Advanced Network Systems and Security

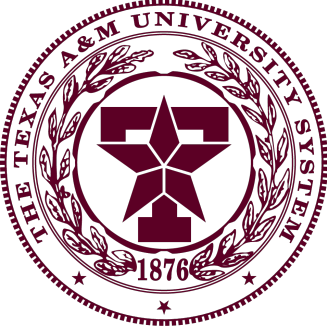
Lab Report 1

Socket Programming

Krish Sareen - Fernando Velez

330000230 - 527008214

January 31, 2023



**1 Introduction**

This lab was split into four different tasks. In the first task (task 0), we are simply downloading all the files we will be needing, including Python, Wireshark, and a zip file with two Python files that will be needed later in the lab. This step was important to do first, because we will be disconnecting from the internet in the next tasks.

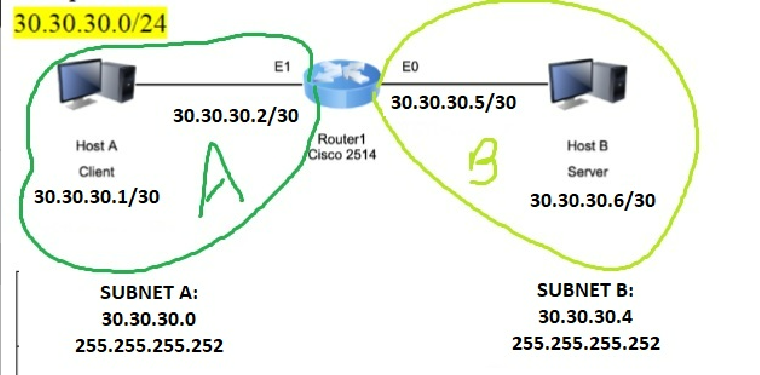
In the next task (task 1), we are setting up the network for the host to communicate with each other. First we simply identify the different subnets and proceed to assign IP addresses to each device or router interface within the network. Next we use the instructions provided on canvas to properly connect a laptop to the router’s console port using Putty. Once everything was properly connected we could begin configuring the router. This is done by using basic configuration commands learned in ESET 315, such as “enable”, “config terminal”, “int e0”, “ip address X.X.X.X 255.255.255.X”, “no shutdown”, and “end”. Once done configuring, we can use “sh run” to check if our configurations were saved how we wanted them. After that, we change the IP address on our host devices, connect them to the router, and attempt to ping a host in a different subnet.

Next, in the third task (task 2) we begin the socket programming. Here, we decide who will be the server and who will be the client. Then, each person opens their related Python code (e.g. the laptop used as the server will open the “tcpserver\_lab1.py” file). These files are then modified to have the appropriate IP addresses we chose for our network, and we make sure that both the files are using the same port number as well. Now the server can begin running their program. Doing so will constantly run the server, allowing a client to send a request to the server. When the request is sent, the 3 way handshake is initiated to synchronize the client and the server. After the 3 way handshake, the client can now send their packet to the server which in our case is a message that we typed. The server will then receive the packet, allowing the person who is the server to see the message typed by the client. Our code then allows the server to send a message back to the client in the same manner. The client then receives the response from the server and closes their connection to the server. The code for the client stops running, ending the conversation. If the client wants to send another message, the program has to be run again, and a new connection will be established repeating the earlier process. The server on the other hand will keep running until it is manually stopped. For the final task (task 3), we modify the previous code to make the server and client have a continuous, infinite conversation until the client says “bye”, instead of only being a 2-message application.

**2 Results**

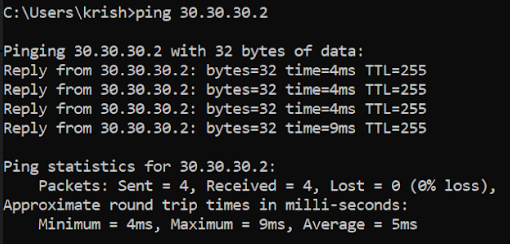
**2.1 Question 1** - Network and IP Addresses

The network in Figure 1 was constructed using principles of variable length subnet masking. Because each subnet needed only 2 user IP addresses, masks of /30 were used, allowing for 2 host bits and 30 mask bits, meaning 2^2 = 4 addresses - 2 = 2 users per subnet. The first subnet was 30.30.30.0, with the first user being 30.30.30.1 and the last being 30.30.30.2. The second subnet 30.30.30.4 begins with address 30.30.30.5 and ends with 30.30.30.6. The network was configured on a Cisco 2514 Router and 2 host computers.

