Andre Hei Wang Law

4017 5600

COEN 3166 – FL-X

**Mininet Assignment 2 + Wireshark Assignment 3**

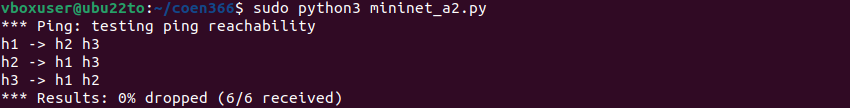
1. **Mininet Assignment 2**

1. Add a controller ‘c0’ in your topology.

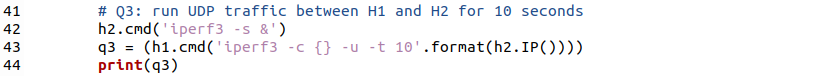


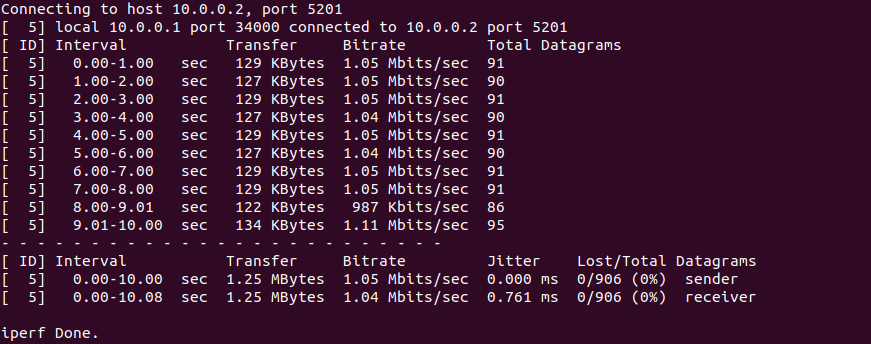
2. Test the reachability between every host using pingall (take screenshot)



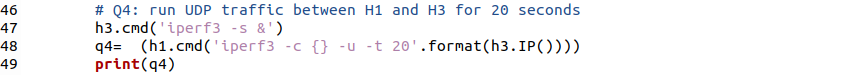


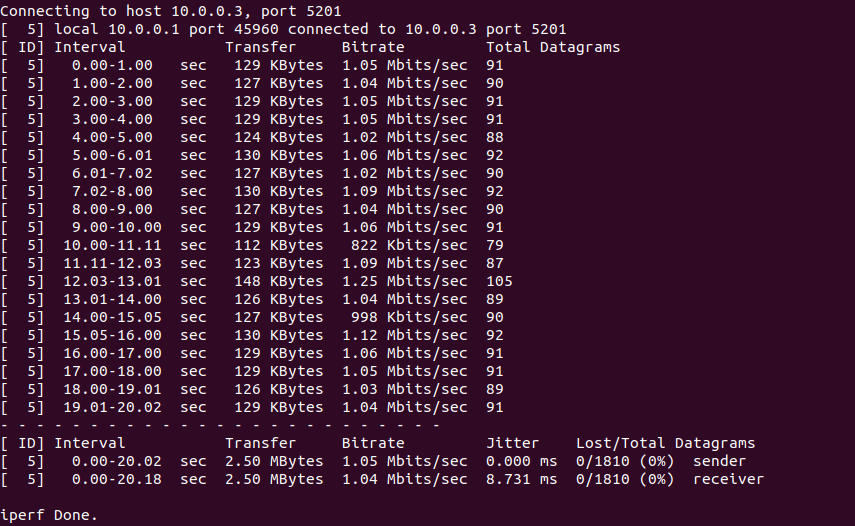
3. Run UDP traffic for 10 seconds between H1 (client) and H2 (server). Provide the iperf commands and the result (take screenshot).



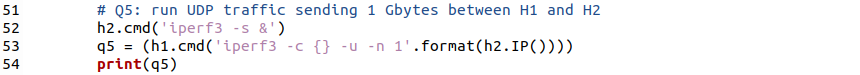


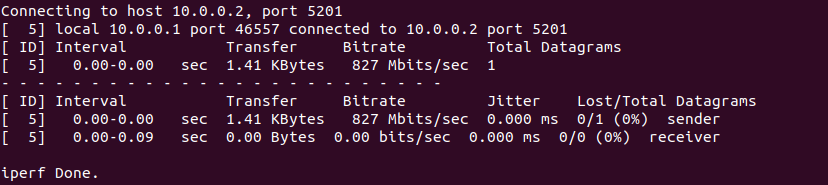
4. Run UDP traffic for 20 seconds between H1 (client) and H3 (server). Provide the iperf commands and the result (take screenshot).



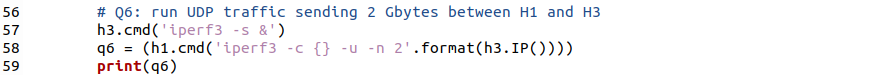


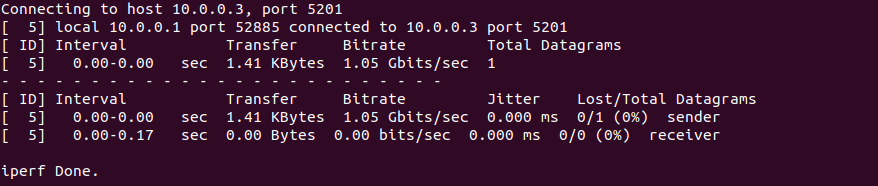
5. Run UDP traffic that sends 1 Gbytes between H1(client) and H2 (server). Provide the iperf commands and the result (take screenshot).



