Running the programs:

The echo server and client (tcp_server.py and tcp_client.py) can just be run with "python3 tcp_server.py [port number]" and "python3 tcp_client.py [hostname/IP] [port number]". The code loops infinitely, allowing the user to send as many messages as they want to be echoed back. Sending the message "quit" causes the client program to end. The server program notes that the connection ended but will keep awaiting new connections.

For part 2 (tcp_server_pt2.py and tcp_client_pt2.py), run the command "python3 tcp_server_pt2.py [port number]" and "python3 tcp_client_pt2.py [hostname/IP] [port number]". The server will immediately begin listening while the client program will prompt you for more information about the experiment to run.

Testing:

I didn't focus too much on testing part 1 of my code. I essentially just verified that some randomly selected messages would be echoed correctly. Most of my testing was conducted on the code for part 2 of this assignment, specifically on the server side. For the client side, I ran all of these experiments against the provided servers to ensure it was built correctly (e.g., could handle 404s). First, I tested at least one error for each chunk of the startup message to ensure that the server would throw a 404 error: not beginning the message with the letter 's', giving a measurement type other than "rtt" or "tput", setting the number of probes < 1, setting the message size to a value not allowed by the measurement type, giving a server delay of less than zero, and messing with the white space. Next I ensured that sending two setup messages in a row to ensure the server wouldn't tolerate this either. During the measurement phase, I checked that the following errors caused a 404 to be thrown: not beginning the message with 'm', **t**he sequence numbers were always increasing, the payload size was the same as the size declared in the setup, and white space was all correct. Finally, during the termination stage I checked that the sequence number of the last message was equal to the number of probes declared in the setup message, and that the message was exactly "t\n".

With all of those tests passing, I felt pretty confident about my program for the majority of cases but I also added a few edge case checks such as ensuring that the number of probes and message size are both integers and that server delay is in fact a number. I also caught an error where after a connection was established, sending a message with a sequence number that wasn't an integer would cause the server to lock up but I added a check for that as well.