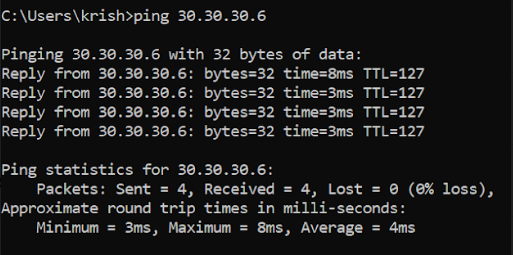


**Figure 1.** Subnet diagram of network 30.30.30.0/24

The ping commands to test the connectivity of the network are shown in Figure 2 and 3. Each device in the network was able to successfully ping every other, at every interface available.



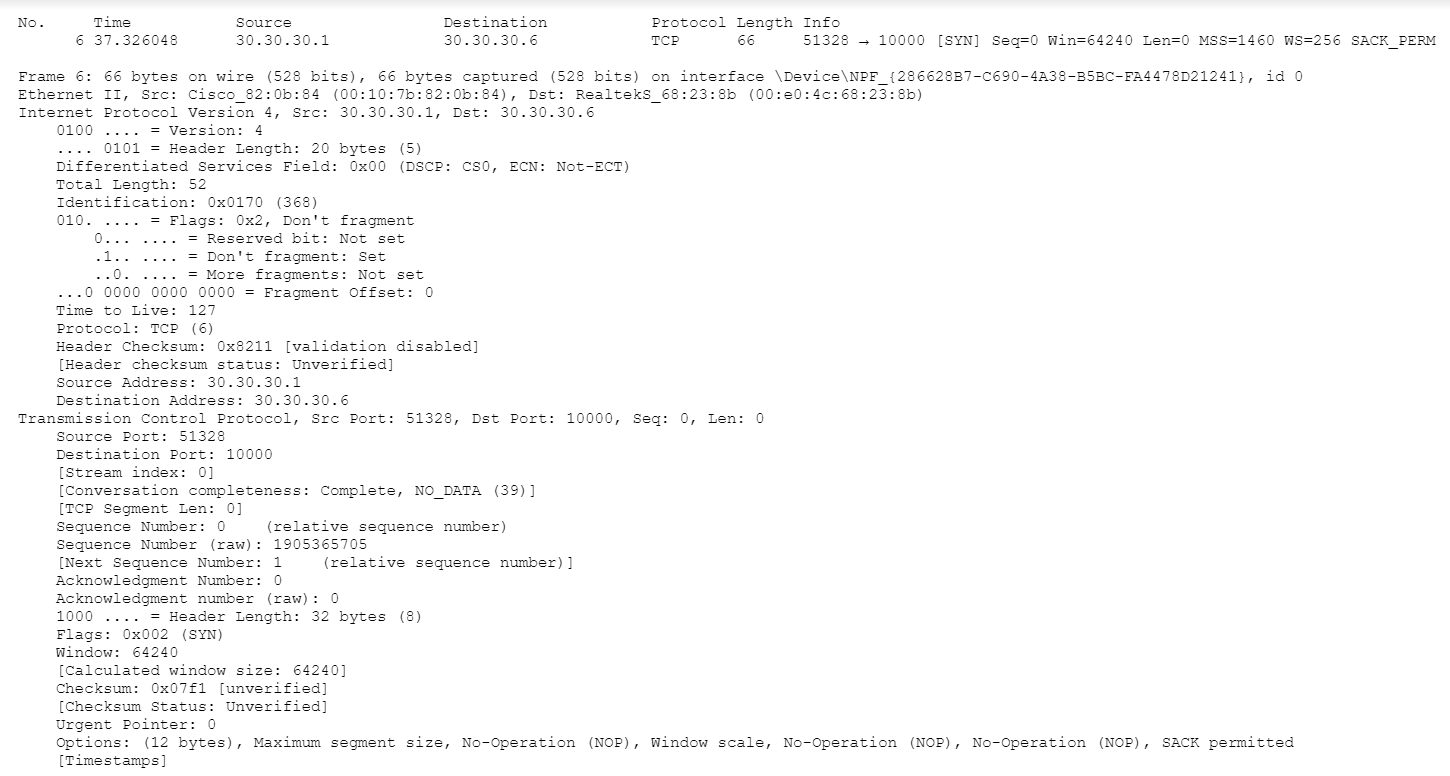
**Figure 2.** Ping from 30.30.30.1 (Host A) to 30.30.30.2 (Router E1)



