CPSC 3600

Spring 2021

Final Exam

Anthony Jones

Q1 **We talked about the OSI Seven Layer Network model, and we identified which of these layers are considered to be a part of TCP/IP. Identify which of the following are layers from the OSI Seven Layer model that are considered to be the layers that form TCP/IP:**

Well, the four layers of the TCP/IP model are:

Application

Transport

Network (also called Internet)

Network Interface

Therefore, the layers provided from the OSI Seven Layer model to also be considered layers of the TCP/IP are only:

Application

Transport

The others listed are protocols or other networking concepts.

Q2 **We talked about naming and addressing. Circle the statements that are true:**

A Host can be identified by a domain name (assuming the name is correctly registered in the DNS system).

From a School of Computing Linux machine such as joey8.computing.clemson.edu, if we issue a ‘ping www.ibm.com’, we engage the resolver (the DNS client) on joey8 which interacts with one or more router servers to resolve the name www.ibm.com to a valid IP address.

A Host can be identified by its IPv4 address in dotted quad format such as 192.168.1.2.

A Host can be identified by localhost but only by a user or program on a Host that is operating on the same network.

An IP v4 IP address contains two pieces of information: the network id that the host is connected with and the Host ID that uniquely identities that Host on that Network.

When we say a Class A network is larger than a Class C network, we are referring to the maximum size of each network in terms of if the network is based on long distance fiber links or based on Ethernet Local Area Network.

Does the following Host exist on Clemson’s network? o 2620:103:a000:401:f8cb:5613:72ff:fa70

Q3 **Let us say the application message is 1000 bytes, this includes a 4 octet seq number. UDP adds 8 bytes of protocol overhead, IP adds 20 byte, and a MAC frame adds 28 byte. The message is sent from H1 to H2 over a direct link with an effective data rate of 10Mbps. The one way propagation delay is 10 ms. No other delays come into play and packet loss does not occur. You are to find the time it takes to send one message from H1 until the message is received by H2. Circle the best answer.**

Effective data rate in bytes / second:

10Mbps \* (125000 byte / 1Mb) = 1.25e+6 bytes / second

Total packet size:

1000 bytes + 8 bytes + 20 bytes + 28 bytes = 1056 bytes

One-way trip time:

(total packet size) / (effective data rate) =

(1056 bytes) / (1.25e+6 bytes / second) = 0.0008448 seconds

One-way trip time in milliseconds:

0.0008448 seconds \* (1000 ms / 1 second) = 0.8448 ms

Add on propagation delay:

10 ms + 0.8448 ms =

10.845ms

Q4 **Let us say the application message is 2,960 bytes inclusive of the application sequence number. The protocol overhead stated in Q3 applies. Assume the maximum transmission unit (MTU) of the outbound network interface is 1500 bytes.**

4a. **What is the size of the IP datagram before it gets fragmented?**

The UDP datagram would be 2960 + 20 + 8 + 28 = 3016 bytes before fragmented.

4b. **Roughly what is the time difference between when the message is received at H2 from when H1 transmits the first bit? Please show your work.**

First IP Packet:

IP header: 20 bytes

UDP header: 8 bytes

MAC frame: 28 bytes

Data (first fragment): 1500 – (20 + 8 + 28) = 1444 bytes

Second IP Packet:

IP header: 20 bytes

MAC frame: 28 bytes

Data (second fragment): 1500 – (20 + 28) = 1452 bytes

Third IP packet:

IP header: 20 bytes

MAC frame: 28 bytes

Data (what’s left): 2960 – (1444 + 1452) = 64 bytes

Total size of three fragments: 1500 + 1500 + 112 = 3112 bytes

*\*\*\*I could not determine if MAC frames were to be added on each fragment, I used* [*https://inet.omnetpp.org/docs/showcases/wireless/fragmentation/doc/index.html*](https://inet.omnetpp.org/docs/showcases/wireless/fragmentation/doc/index.html) *as my reference and decided so\*\*\**

Assuming the same effective data rate of 1.25e+6 bytes / second:

One-way trip time:

(total fragments size) / (effective data rate) =

(3112 bytes) / (1.25e+6 bytes / second) = 0.0024896 seconds

One-way trip time in milliseconds:

0.0024896 seconds \* (1000 ms / 1 second) = 2.4896 ms

Add on propagation delay:

10 ms + 2.4896 ms =

12.490ms

Q5 **Which of the following are true statements (circle all the true statements):**

