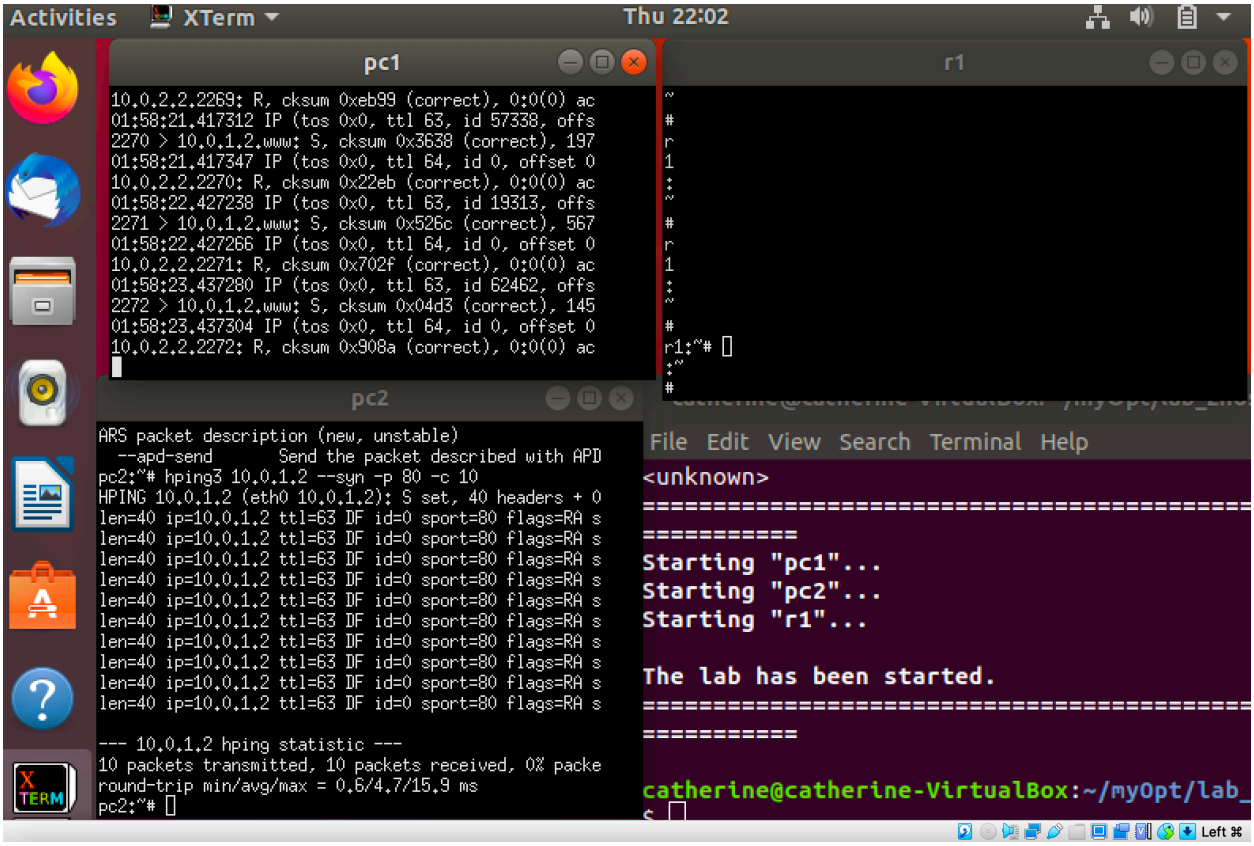
NAME: CATHERINE SMITH

CS 5700 Fundamentals of Networking

Project #2 **Assigned: 5/20/20** **Due: 6/13/20** [100 points]

PART II.



I used Ubuntu (running on a VirtualBox VM) to download and run netkit. See bottom left corner (pc2) for evidence of hping running successfully.

PART III.