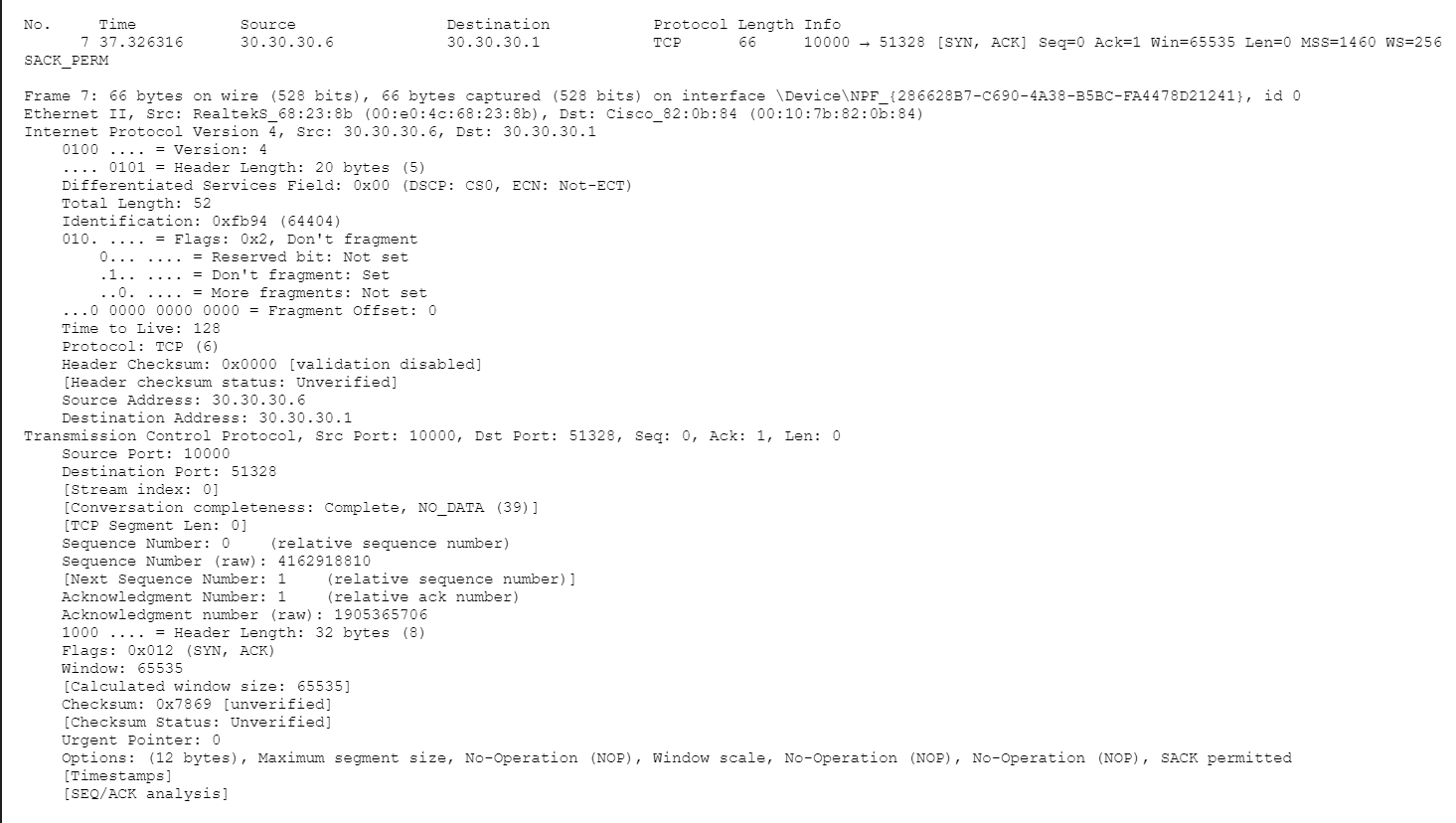
**Figure 3.** Ping from 30.30.30.1 (Host A) to 30.30.30.6 (Host B)