**1. struct sockaddr\_storage clntAddr; // Client address**

**2. struct sockaddr\_in \*clntAddrPtr = (struct sockaddr\_in \*) &clntAddr;**

**3.**

**4. // Set Length of client address structure (in-out parameter)**

**5. socklen\_t clntAddrLen = sizeof(clntAddr);**

**6. ….additional code**

**7. ….enter while loop …**

**8. recvMsgSize = recvfrom(sock, RxBuffer, MAX\_DATA\_BUFFER, 0,**

**9. (struct sockaddr \*)clntAddrPtr, &clntAddrLen);**

**10.**

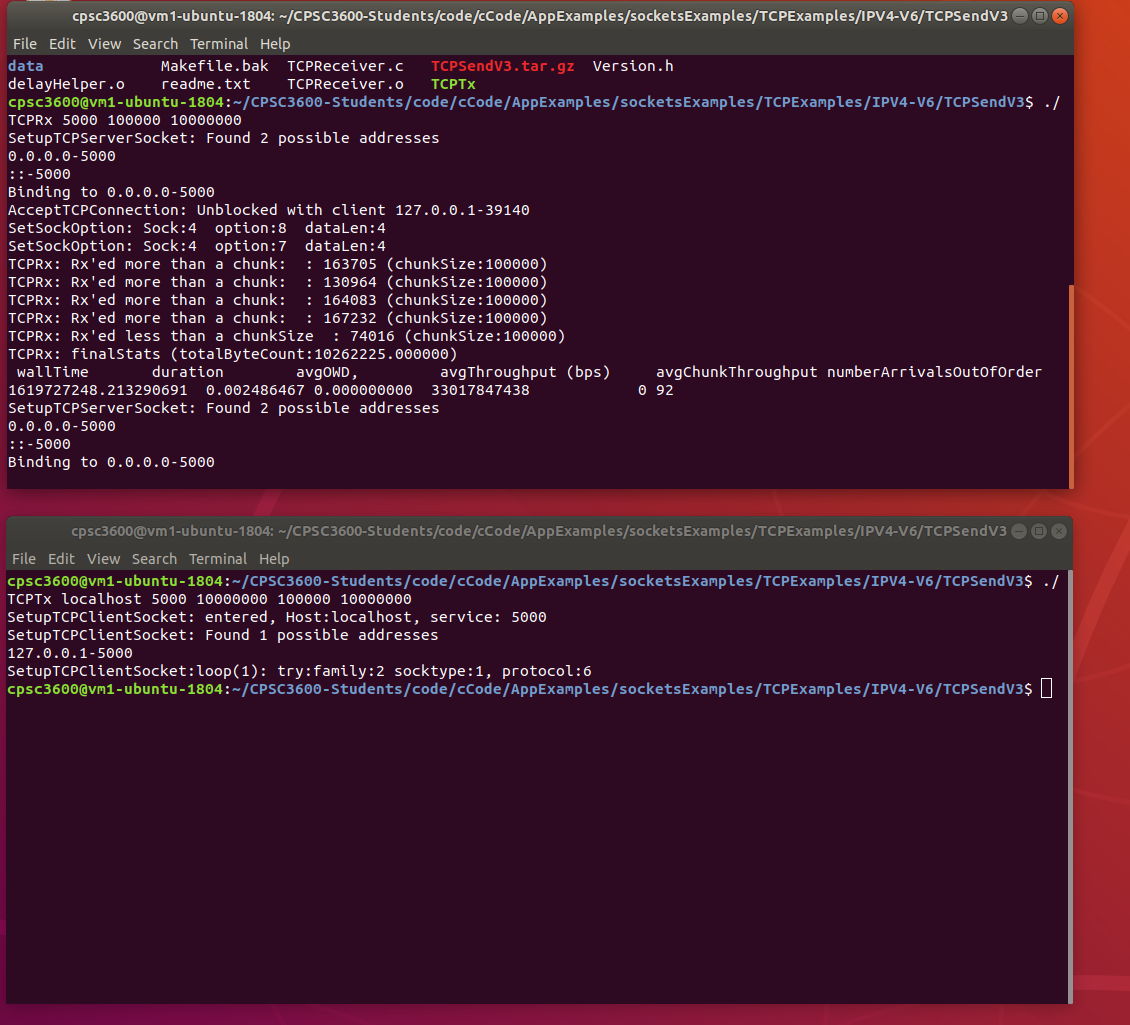
The purpose of Line 1 is to allocate memory needed so the server can specify the client’s IP address.

In line 8, the server learns the client’s IP address

The way this is coded prevents an IPV6 client from being able to connect and interact with the server.

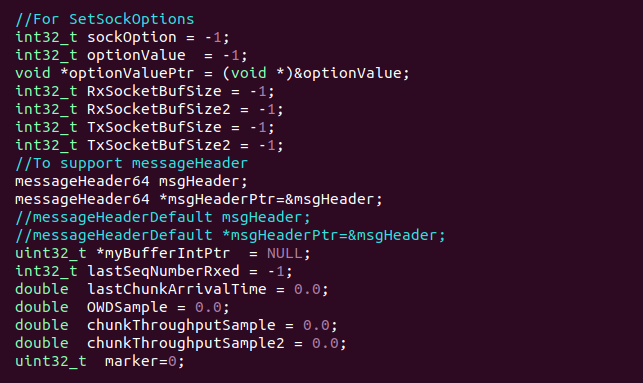
Q6 **Refer to the code code/cCode/AppExamples/socketsExamples/TCPExamples/IPV4- V6/TCPSendV3:**

