Lab3 - Working with Mininet and the POX Controller

100 points

In Prelab3, we learned about Software Defined Networking (SDN) and the POX Controller to design our very own Firewall. In this Lab, we are going to learn and experiment some more using the POX Controller to build a router. We will also examine working with an HTTP Server within Mininet. The assignment concludes with some very interesting exercises that explore the internal workings to the TCP protocol.

Helpful Resources:

- **http://mininet.org/walkthrough/#run-a-simple-web-server-and-client**
- Computer Networks: A Top Down Approach - Chapter 3 covers TCP
- **POX Wiki**
- Inside your VM, the pox/forwarding/l2_learning.py example file

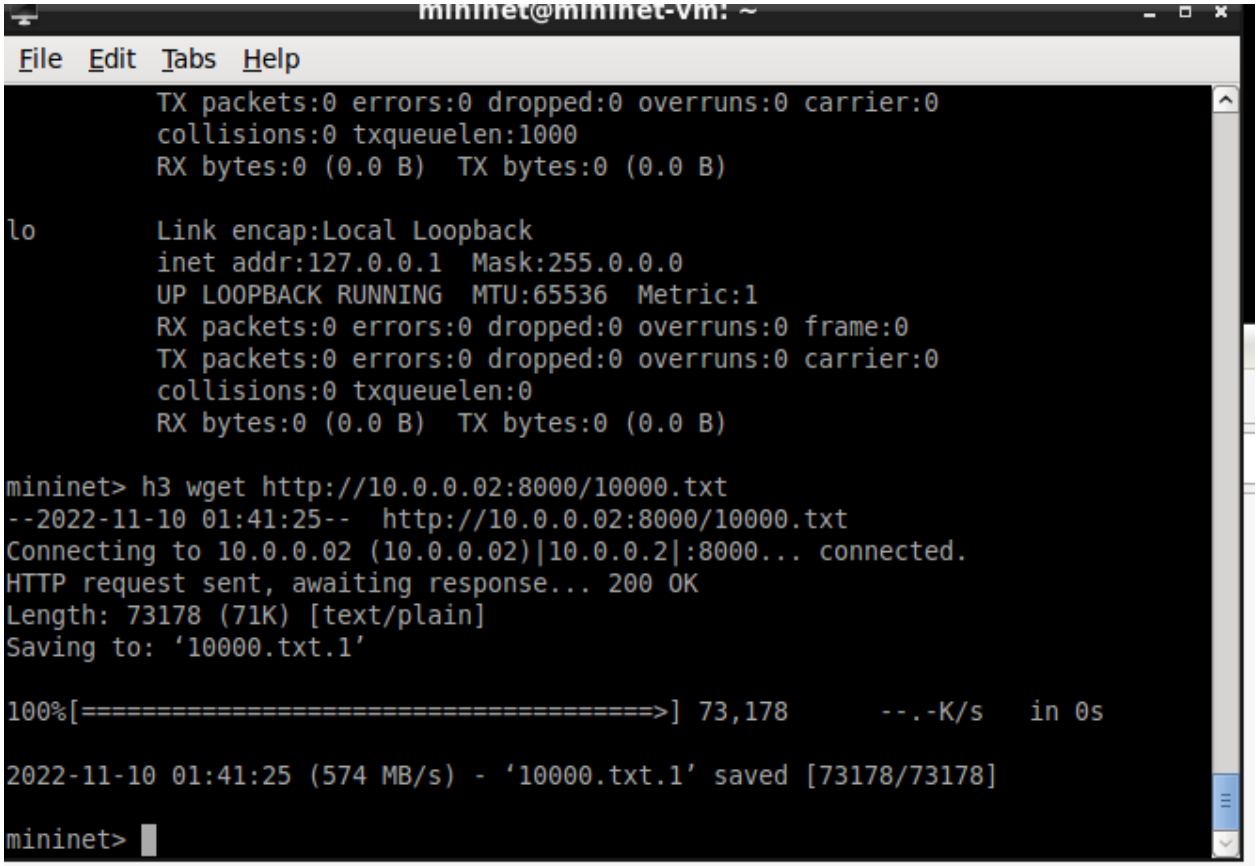
Screenshots: For all the questions below that require a screenshot, make sure that a date timestamp is visible next to your results. Points will be deducted if a visible timestamp is missing. *For explanations that use results from screenshots, you must either reference specific elements of the screenshot (packet number, terminal command, etc.) in your explanation or annotate the screenshot in some way (highlighting, drawing tool, etc.).*

HTTP Request-Response Generation [25 pts]

Let's use the Mininet HTTP Server! Using the **old VM**, create a simple connected Mininet topology with hosts using the command:

sudo mn --topo=single,3

Configure `h2` as an HTTP server (see Mininet Walkthrough) hosting the 10000.txt file. Then, send an HTTP request from `h3` to `h2` for the file 10000.txt. (Hint: you can make use of `wget` to download the file from the server.)

