Group Member Name(s): Tyson-Tien Nguyen

Group Member Student ID(s): 921031514

Initial Attempt:

My initial attempt involves reading up on the Python Socket API documentation to understand how it's supposed to be used. Additionally, I also read up on some example usages of the Python Socket API where they implemented a TCP Server/Client version instead. I noted that SOCK_DGRAM was for UDP and SOCK_STREAM was for TCP. After learning the basics of the Python Socket API, I searched up a more in depth guide on how to use it properly and stumbled upon the website RealPython where they were using the Python Socket API to implement their take on a TCP Server/Client. And I focused on the flowchart that they had for the TCP Client/Server and understood how the three-way handshake was implemented. However, I recalled that UDP didn't need to establish a connection to start sending/receiving data and so I searched up a UDP explanation and went to GeeksForGeeks to do more research on the process of a UDP Server/Client. And they confirmed that UDP didn't need to use the listen() function as well as the accept() function of the Python Socket API and can immediately start sending/receiving datagrams. And I saw how simple UDP was compared to TCP in terms of code implementation. So as I started implementing the UDP server first, I created the socket and bind it to my localhost and an unused port number on my computer, in this case 127.0.0.1 and 65432 respectively. I then created a variable to keep track of the total bytes of message that has been received from the client as well as a boolean transmission flag to keep track of when the client would initially start transmitting the payload to the server. Then, I used a while statement to make the UDP server wait for the UDP client to start transmitting data. Once the UDP Client

starts transmitting data, that's when I log down the transmission time and set the transmission flag to true, telling the UDP server that the UDP client has started transmission. Then, once the UDP client has finished transmitting the packet, that's when the UDP Server checks for the "Finish" string, and if it's there, then that means the UDP client has finished the transmission, if it's not then the UDP Client still has more to send. Once transmission has been confirmed to be done, the final transmission time is logged and the UDP server prints out the amount of data it has received as well as the addresses of the client and itself, and the timestamp. Next, the UDP server calculates the throughput in bytes per second as the units, packs up the double/encode it into bytes, and sends this information back to the client.

Looking at the UDP client implementation now, I have used the input() function to take in any user input from the terminal, convert that user input into an integer and then multiply it by 1024 twice in order to get the MB units. The host and port is the same as the UDP Client is on the local host as well, but the port for the UDP Client is assigned randomly to whatever port is available. The payload is then generated with the message "Hello World!" in bytes and we send this a bunch of times to the UDP server until it reaches the total bytes that the user has inputted into the UDP client terminal. This is so that we don't overload the UDP Server with a big payload and servers as a way to "fragment" the data. Additionally, when the first fragment is sent, that's when we log down the start of the transmission that will be used later as a timestamp. Once all the fragmented data has been sent, we also include an end marker at the end of the message to tell the UDP server that the UDP client has finished transmitting. We then print into the terminal that the UDP client has finished transmitting all of its packet to the UDP server and displays information about the addresses of both the client and the server, as well as the amount of bytes sent and the timestamp of the start of the transmission time. Once that is done, we

receive the throughput information back from the UDP server, unpack the throughput/decode it, and convert the units bytes per second to kilobytes per second. We then print the throughput in kilobytes per second into the terminal for the user to see.

Additionally, though my explanation for my implementation is a bit broad, I commented a majority of my lines of code to explain my thought process as well, so the comments in my UDP server and UDP client implementation goes into further detail on what a majority of each code of line does.

LLM Usage:

For this project, I chose not to use an LLM at all since I believe that it will hinder my ability to learn how to code properly as well as learn how to cope with the struggle of not knowing the full requirements of how the project should look like. Furthermore, in my belief, after using LLMs in the past, I found that I have become too dependent on them for programming and decided to swear off them, unless they are the last resort. And I believe that I'm at a point in my life where I should learn how to code properly first without any assistance from an LLM and strengthen my ability to learn on the go for coding projects as that is what I've seen Software Engineers in industry do. Then, I can decide how I can best utilize LLMs as a tool to benefit my learning and productivity in the future.

How to Run the Code and More:

For this project, the intended way of running the code is to first run the UDP server, then immediately run the UDP client. The UDP client will then prompt the user to enter an integer from 25 to 200. Then, we let the UDP server and UDP client do its thing and then they'll both

print out the statements that they need to print out in their respective terminals. Do note that I did not implement any safety checks for the user input and so any integers that are not in the 25 to 200 range or if the user input is not an integer itself, the program will just take the input as it is, leading to undefined behavior. Other than that, running the program is very simple.

The expected output for the UDP server is just one print statement describing the amount of bytes of data received, the timestamp of when the UDP server received the data as well as the addresses for both the UDP server and UDP client. The same print statement is also printed into the terminal for the UDP client, but this time it's data sent and when the data first started transmission. Additionally, the UDP client will print the throughput in kilobytes in a new line after the first print statement.

Run the udp_server[Tyson-Tien Nguyen]_[921031514].py first and then the udp_client[Tyson-Tien Nguyen]_[921031514].py second.

The report's name is proj1_1_[Tyson-Tien Nguyen]_[921031514].pdf.