3. Five connections are established and closed.

The first four connections end the same way: client and server exchange FIN+ACK segments, and then the server acknowledges the end of the exchange:

```
10.0.1.2   TCP    … 11111 → 34720 [FIN, ACK] Seq=1 Ack=18 Win=5792 Le…
10.0.2.2   TCP    … 34720 → 11111 [FIN, ACK] Seq=18 Ack=2 Win=5840 Le…
10.0.1.2   TCP    … 11111 → 34720 [ACK] Seq=2 Ack=19 Win=5792 Len=0 T…
```

The final connection ends a little differently. The client initiates the end of the exchange using a FIN+ACK segment and then the server sends an acknowledgement:

```
10.0.2.2   TCP    … 34724 → 11111 [FIN, ACK] Seq=18 Ack=1 Win=5840 Le…
10.0.1.2   TCP    … 11111 → 34724 [ACK] Seq=1 Ack=19 Win=5792 Len=0 T…
```

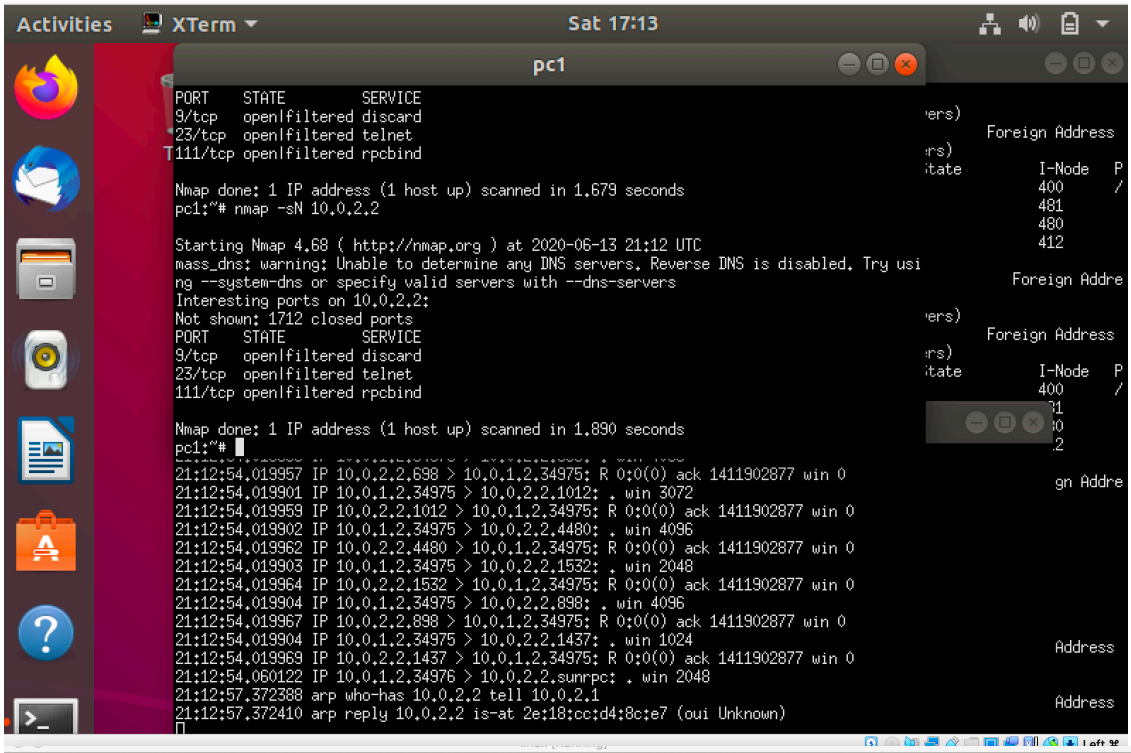
The initial sequence numbers are not increasing every four microseconds. I switched from the "relative sequence" view to examine the original sequence numbers in comparison with the time stamp, and the starting sequence number for each subsequent connection increases by more than you'd expect if you were only increasing by one number every four microseconds.