**2.2 Question 2 -** Wireshark capture

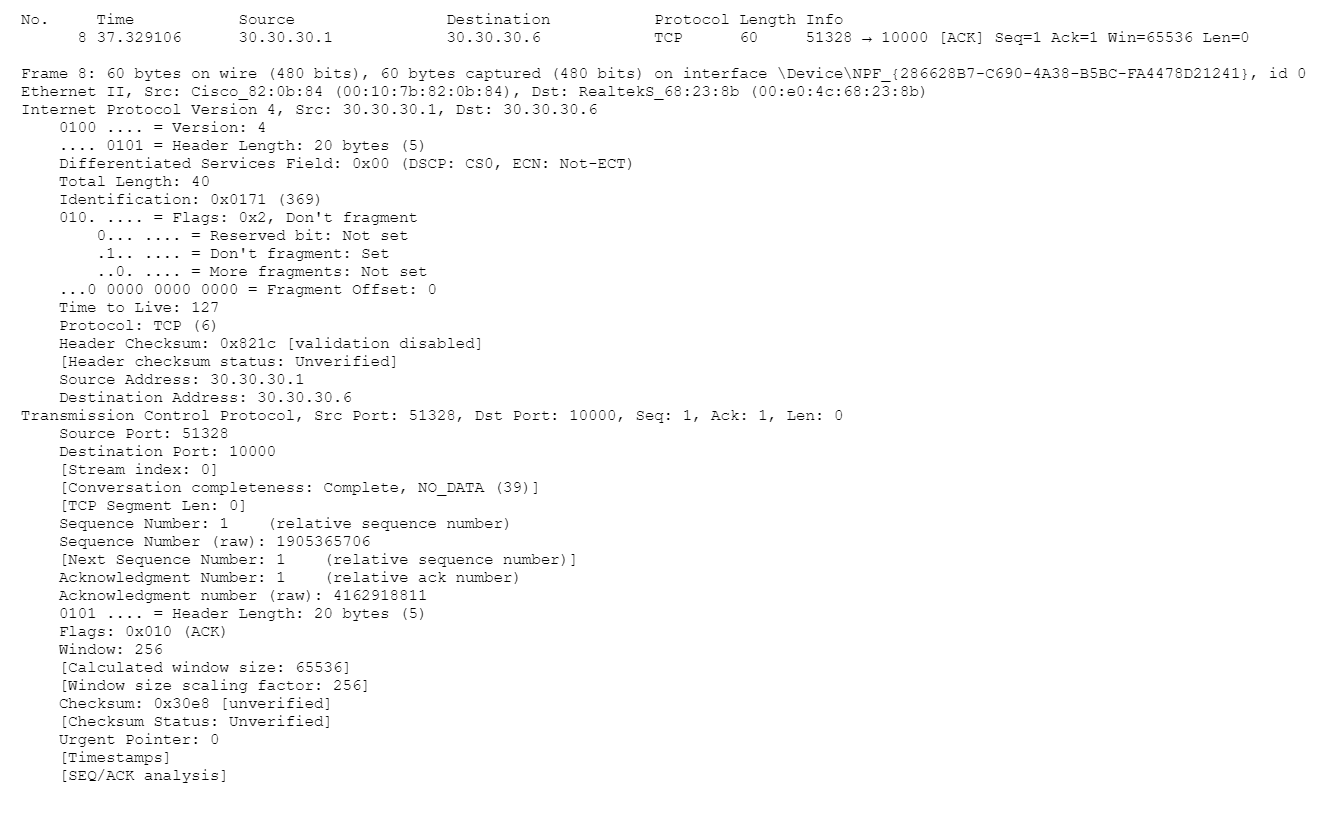
After configuring the network, the hosts set up a TCP/IP handshake, which was initiated by Host A (the client). Figures 4, 5, and 6 show these packets and the basic connection messages that established the socket connection.