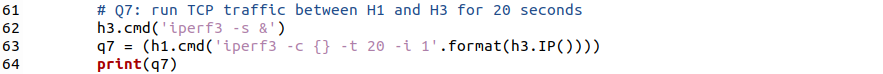


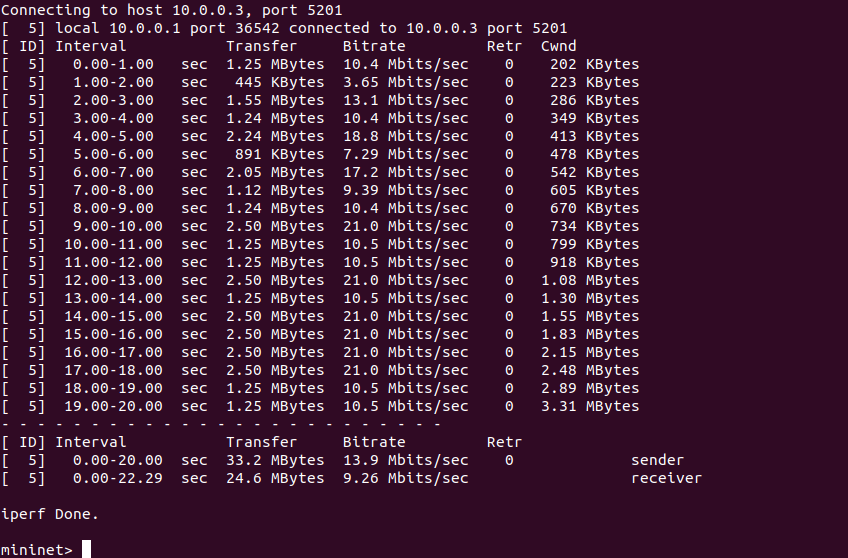
6. Run UDP traffic that sends 2 Gbytes between H1 (client) and H3 (server). Provide the iperf commands and the result (take screenshot).



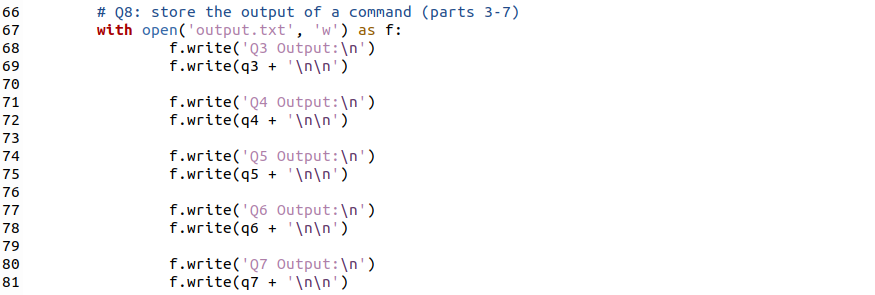


7. Run TCP traffic for 20 seconds between H1 (client) and H3(server), monitor the result on the server each 1 second. Provide the iperf commands and the result (take screenshot).





8. You need to store the output of a command (parts 3- 7) in a file (Just try it on one command) (take screenshot).



**“mininet\_a2.py” code**

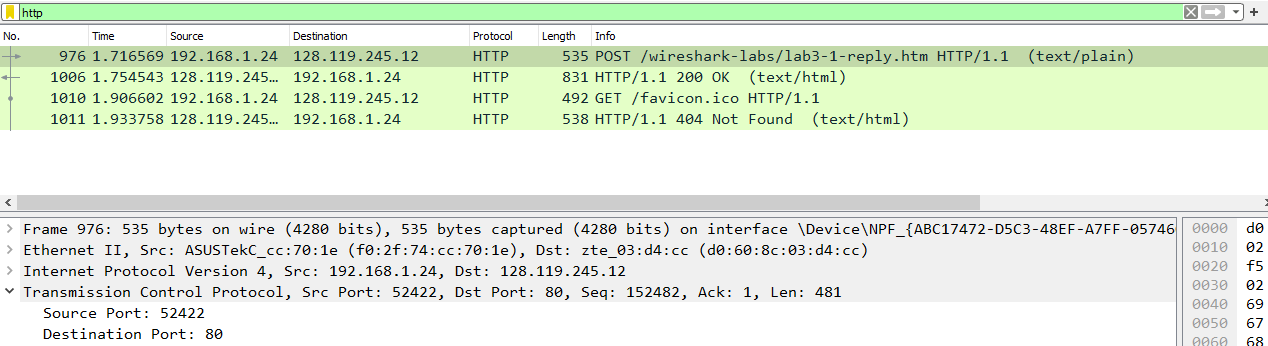
# Andre Hei Wang Law  
# 4017 5600  
# coen366 FL-X  
# mininet a2  
  
from mininet.net import Mininet  
from mininet.node import Controller  
from mininet.cli import CLI  
from mininet.link import TCLink  
  
def create\_topology():  
 net = Mininet(controller=Controller, link=TCLink)  
  
 # Q1: add a controller  
 c0 = net.addController('c0')  
  
 # add hosts  
 h1 = net.addHost('h1')  
 h2 = net.addHost('h2')  
 h3 = net.addHost('h3')  
  
 # add switches  
 s1 = net.addSwitch('s1')  
 s2 = net.addSwitch('s2')  
  
 # create links   
 net.addLink(h1, s1, bw=20, delay='10ms')  
 net.addLink(h2, s1, bw=25, delay='10ms')  
 net.addLink(s1, s2, bw=11, delay='40ms')  
 net.addLink(s2, h3, bw=15, delay='7ms')  
  
 # build  
 net.build()  
 c0.start()  
 s1.start([c0])  
 s2.start([c0])  
  
 # Q2: test the reachability between every host  
 net.pingAll()  
  
 # Q3: run UDP traffic between H1 and H2 for 10 seconds  
 h2.cmd('iperf3 -s &')  
 q3 = (h1.cmd('iperf3 -c {} -u -t 10'.format(h2.IP())))  
 print(q3)  
  
 # Q4: run UDP traffic between H1 and H3 for 20 seconds  
 h3.cmd('iperf3 -s &')  
 q4= (h1.cmd('iperf3 -c {} -u -t 20'.format(h3.IP())))  
 print(q4)  
  