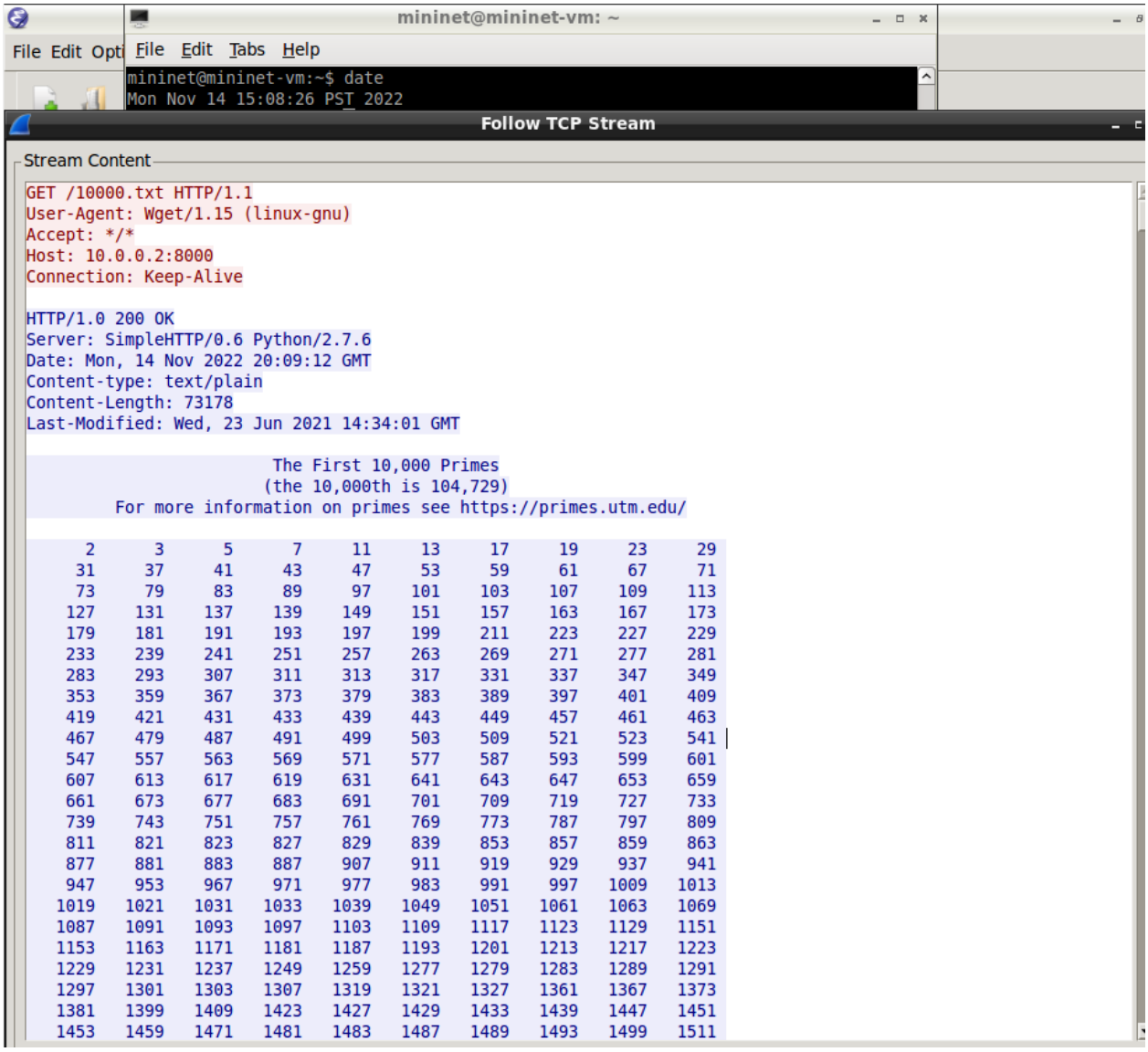


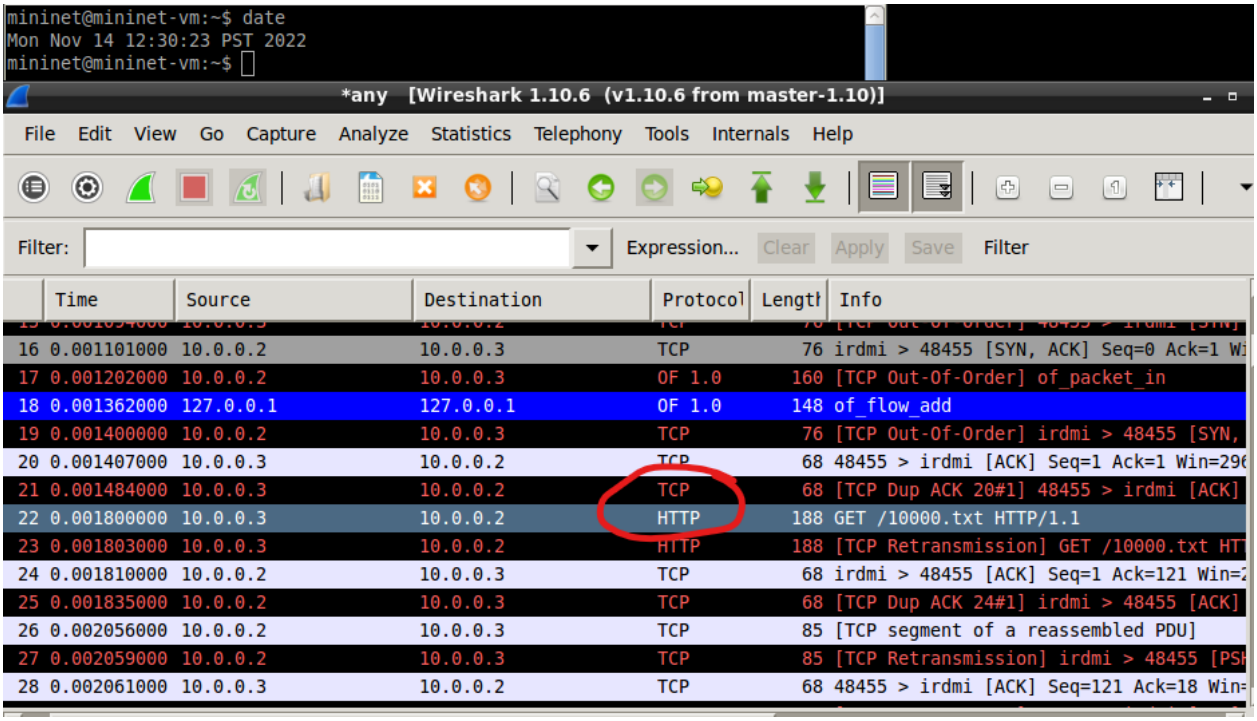
1) **[9 pts]** Capture the transaction using Wireshark and use a filter to display client and server traffic only.
   - What are protocols used and what are they used for in the transaction?

HTTP protocol is used for a query and response for the 10000.txt file and TCP is used to establish the 3 way handshake connection. ARP is used to find the MAC address of h2 and h3.

