Checkpoint 3 WebSocket

Part 1 Environment Set up

Server (on your own computer):

1. download the "***websocketd***" compatible with your operating system from http://websocketd.com/#download
2. move the executable (i.e., ***websocketd.exe*** if you are using windows) to a folder (for example - ***D:\websocketd***).
3. Put the ***wsd_server.py*** file from the boilerplate folder into the same folder as step 2.
4. create a file named "Status" in the same folder. The Status file contains only a number **0** or **1**. Set the initial value to **0**.
5. open Command Prompt, change the working directory to that folder (***D:\websocktd***)
6. run server: `websocketd --port=8080 --devconsole python wsd_server.py`

Client (on the Raspberry Pi VM):

1. From the command line, execute the following command to install the library.
2. **pip install websocket_client**
3. put **ws_client.py** from the boilerplate folder into your home folder;
4. change the server's IP address and port in the **ws_client.py** to the IP address of your host machine using port **8080**. Example - '**<your_ip_address>:8080**'
5. run the client from the command line: **python ws_client.py**

Part 2 Exploring WebSocket functionalities and Performance.

Step 1: Based on the boilerplate, implement the following functions for the client and the server:

Client side:

1. generate a random number every 10 seconds and send it to the server. The message should follow the format of: "***Heartbeat: [timestamp] - [num]***". Please note that all timestamps should include the milliseconds.
2. Upon receiving a message from the server, the client sends a confirmation message back to the server. The message should follow the format of "ACK: [timestamp] - [seq]", where [seq] is the sequence number of the server's message.

Server side:

3. Read the content of the "Status" file every second. If a change of the number is detected, send the updated number to the client. The message should follow the format of "***Status: [timestamp] - [seq] - [status]***". ***[seq]*** is an auto-increment integer, starting from 100.
4. Store the message it receives from the client in a file named "***serverLog***".

Step 2: run the code, measure the performance, and answer the following questions:

1. Connect the client to the server. How many packets need to be exchanged between the client and the server? How many bytes in size do the packets add up to?

2. Observe the communication between the client and the server. How many packets are sent in a period of one minutes? What are these packets for? What is the average size of these packets? (the result may vary, just base your answers on your observation).
3. Manually change the content of the "***Status***" file to "***1"***. Measure the round trip latency on the server side by subtracting the timestamp of the original status message from the timestamp of the "ACK" received by the server from the client.
4. Disconnect the client from the server. How many packets need to be exchanged between the client and the server? How many bytes in size do the packets add up to?