 # Q5: run UDP traffic sending 1 Gbytes between H1 and H2  
 h2.cmd('iperf3 -s &')  
 q5 = (h1.cmd('iperf3 -c {} -u -n 1'.format(h2.IP())))  
 print(q5)  
  
 # Q6: run UDP traffic sending 2 Gbytes between H1 and H3  
 h3.cmd('iperf3 -s &')  
 q6 = (h1.cmd('iperf3 -c {} -u -n 2'.format(h3.IP())))  
 print(q6)  
  
 # Q7: run TCP traffic between H1 and H3 for 20 seconds  
 h3.cmd('iperf3 -s &')  
 q7 = (h1.cmd('iperf3 -c {} -t 20 -i 1'.format(h3.IP())))  
 print(q7)  
  
 # Q8: store the output of a command (parts 3-7)  
 **with** open('output.txt', 'w') **as** f:  
 f.write('Q3 Output:\n')  
 f.write(q3 + '\n\n')  
  
 f.write('Q4 Output:\n')  
 f.write(q4 + '\n\n')  
  
 f.write('Q5 Output:\n')  
 f.write(q5 + '\n\n')  
  
 f.write('Q6 Output:\n')  
 f.write(q6 + '\n\n')  
  
 f.write('Q7 Output:\n')  
 f.write(q7 + '\n\n')  
  
 CLI(net)  
 net.stop()  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 create\_topology()

**“output.txt” file**

Q3 Output:  
  
Connecting to host 10.0.0.2, port 5201  
  
[ 5] local 10.0.0.1 port 34000 connected to 10.0.0.2 port 5201  
  
[ ID] Interval Transfer Bitrate Total Datagrams  
  
[ 5] 0.00-1.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 1.00-2.00 sec 127 KBytes 1.05 Mbits/sec 90   
  
[ 5] 2.00-3.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 3.00-4.00 sec 127 KBytes 1.04 Mbits/sec 90   
  
[ 5] 4.00-5.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 5.00-6.00 sec 127 KBytes 1.04 Mbits/sec 90   
  
[ 5] 6.00-7.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 7.00-8.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 8.00-9.01 sec 122 KBytes 987 Kbits/sec 86   
  
[ 5] 9.01-10.00 sec 134 KBytes 1.11 Mbits/sec 95   
  
- - - - - - - - - - - - - - - - - - - - - - - - -  
  
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams  
  
[ 5] 0.00-10.00 sec 1.25 MBytes 1.05 Mbits/sec 0.000 ms 0/906 (0%) sender  
  
[ 5] 0.00-10.08 sec 1.25 MBytes 1.04 Mbits/sec 0.761 ms 0/906 (0%) receiver  
  
  
  
iperf Done.  
  
  
  
  
  
Q4 Output:  
  
Connecting to host 10.0.0.3, port 5201  
  
[ 5] local 10.0.0.1 port 45960 connected to 10.0.0.3 port 5201  
  
[ ID] Interval Transfer Bitrate Total Datagrams  
  
[ 5] 0.00-1.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 1.00-2.00 sec 127 KBytes 1.04 Mbits/sec 90   
  
[ 5] 2.00-3.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 3.00-4.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 4.00-5.00 sec 124 KBytes 1.02 Mbits/sec 88   
  
[ 5] 5.00-6.01 sec 130 KBytes 1.06 Mbits/sec 92   
  
[ 5] 6.01-7.02 sec 127 KBytes 1.02 Mbits/sec 90   
  
[ 5] 7.02-8.00 sec 130 KBytes 1.09 Mbits/sec 92   
  
[ 5] 8.00-9.00 sec 127 KBytes 1.04 Mbits/sec 90   
  
[ 5] 9.00-10.00 sec 129 KBytes 1.06 Mbits/sec 91   
  
[ 5] 10.00-11.11 sec 112 KBytes 822 Kbits/sec 79   
  
[ 5] 11.11-12.03 sec 123 KBytes 1.09 Mbits/sec 87   
  
[ 5] 12.03-13.01 sec 148 KBytes 1.25 Mbits/sec 105   
  
[ 5] 13.01-14.00 sec 126 KBytes 1.04 Mbits/sec 89   
  
[ 5] 14.00-15.05 sec 127 KBytes 998 Kbits/sec 90   
  
[ 5] 15.05-16.00 sec 130 KBytes 1.12 Mbits/sec 92   
  
[ 5] 16.00-17.00 sec 129 KBytes 1.06 Mbits/sec 91   
  
[ 5] 17.00-18.00 sec 129 KBytes 1.05 Mbits/sec 91   
  
[ 5] 18.00-19.01 sec 126 KBytes 1.03 Mbits/sec 89   
  
[ 5] 19.01-20.02 sec 129 KBytes 1.04 Mbits/sec 91   
  
- - - - - - - - - - - - - - - - - - - - - - - - -  
  
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams  
  
[ 5] 0.00-20.02 sec 2.50 MBytes 1.05 Mbits/sec 0.000 ms 0/1810 (0%) sender  
  
[ 5] 0.00-20.18 sec 2.50 MBytes 1.04 Mbits/sec 8.731 ms 0/1810 (0%) receiver  
  
  
  
iperf Done.  
  
  
  
  
  
Q5 Output:  
  
Connecting to host 10.0.0.2, port 5201  
  
[ 5] local 10.0.0.1 port 46557 connected to 10.0.0.2 port 5201  
  
[ ID] Interval Transfer Bitrate Total Datagrams  
  
[ 5] 0.00-0.00 sec 1.41 KBytes 827 Mbits/sec 1   
  
- - - - - - - - - - - - - - - - - - - - - - - - -  
  
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams  
  
[ 5] 0.00-0.00 sec 1.41 KBytes 827 Mbits/sec 0.000 ms 0/1 (0%) sender  
  
[ 5] 0.00-0.09 sec 0.00 Bytes 0.00 bits/sec 0.000 ms 0/0 (0%) receiver  
  
  
  
iperf Done.  
  