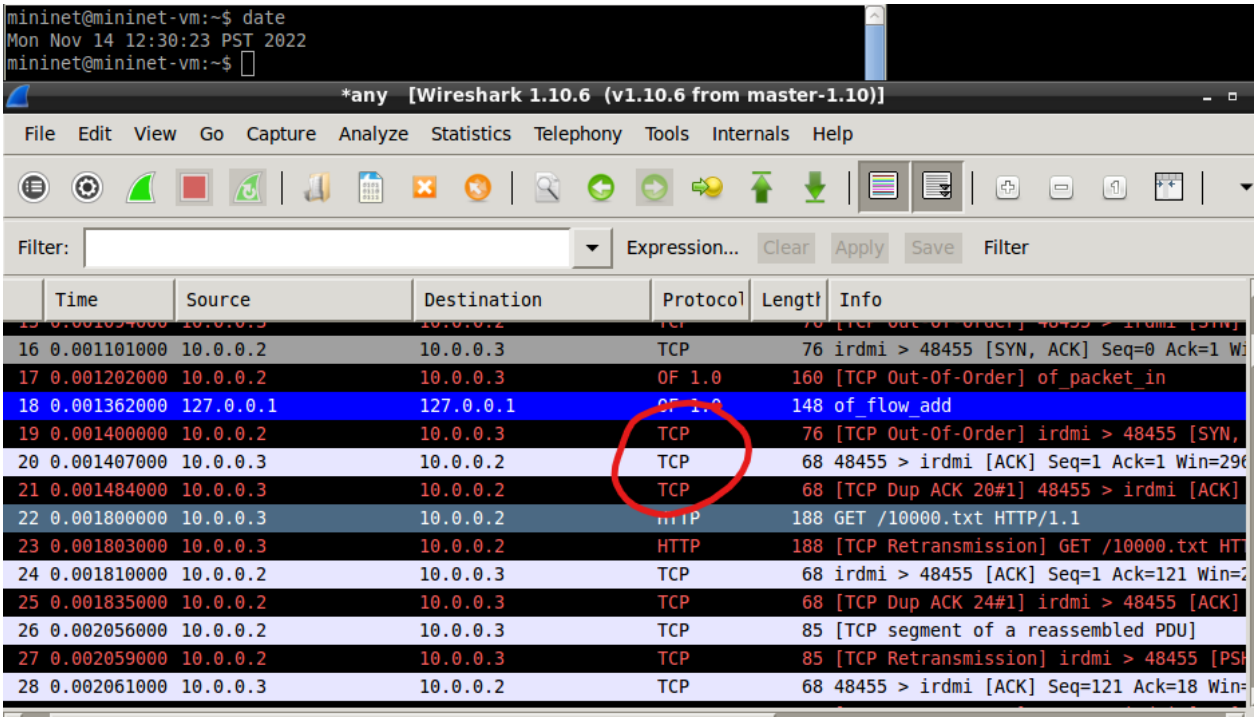
- In Wireshark take a screenshot of the transaction between the client and server and highlight two protocols. Ignore OpenFlow packets (OF).

I am including this screenshot because the HTTP GET Response was not shown in wireshark.





2) **[6 pts]** In the Wireshark screenshot, highlight the TCP handshake. State their frame number (No. column).