**Figure 4:** First handshake packet

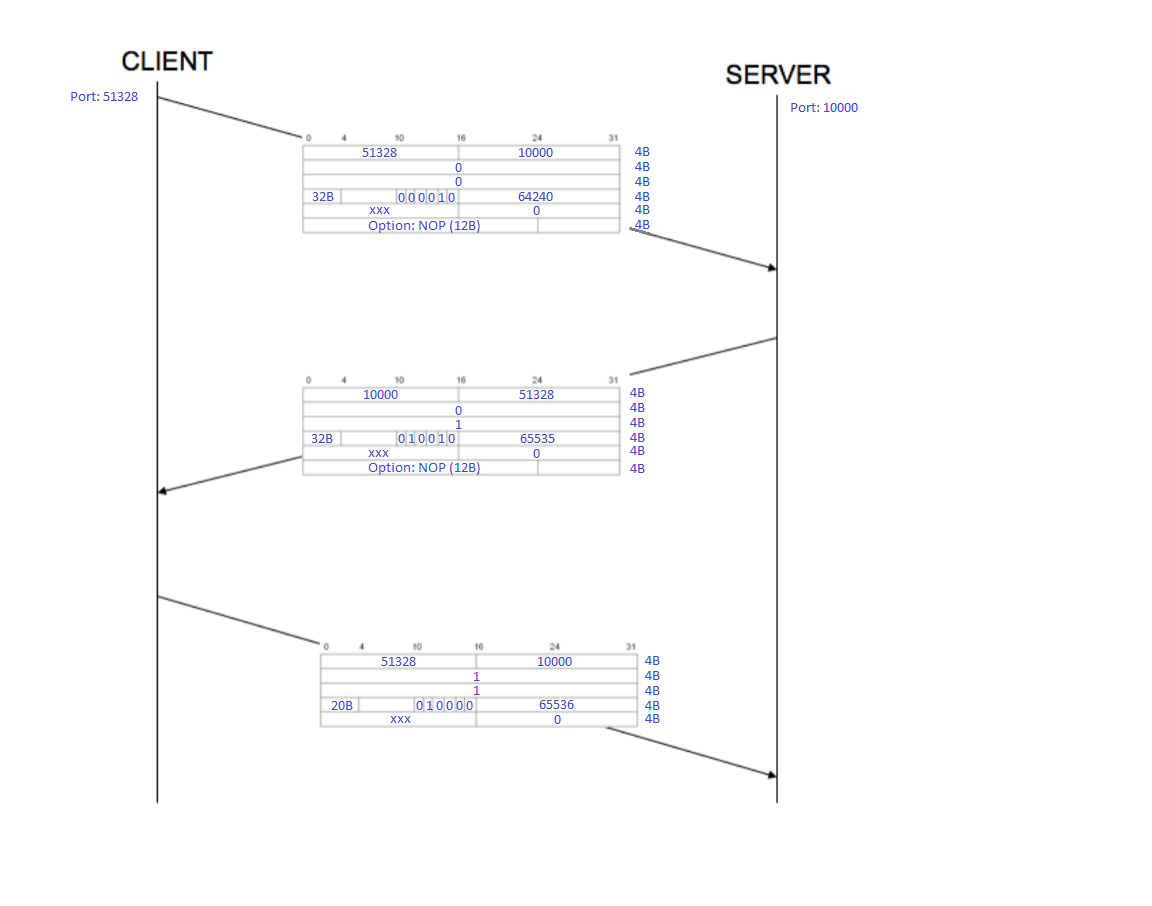


**Figure 5:** Second handshake packet



**Figure 6:** Final handshake packet

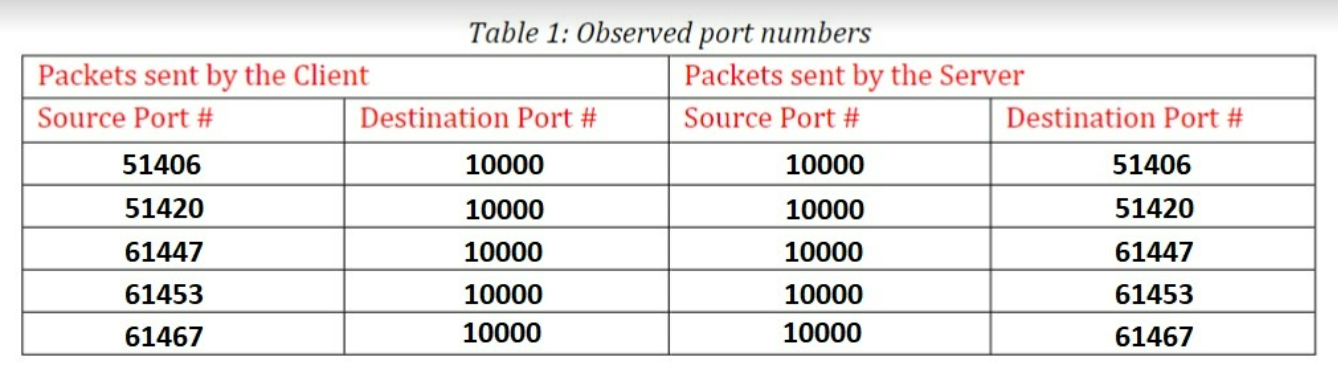