  
  
  
  
Q6 Output:  
  
Connecting to host 10.0.0.3, port 5201  
  
[ 5] local 10.0.0.1 port 52885 connected to 10.0.0.3 port 5201  
  
[ ID] Interval Transfer Bitrate Total Datagrams  
  
[ 5] 0.00-0.00 sec 1.41 KBytes 1.05 Gbits/sec 1   
  
- - - - - - - - - - - - - - - - - - - - - - - - -  
  
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams  
  
[ 5] 0.00-0.00 sec 1.41 KBytes 1.05 Gbits/sec 0.000 ms 0/1 (0%) sender  
  
[ 5] 0.00-0.17 sec 0.00 Bytes 0.00 bits/sec 0.000 ms 0/0 (0%) receiver  
  
  
  
iperf Done.  
  
  
  
  
  
Q7 Output:  
  
Connecting to host 10.0.0.3, port 5201  
  
[ 5] local 10.0.0.1 port 36542 connected to 10.0.0.3 port 5201  
  
[ ID] Interval Transfer Bitrate Retr Cwnd  
  
[ 5] 0.00-1.00 sec 1.25 MBytes 10.4 Mbits/sec 0 202 KBytes   
  
[ 5] 1.00-2.00 sec 445 KBytes 3.65 Mbits/sec 0 223 KBytes   
  
[ 5] 2.00-3.00 sec 1.55 MBytes 13.1 Mbits/sec 0 286 KBytes   
  
[ 5] 3.00-4.00 sec 1.24 MBytes 10.4 Mbits/sec 0 349 KBytes   
  
[ 5] 4.00-5.00 sec 2.24 MBytes 18.8 Mbits/sec 0 413 KBytes   
  
[ 5] 5.00-6.00 sec 891 KBytes 7.29 Mbits/sec 0 478 KBytes   
  
[ 5] 6.00-7.00 sec 2.05 MBytes 17.2 Mbits/sec 0 542 KBytes   
  
[ 5] 7.00-8.00 sec 1.12 MBytes 9.39 Mbits/sec 0 605 KBytes   
  
[ 5] 8.00-9.00 sec 1.24 MBytes 10.4 Mbits/sec 0 670 KBytes   
  
[ 5] 9.00-10.00 sec 2.50 MBytes 21.0 Mbits/sec 0 734 KBytes   
  
[ 5] 10.00-11.00 sec 1.25 MBytes 10.5 Mbits/sec 0 799 KBytes   
  
[ 5] 11.00-12.00 sec 1.25 MBytes 10.5 Mbits/sec 0 918 KBytes   
  
[ 5] 12.00-13.00 sec 2.50 MBytes 21.0 Mbits/sec 0 1.08 MBytes   
  
[ 5] 13.00-14.00 sec 1.25 MBytes 10.5 Mbits/sec 0 1.30 MBytes   
  
[ 5] 14.00-15.00 sec 2.50 MBytes 21.0 Mbits/sec 0 1.55 MBytes   
  
[ 5] 15.00-16.00 sec 2.50 MBytes 21.0 Mbits/sec 0 1.83 MBytes   
  
[ 5] 16.00-17.00 sec 2.50 MBytes 21.0 Mbits/sec 0 2.15 MBytes   
  
[ 5] 17.00-18.00 sec 2.50 MBytes 21.0 Mbits/sec 0 2.48 MBytes   
  
[ 5] 18.00-19.00 sec 1.25 MBytes 10.5 Mbits/sec 0 2.89 MBytes   
  
[ 5] 19.00-20.00 sec 1.25 MBytes 10.5 Mbits/sec 0 3.31 MBytes   
  
- - - - - - - - - - - - - - - - - - - - - - - - -  
  
[ ID] Interval Transfer Bitrate Retr  
  
[ 5] 0.00-20.00 sec 33.2 MBytes 13.9 Mbits/sec 0 sender  
  
[ 5] 0.00-22.29 sec 24.6 MBytes 9.26 Mbits/sec receiver  
  
  
  
iperf Done.

1. **Wireshark Assignment 3**

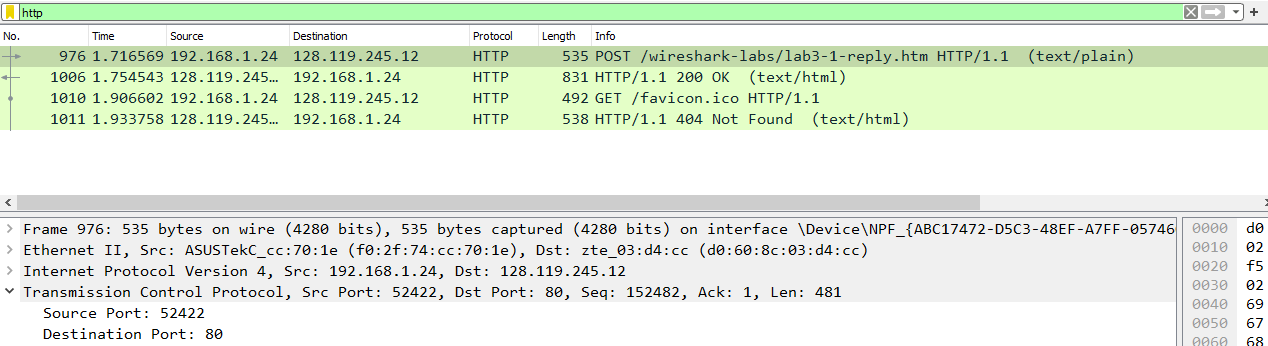
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window” (refer to Figure 2 in the “Getting Started with Wireshark” Lab if you’re uncertain about the Wireshark windows.