The frame numbers for the TCP handshake are 19-21.

3) **[10 pts]** Describe ALL of the steps (from beginning to the **end**) you observe

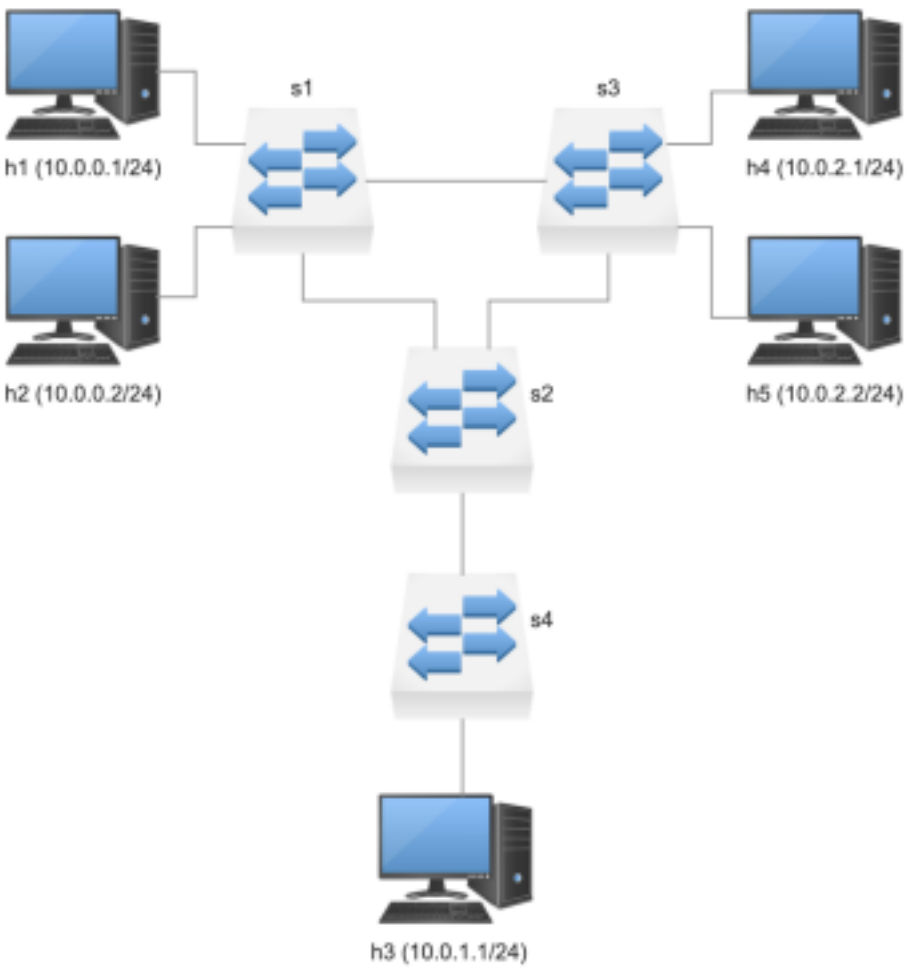
taken by the client to retrieve the file from the server.

1. **H3 establishes a TCP handshake with h2.**
2. **H3 sends a http GET request for the 10000.txt file**
3. **Packets are being transmitted from h2 to h3 and h3 is sending back ACK responses.**
4. **H3 receives the file from h2 eventually.**

Router [38 pts]

Your goal will be to allow or block traffic between the different devices based on the 3 rules given below. **You will need to specify specific ports for all traffic (i.e., you cannot use flooding: `of.OFPP_FLOOD`).** You may do this however you choose—although as a suggestion, you may find it easiest to determine the correct destination port by using the source and destination IP addresses or the source port on the switch from which the packet originated.

