Ina Patrice Gonzales and Ryan Monaghan

Dr. Mao

CISC 4615: R01

4 February 2020

Lab 1 Write Up

WHAT WE LEARNED:

In this lab, we learned about socket programming and how to make a server and client send messages back and forth to each other through Transmission Control Protocol (TCP). TCP is comparable to another form of Internet Protocol (IP) traffic called User Datagram Protocol (UDP). We also learned about the workflow of a server and how it differs from the workflow of a client. Further, in order to make the server and client communicate back and forth, we learned that one must run the programs in a specific order. That being said, the server must always be listening and has to wait for the client to connect before sending and receiving data. Likewise, to establish a specific connection we found that one must have a specific port number. Moreover, one must use the dot operator to access the member function “close()” to terminate an existing connection. In addition to this, there are certain methods such as “bytes.decode()” and “bytes.encode()” that are needed to send as well as receive messages between the client and server.

As C++ programmers, we found that working in the Python programming language was easy to adapt to. However, learning a new concept like socket programming was a challenge for us because of the language change. While the theory of socket programming is rather straightforward, its implementation was rather difficult. In addition to this, we found that keeping

track of server and client messages was confusing at times. In Part 2 specifically, we had to utilize a list data structure to print out all of the messages sent between the client and server. This was especially difficult in that we were not familiar with its implementation.

HOW TO COMPLETE THE LAB:

To complete this lab we had to create one file for the client and one file for the server. To begin the task of sending and receiving messages between the two files you must follow a specific order of operations. You must start by creating a socket in the server and client files. The server then binds with the “bind()” function so that it can be associated with a specific network and port number. The server will now listen for the client to approach it and make a connection. The client will have to know the specific network and port number that the server is binded to. Once the server accepts the incoming connection from the client, a connection is made and information can now be sent or received by the two. The “send()” function allows the client or server to send data on the socket with the descriptor socket. The “recv()” function allows for the client or server to receive data with the descriptor socket and can only apply to connected sockets. Once all of the messages that we were assigned to make the server and client speak to each other were finished, we had to close the connection between them by using the “close()” function.

HOW TO TEST YOUR CODE (with Screenshots):

Moreover, in order to test our code for part 1 the user must first run the server file. In doing so, the server will listen for an incoming connection by the client to continue to the next step of listening, accepting the connection, and then sending or receiving information. Next, the client will run to send a message to the server. To explain further, we hard-coded the messages

for both the server side and the client side to get those messages to send back and forth to each other. That being said, if a message was trying to reach the server side from the client side, we need the bytes to first be encoded before sending on the client side, and decode them on the server side to get the message. This process is the same if we are trying to send a message from the server side to the client side. Furthemore, the client and the server both share a port number, and are connected through the server name and port number, or in this case 'local host' and 12000. In doing so, we can send encoded messages as bytes back and forth to be decoded on either side, until we close the socket connection. Thus, for part 1 the Server receives the encoded messages from the client, and then decodes them, and we get our first message: "Hello Fordham," and then our next message received goes through the same process, and once the message is decoded we see "Bye" in our output. Likewise, for the client side we receive an encoded message from the server and decode it to get the output "Bye." Then, once both sides are done sending and receiving we will close the connection, and the message "Connection Closed" will be printed to the screen.

Output for Part 1:

That being said, for part 2, we have the existing messages sent back and forth between the server and client. However, now we will store the messages on the client side. In doing so, once we stored all the messages and put them into the client side, we will print out the message

list after closing the connection between server and client, respectively. Thus, our output outputs every message to the client side in order from which a message was sent and received.

Output for Part 2

```
Inas-MacBook-Pro-2:lab1 inagonzales$ python3 server-2.py
The server is ready to receive
Connection Closed
```

```
Inas-MacBook-Pro-2:lab1 inagonzales$ python3 client-2.py
Connection Closed
This is the chat from the server and the client:
Client: Hello Fordham
Server: Hello CIS Students
Client: Hello CISC4615
Server: Bye
Inas-MacBook-Pro-2:lab1 inagonzales$
```

CONCLUSION:

In conclusion, socket programming had a difficult learning curve, however, we were able to overcome its complexities. Overall, it was interesting to see how a server and client can communicate back and forth. This lab allowed us to practice our Python programming skills in a new field of study which will diversify our resume.