Source IP Address: 192.168.1.24, Source Port: 52422

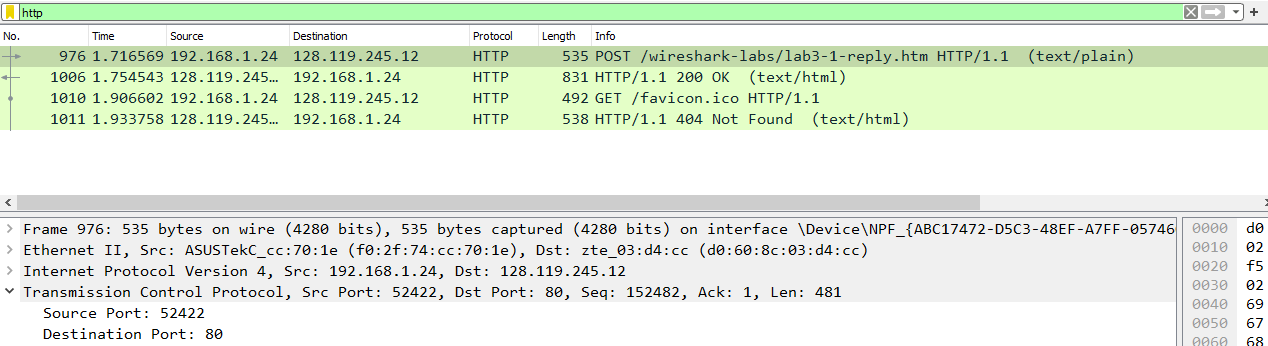
2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?





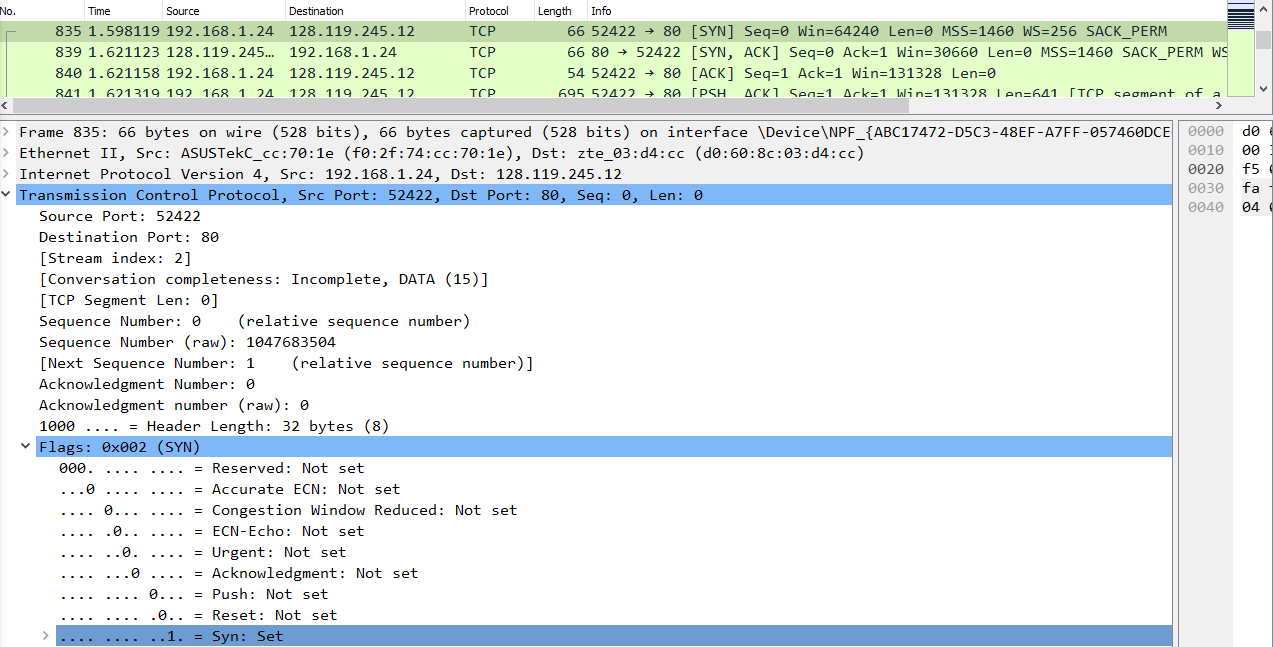
Destination IP Address: 128.119.245.12, Destination Port: 80

3. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?



Source IP Address: 192.168.1.24, Source Port: 52422

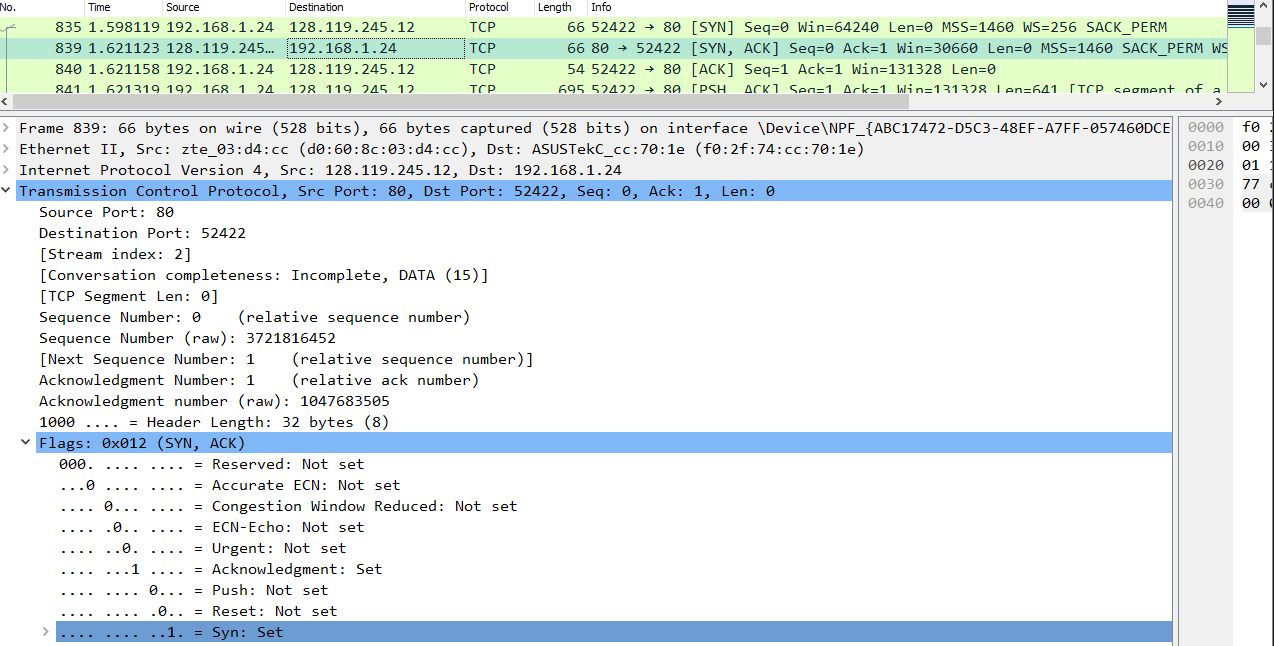
4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?