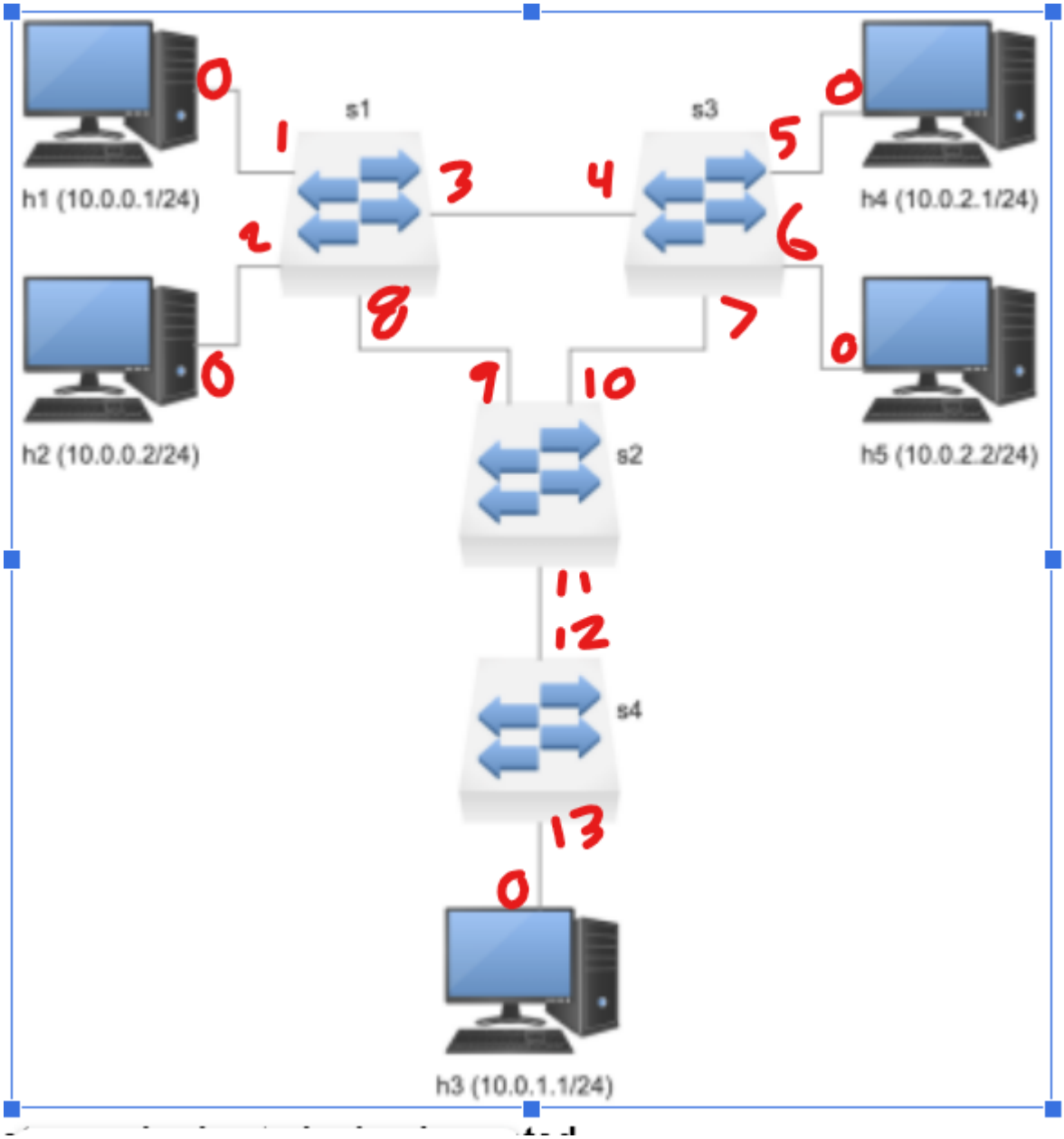
These files will get you started—you will need to modify both:
lab3_topo_skel.py / lab3_controller_skel.py
Finish the topology file by adding all the hosts (be sure to manually specify the MAC address, IP address and subnet for each host), switches and links, then implement the controller based on Rule 1, Rule 2 and Rule 3.



Network setup and rules to be implemented

- **Rule 1:** ICMP traffic is forwarded only between subnets 10.0.0.0/24 and 10.0.1.0/24 or between devices that are on the same subnet.
- **Rule 2:** TCP traffic is forwarded only between subnets 10.0.0.0/24 and 10.0.2.0/24 or between devices that are on the same subnet.
- **Rule 3:** All other traffic should be dropped.

1) **[11 pts]** Implement the topology above from the skeleton file. Annotate the figure with the ports associated with each end of each link and include your annotated figure in your submission.