In Figure 7 below, the diagram for the 3 way handshake can be seen.



**Figure 7:** Handshake Diagram

**2.3 Question 3 -** Running the Chat Program

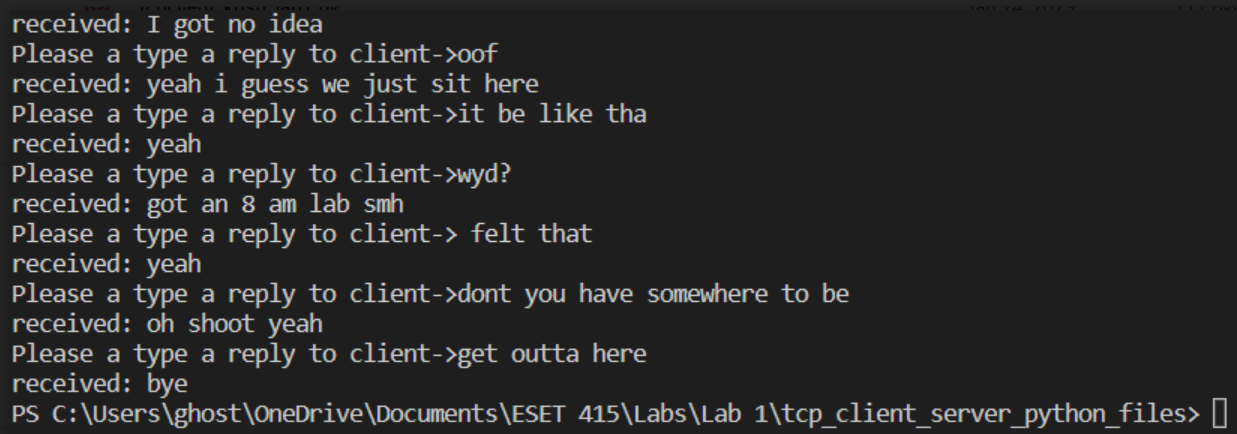
**Table 1:** TCP Ports for5 packets between Host and Server