Sequence number of SYN is 0. “Syn” flag is “Set” means that it is a SYN segment.

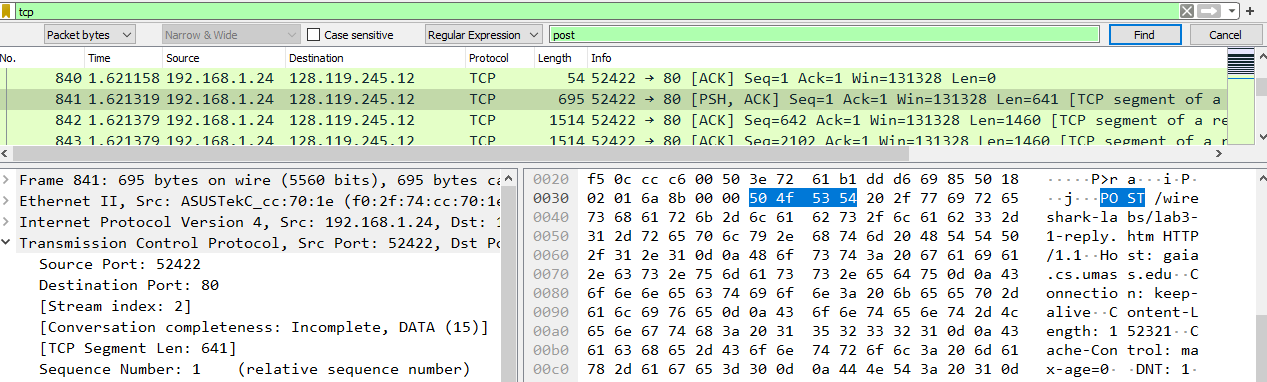
5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?





Sequence number of SYNACK segment is 0. The Acknowledgement field is 1. It is determined by incrementing the initial sequence by 1. The segment that identifies the segment as a SYNACK segment is the Acknowledgment “Set” and Syn “Set” flags.

6. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you’ll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a “POST” within its DATA field.

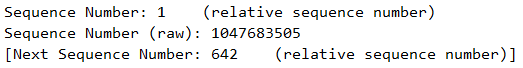




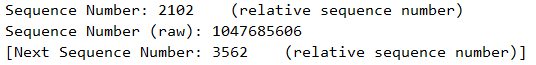
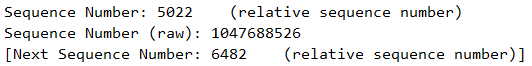
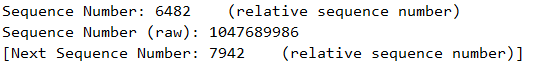
Sequence number of the TCP segment containing the HTTP POST is 1.

7. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 242 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 242 for all subsequent segments.

-Sequence numbers of the first six segments:

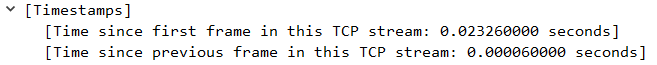
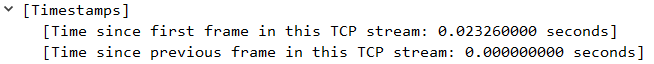
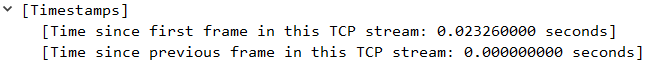
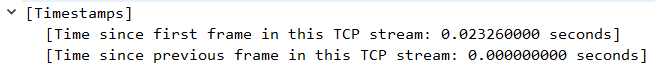
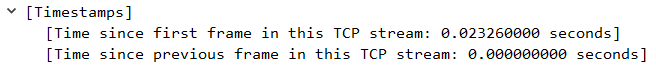
 





-Time of each segment SENT (red) and RECEIVED (blue):

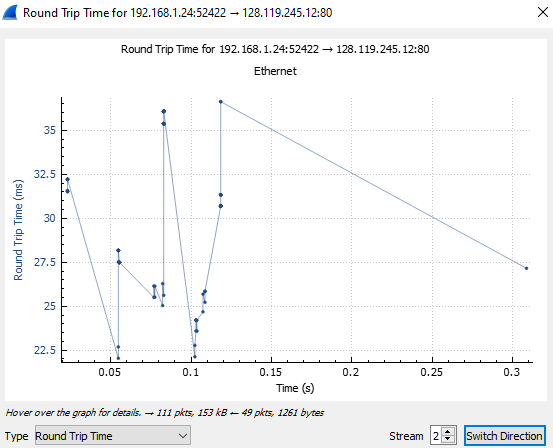
     



-RTT value for each six segment (subtraction between sent and received):