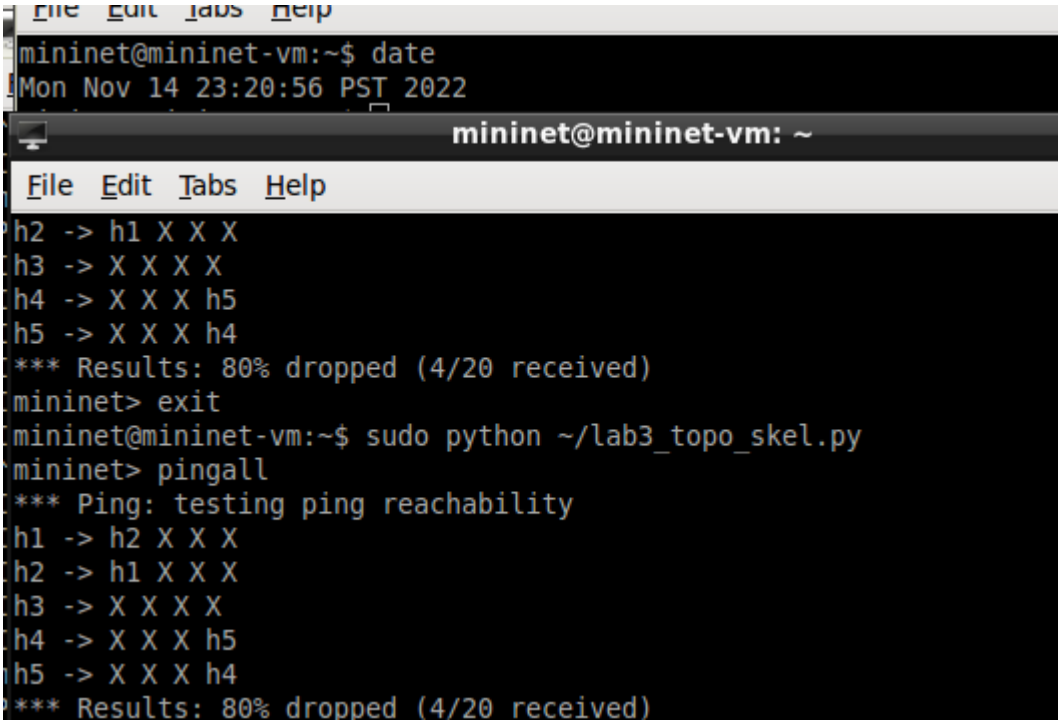


2) **[5 pts]** Explain how packets are forwarded differently in this assignment compared to Prelab3. Think about your accept() function in Prelab 3.

Packets are forwarded via ports from each switch and host. Instead of flooding the ports, we are assigning each switch and host a port number.

3) **[10 pts]** Verify Rule 1 with *pingall*. Are the results what you expect? Why?

Include a screenshot and explanation.

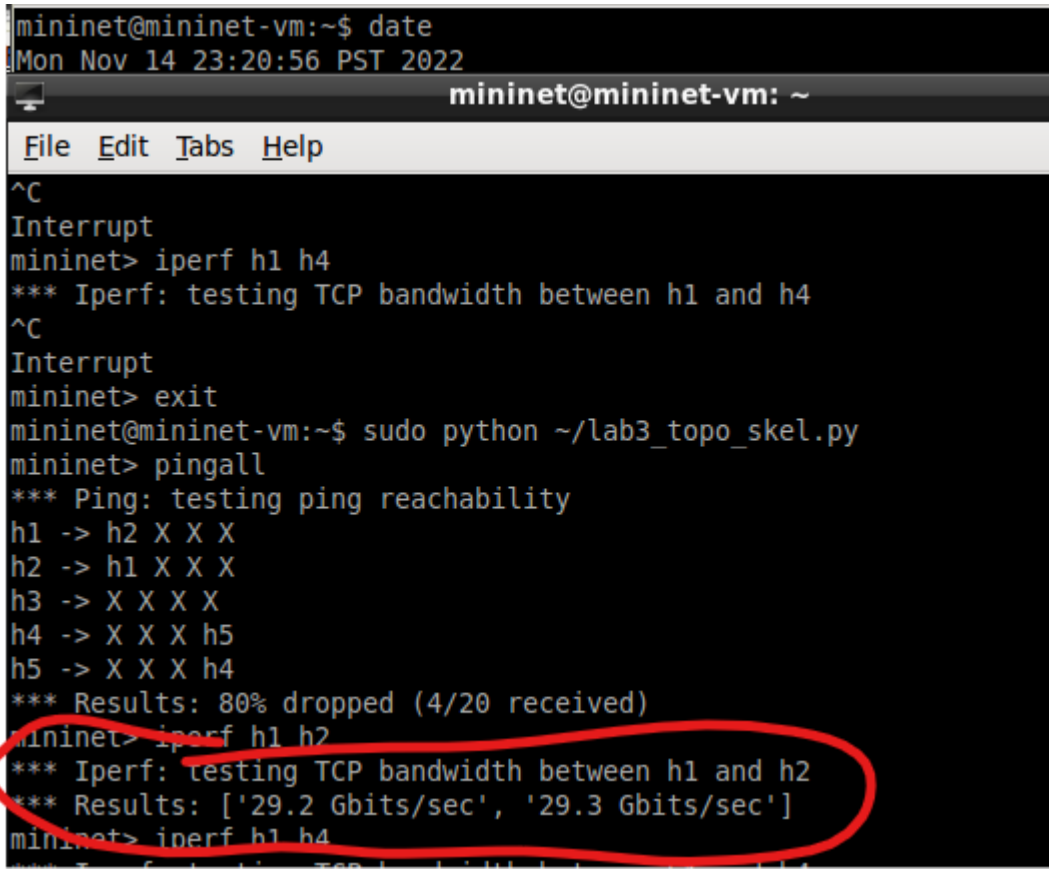


These are not the results I was expecting. I want h1 to ping both h2 and h3, h2 to ping h1 and h3, and for h3 to ping h1 and h2.

