: The initial attempt involved a lot of research on how TCP sockets are used and noting down what a proxy server does at its core. I again used a lot of the Real Python website to help me understand the flowchart/process of TCP when using sockets and referred back to the socket Python documentation to understand the different functions that are used between TCP and UDP. I first partially implemented the TCP server, simply creating the socket, binding it, and then having the server socket listen for the proxy server to connect to the TCP server. Once the proxy server connects to the TCP server, the server would accept the connection. Now for the client, we created the client socket, the JSON block of data, and connected the client socket to the proxy server. The proxy server would then accept the connection from the client and parse the data. I implemented the IP filtering in the proxy server where it checks the server IP and makes sure it isn't in the tuples. If it's all clear, the proxy server would forward the client's message to the server, else the proxy server would send an error message back to the client. The main issue that I had difficulty with was having the proxy server establish two different connections. One with the client and the other with the server. This problem tripped me up as I didn't understand how we can have two simultaneous connections. But I referred back to the Real Python resource as well as the socket Python documentation. And learned that I couldn't establish two different connections with one socket, as one socket can only perform only one task, be it a listening socket or a speaking socket. And so that gave me the idea to create another socket within my already established connection with the client and have this socket be the speaking socket for the proxy server. Allowing me to officially contact the

server, establish a connection with the server, and allow me to have two different simultaneous connections up and running. After that, I implemented the control flow for the rest of the program, having the message from the client forwarded to the server. The server would then extract the proxy server IP, send the reply message it wants to the client via the proxy route, and automatically close the connection. The proxy server would receive the message reply from the server and then relay that back to the client. Additionally, as stated before, the proxy server has a dictionary that contains two tuples inside of it. And those two tuples contain banned IP addresses and ports as well. And so after finishing up the program, the outcome was a functioning client to proxy server to server and in the reverse order as well. Where banned IP addresses prevented the message from being forwarded to the destination server. And the user can input any 4-letter string that they want to send to the server.

How to Run Code: First, we always run the server[Tyson-Tien Nguyen]_[921031514].py file first, then we run the proxy_server[Tyson-Tien Nguyen]_[921031514].py, and then we run client[Tyson-Tien Nguyen][921031514].py. Input 4-letter string and let the program run to see the responses from the Client, Proxy Server, and Server. Additionally, you can alter the “server_ip” and “server_port” to test out the Proxy Server IP filtering.

Expected Results: If the Server IP is not banned, then the Proxy Server forwards the message to the Server, if it is banned, then the Proxy Server sends an error message back to the Client. The print statements will help elaborate in more detail what each component of the communication line does.

Name of Files to Run: server[Tyson-Tien Nguyen]_[921031514].py file first, then proxy_server[Tyson-Tien Nguyen]_[921031514].py, and then client[Tyson-Tien Nguyen][921031514].py.

Name of the Report Files: proj1_2_[Tyson-Tien Nguyen]_[921031514].