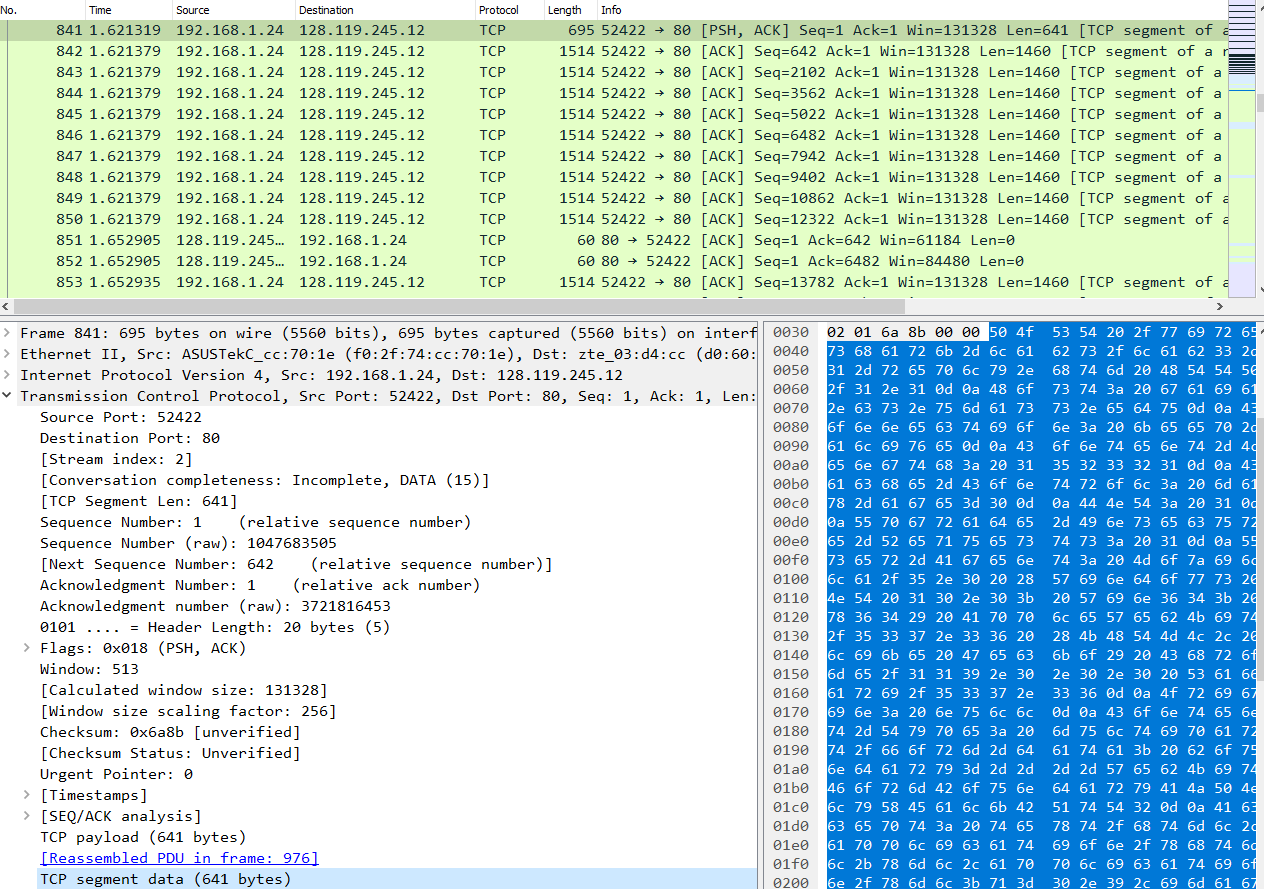
     

-Estimate RTT time: The average RTT time is 0.023039 seconds



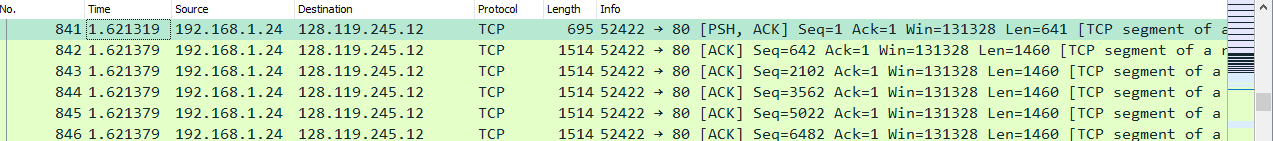
Sample image of the round-trip graph of the first HTTP POST segment.

8. What is the length of each of the first six TCP segments?4



9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?



Since Win are all 131328, the minimum amount of buffer space is 131328. It does not throttle.

10. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

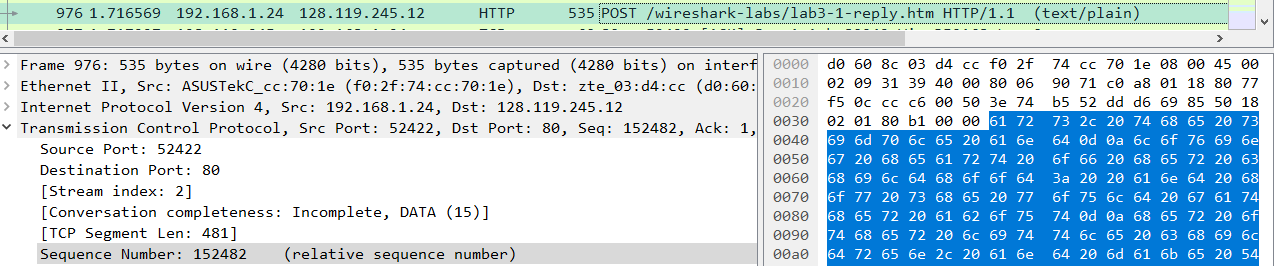
There are no retransmitted segments in the trace file. This can be explained by the fact that no same sequence number appears at two different times, thus no re-requests of previous segments.