6a. **Build and run the client and server on your VM. Have the TCPTx program transfer 10Mbytes. Cut and paste the observed throughput results. You can run the client and server on locally on your VM.**

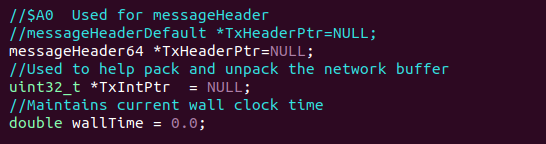


6b. **Modify the client and server so that it uses messageHeader64 instead of messageHeader . This requires changes in the client and the server of the program. The struct for messageHeader64 is located in commonCode/messages.h**

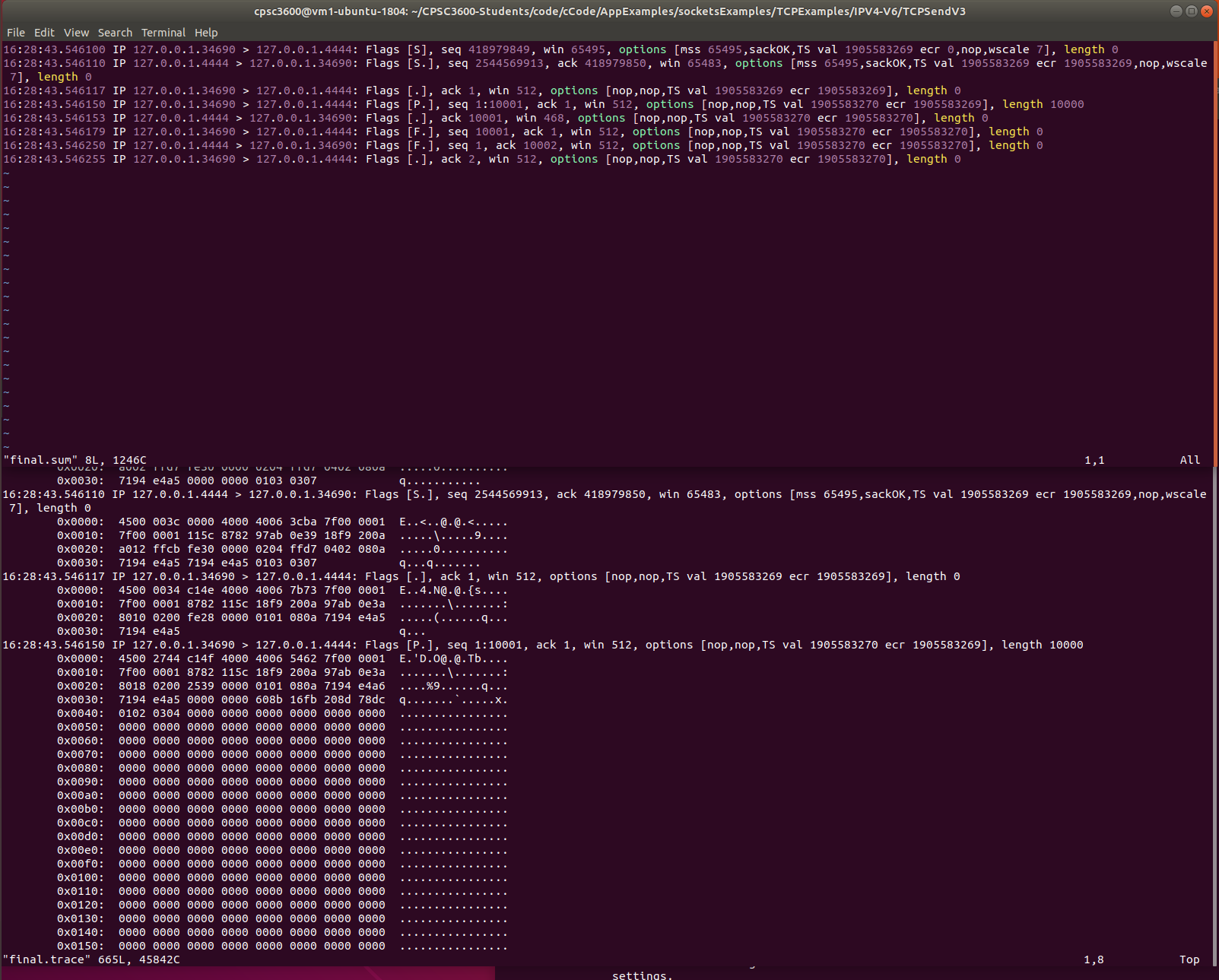
In Receiver:



In Sender:



6c. **With your modified code from Q6b, run the client (TCPTx) specifying 10000 bytes to transfer. Use 1 Mbyte for the client and server’s chunkSize and pipeSize. Using tcpdump, show evidence that your modifications to the client and server are correct. You can do this on your VM using localhost.**



As you can see the modifications worked. This was the test.sum and test.trace files that were created by tcpdump. You can see the messages being sent by the server and client (I had to use host 4444 because for whatever reason using normal host 5000 was bugging out).