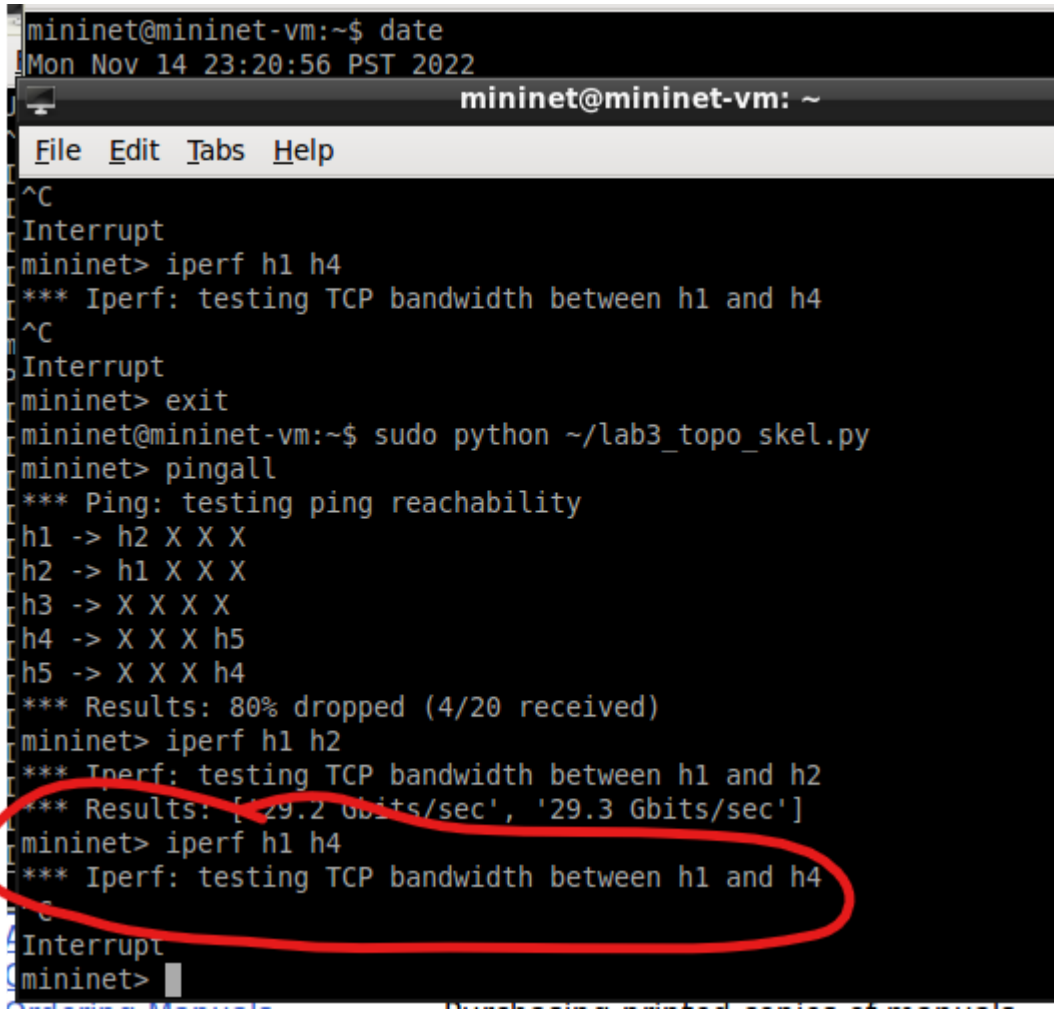
4) **[12 pts]** Verify Rule 2 with *iperf*. Are the results what you expect? Why? Include a screenshot of an *iperf* between these pairs of nodes:

These results are not what I expected, since my code isn't fully fleshed out. H1 and h2 should have a connection, and h4 and h5 should also have a connection, but I don't have a connection between IP addresses with subnets 0 and 2.