11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).

The ACK number increases 1420 bytes each time. I couldn’t find in my cases of a receiver ACKing every other received segment.

12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

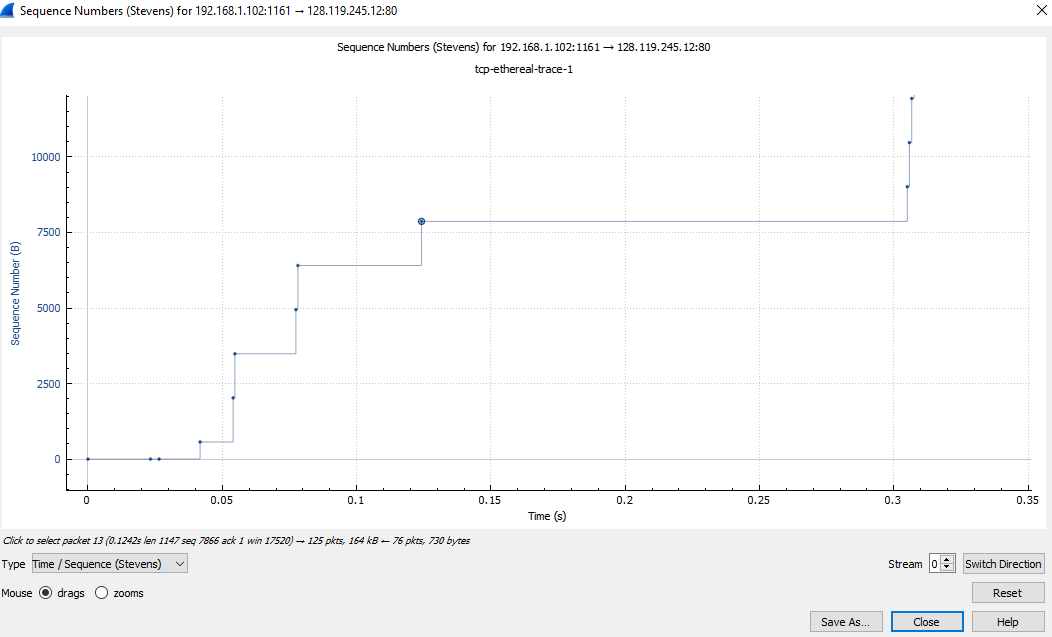
The throughput is the number of bytes transferred per time in which the number of bytes transferred represents the difference between the first and last segment numbers. Knowing that the first segment number is 1 and the last segment number is 152482, we can calculate the data transferred to be 152481 bytes. As for the time difference, it is 1.716569-1.621319 = 0.09525. As such, the throughput is 152482/0.09525 = 1600860.89239 bytes per second.





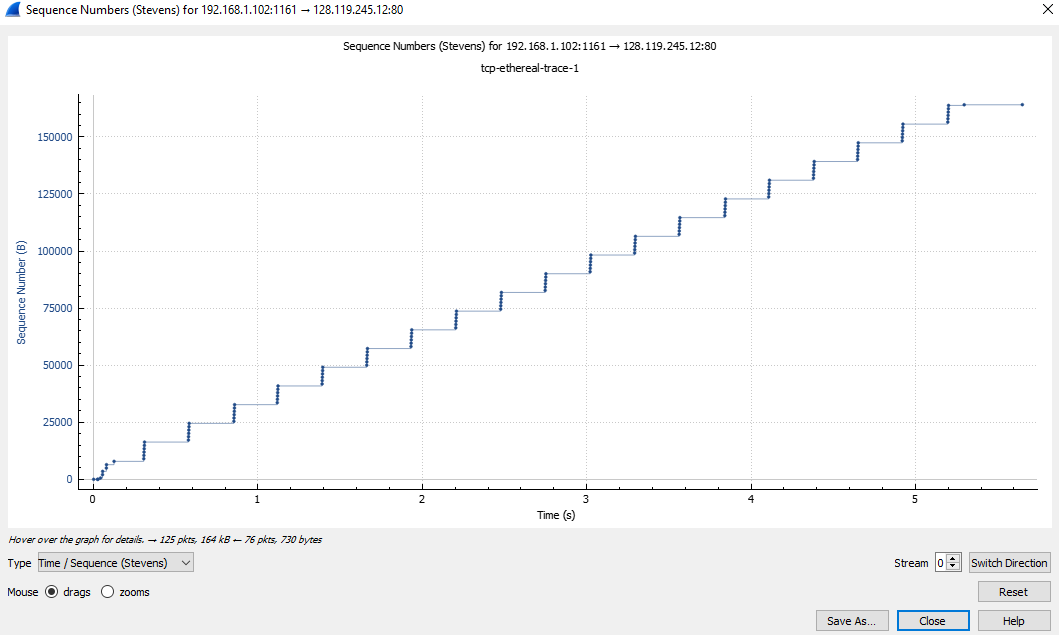
13. Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP’s slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we’ve studied in the text.

-The slowstart phase begins at time 0.00 seconds and ends at 0.1242 seconds.





-It is in congestion (horizontal line) every 6 points





-The ideal behavior of a TCP connection is to allow data to be transmitted as fast as possible with no lost in data while also not too fast which results in continuous queuing delay. In this case, there is a difference where the duration of each congestion has a fluctuation, thus different delay each time. The slowstart can also vary depending on the TCP connections.

1. **Concepts Learned from this Lab**

For Mininet lab, I learned abut the effect of bandwidth and delay on a network. I also practiced with “iPerf3”, a real-time throughput measurement tool, to create a two switch and three hosts topology network. For Wireshark lab, I practice on a basic TCP protocol and learned about sequence and acknowledgement numbers. Finally, I worked on TCP congestion control in action.