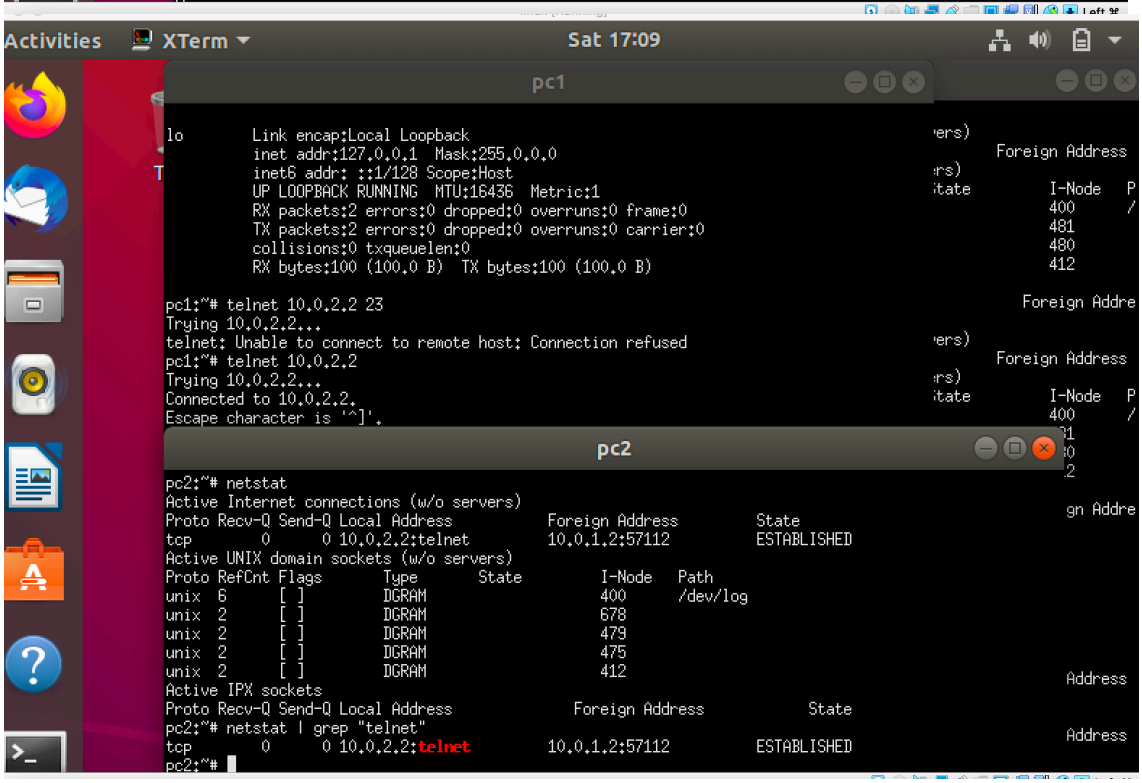
4. The attempted connection cannot be established. Perhaps the intended port is closed or the intended receiving host is not listening for TCP connections. Regardless, the host sends repeated [RST, ACK] packets to indicate that the connection isn't established and the socket should be closed.

5. Max segment size is a standard option indicating the maximum segment size that a host can handle. Window scale option allows increases to the transmission window, which can help support performance if end-to-end delay is expected to be high. The SACK permitted options allows selective acknowledgement of packets in order to avoid causing congestion with unnecessary retransmissions.

PART IV.

13. nmap works by sending SYN segments to every port on the IP address provided. Based on what responses are received, nmap can infer which processes are running. It's a bit like knocking on every door in a neighborhood—you'll know who's home based on who answers the door. See the following screenshots of telnet and nmap running:





15.

1. MSS, SACK permitted, window scale, and timestamps.
2. It appears that host 0 did not receive acknowledgments within the expected timeframe and overwhelmed the server, leading to very poor roundtrip times that also caused congestion for host 1. The server recovered its performance and round-trip times were good for hosts 2 and 3.
3. No, it didn't actually implement delayed acknowledgements.
4. **Trace 0:** 1) Segment #142 appears to be a fast retransmission. It is retransmitted after a string of duplicate ACKs. Segment #168 is also retransmitted immediately after 3 duplicate ACKs. 2) There do no appear to be any retransmissions due to timeout. **Trace 1:** 1) #77 is a fast retransmission. 2) #156 is a retransmission. **Trace 2:** Uses SACK retransmission. 1) #64 is a fast retransmission. 2) #238 is a retransmission (not fast). 1) **Trace 3:** Uses SACK retransmission. 1) #74 is a fast retransmission. It comes directly after three duplicate ACKs. 2) #12 appears to be a retransmission. Transmission stops completely and seemingly reverts to slow start, so it would not count as a fast retransmission.
5. The round-trip-time graph could be computed by taking the difference between a packet's transmission and the acknowledgment. The time-sequence graph can be plotted by comparing sequence numbers with timestamps.
6. There are a number of ways you could implement these analysis heuristics. For instance, you could keep a table with segments as keys and create flags—initially set to false—for retransmission, ACK and duplicate ACK, and timeout.
7. A file about TCP protocol from the DARPA days.