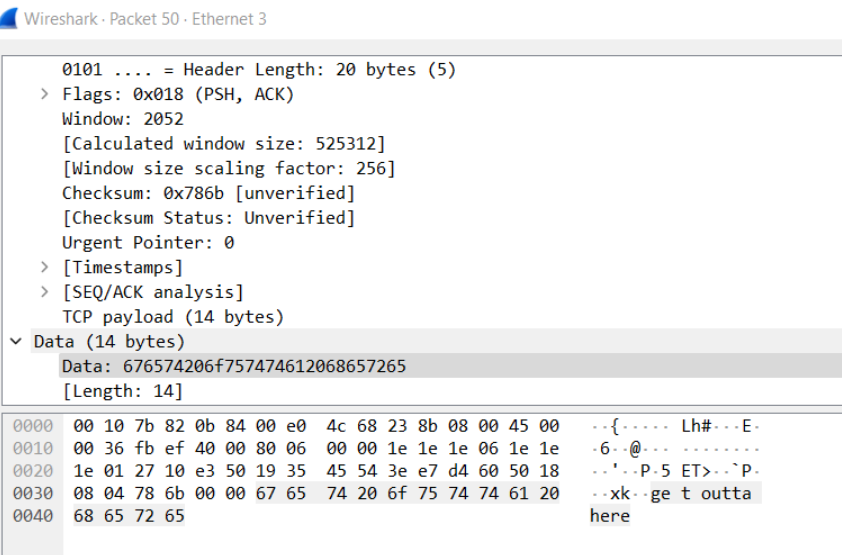
In this table, because of multiple separate connections, client ports keep changing. However, the server port remains the same throughout all of the packets. Once the client begins the handshake, the port number used remains the same until the connection is over, and after that happens, a new port is used. In the server, it is specified which port to use, as that is used to create the original packet to start the handshake. This makes it important for the server to keep the same ports for service throughout use, especially before connections are made. As for the pc, it is not important which port is used, as long as it stays the same throughout a connection. There is likely some reason on the operating system level that results in a new port number every time. One thing is definitely clear: the port numbers increased by value in each message, showing us that the older ports are likely used up for other functions as time goes on, although they will likely become available again after a few moments of being used.

**2.4 Question 4 -** Edited code and Message Packet

At the end of the lab, we changed the code in both the server and client files to keep the chat going until the client said the word ”bye”. This allowed for only one handshake to create a permanent connection between devices until one disconnected. During our text conversation, we captured one of the messages on wireshark and saw the unencrypted message. When a single message is sent, it becomes one or multiple packets that send from Host to Router and then to final destination. Figure 8 shows a message and its unencrypted contents.



**Figure 8:** Terminal conversation from perspective of the Server



**Figure 9:** Terminal conversation from perspective of the Server

This captured packet is part of the earlier conversation in Figure 9. The message “get outta here” is clear and visible in the data section of the packet, which is extremely interesting. This network was established through cables, but many networks can be configured wirelessly, which would allow for any computer to use wireshark and look at the packets. In this case, the message is unsecure, so any computer could see it. This highlights the importance of end-to-end security in networking.

**3 Conclusions**

Through this lab, we were able to get a quick refresher on how to build simple networks (connecting the devices and configuring the router). This is crucial as some students took ESET 315 multiple semesters ago, and they may have needed to be reminded of the technologies and skills learned in that course. We were also able to observe how a client and a server interact with each other on the tcp layer. Such as what ports are used or changed when interacting multiple times, how the data sent appears as packets, and how the 3 way handshake appears on the packet level. Packets were tracked using Wireshark. During our lab, we only really ran into one issue where our laptops couldn’t ping each other in the network. We later realized that we had simply connected them to the router incorrectly and actually switched them. Troubleshooting this problem and some initial issues with subnet addressing helped us regain confidence and speed when building our network. In the next lab, we will carry these skills and improve them even more going forward. This is also the first time we ran applications on top of layer 4 network architecture, making

**A Lab Journal**

A.1 Python

List builtin functions that were used and describe them. This is an opportunity for you to keep track of things, so you can look back at them later for reference.

* chr(*int number*) - returns the character corresponding to the value in the ASCII table.
* ord(*char*) - returns the integer corresponding to the character’s position on the ASCII table

A.2 Putty (configurations below are done using int e0 as an example)

* enable (en) - enter privileged EXEC mode
* config terminal (config t) - enter configuration mode
* For Cisco 2514: interface e0 (int e0) - specify the interface (in this case Ethernet0) for the Interface level
* For Cisco 4221: interface g0/0/0 (int g0/0/0) - specify the interface (in this case Ethernet0) for the Interface level
* ip address 10.10.10.1 255.255.255.0 - assign current interface (in this case Ethernet0) with an IP address and subnet mask
* no shutdown - enable the interface to send and receive packets
* end - get out of Privileged EXEC mode
* show running-config (sh run) - check the current running configuration, this can be used to check if your configurations were saved and accepted by the router.