15. For the exercises below, we have performed measurements in an emulated31network similar to the one shown below.

Figure 4.69: Emulated network

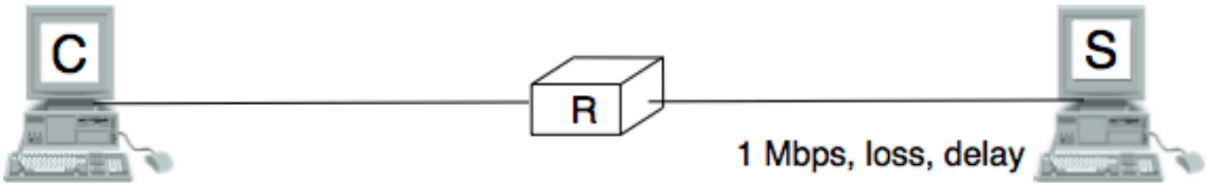


Figure 4.69: Emulated network

The emulated network is composed of three UML machines32: a client, a server and a router. The client and the server are connected via the router. The client sends data to the server. The link between the router and the client is controlled by using the netem Linux kernel module. This module allows us to insert additional delays, reduce the link bandwidth and insert random packet losses.31With an emulated network, it is more difficult to obtain quantitative results than with a real network since all the emulated machines need to share the same CPU and memory. This creates interactions between the different emulated machines that do not happen in the real world. However, since the objective of this exercise is only to allow the students to understand the behaviour of the TCP congestion control mechanism, this is not a severe problem.32FormoreinformationabouttheTCPcongestioncontrolschemesimplementedintheLinuxkernel,seehttp://linuxgazette.net/135/pfeiffer.html andhttp://www.cs.helsinki.fi/research/iwtcp/papers/linuxtcp.pdfor the source code of a recentLinux. A description of some of the sysctl variables that allow to tune the TCP implementation in the Linux kernel may be found inhttp://fasterdata.es.net/TCP-tuning/linux.html. For this exercise, we have configured the Linux kernel to use the NewReno schemeRFC 3782that is very close to the official standard defined inRFC 5681

We used netem to collect several traces :
•exercises/traces/trace0.pcap
•exercises/traces/trace1.pcap
•exercises/traces/trace2.pcap
•exercises/traces/trace3.pcap
Using wireshark or tcpdump, carry out the following analyses :

1. Identify the TCP options that have been used on the TCP connection
2. Try to find explanations for the evolution of the round-trip-time on each of these TCPconnections. For this, you can use theround-trip-timegraph of wireshark, but becareful with their estimation as some versions of wireshark are buggy

3. Verify whether the TCP implementation used implemented delayed acknowledgements
4. Inside each packet trace, find :
    1. one segment that has been retransmitted by using fast retransmit. Explain this retransmission in details.
    2. one segment that has been retransmitted thanks to the expiration of TCP's retransmission timeout. Explain why this segment could not have been retransmitted by using fast retransmit.
5. wiresharkcontains several two useful graphs : the round-trip-time graph and the time sequence graph. Explain how you would compute the same graph from such a trace.
6. When displaying TCP segments, recent versions of wireshark contain expert analysis heuristics that indicate whether the segment has been retransmitted, whether it is a duplicate ack or whether the retransmission timeout has expired. Explain how you would implement the same heuristics as wireshark.
7. Can you find which file has been exchanged during the transfer ?