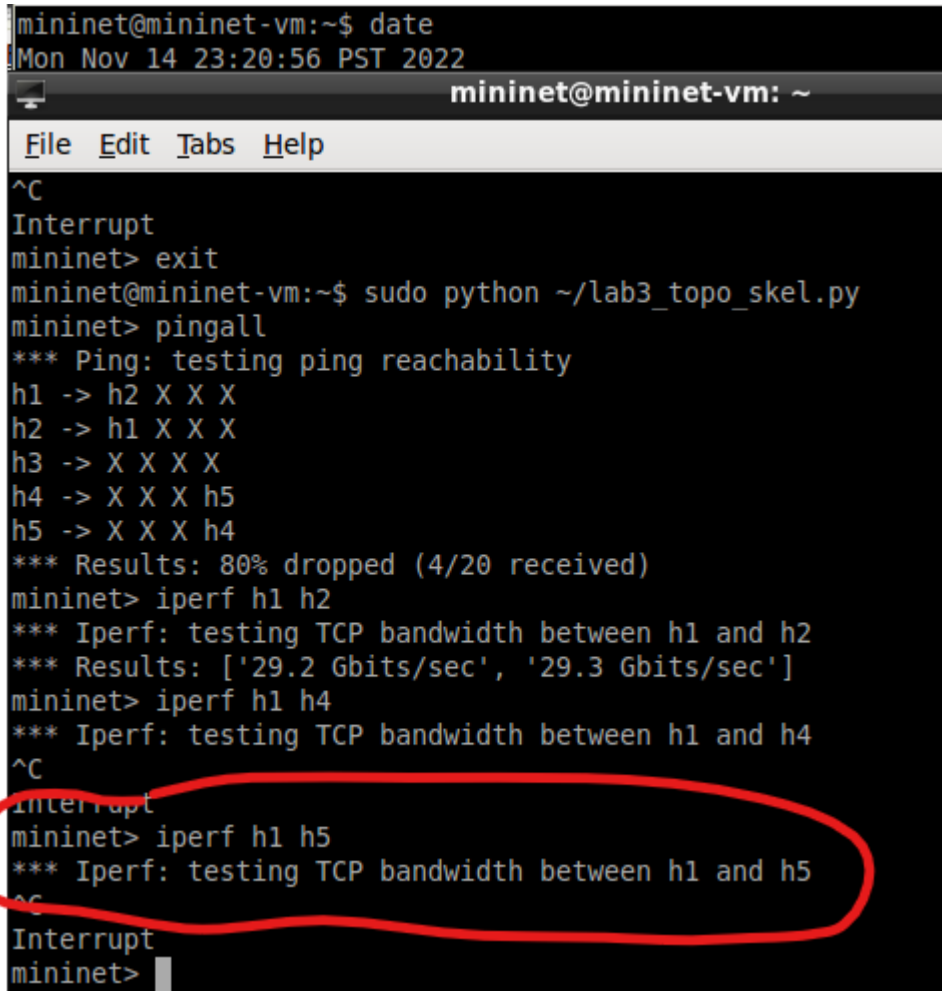
- h1 and h2


mininet@mininet-vm:~$ date
Mon Nov 14 23:20:56 PST 2022
mininet@mininet-vm: ~
File Edit Tabs Help
^C
Interrupt
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
^C
Interrupt
mininet> exit
mininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X
h2 -> h1 X X X
h3 -> X X X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 80% dropped (4/20 received)
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['29.2 Gbits/sec', '29.3 Gbits/sec']

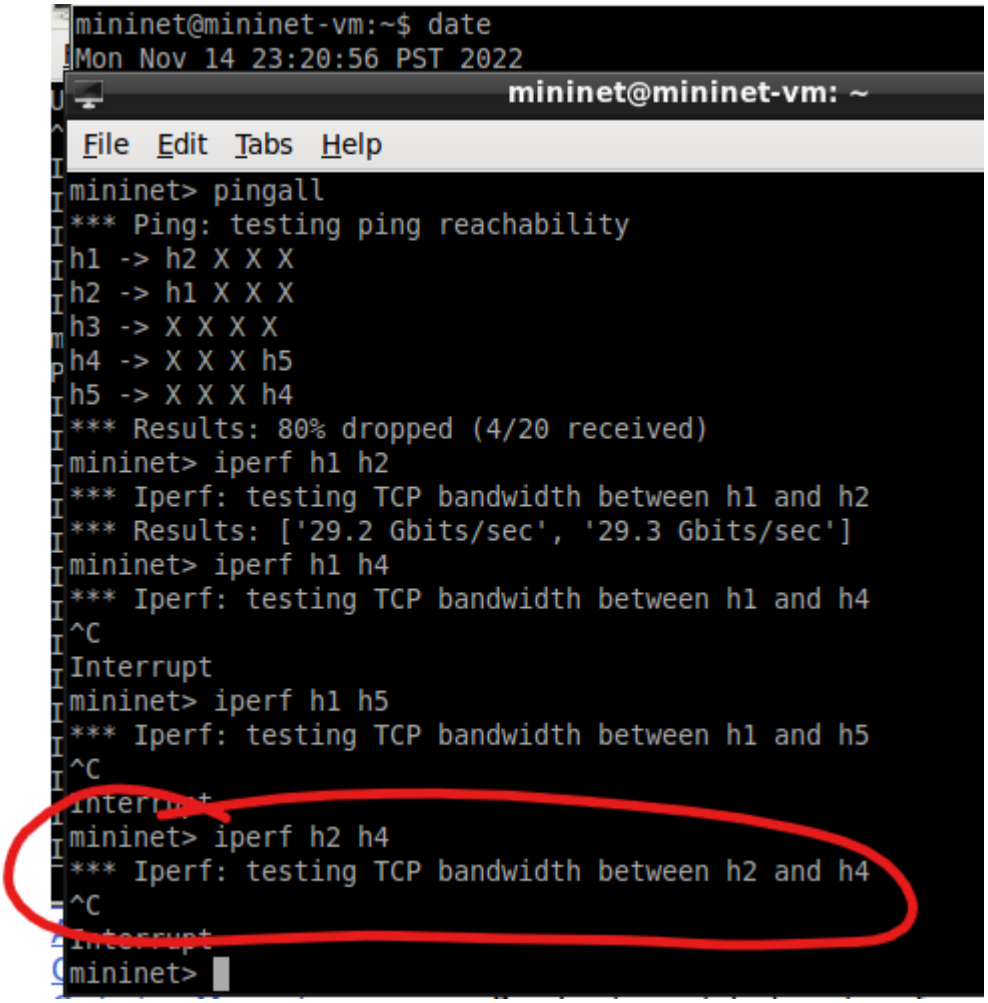
- h1 and h4


mininet@mininet-vm:~$ date
Mon Nov 14 23:20:56 PST 2022
mininet@mininet-vm: ~
File Edit Tabs Help
^C
Interrupt
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
^C
Interrupt
mininet> exit
mininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X
h2 -> h1 X X X
h3 -> X X X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 80% dropped (4/20 received)
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['29.2 Gbits/sec', '29.3 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
Interrupt
mininet>

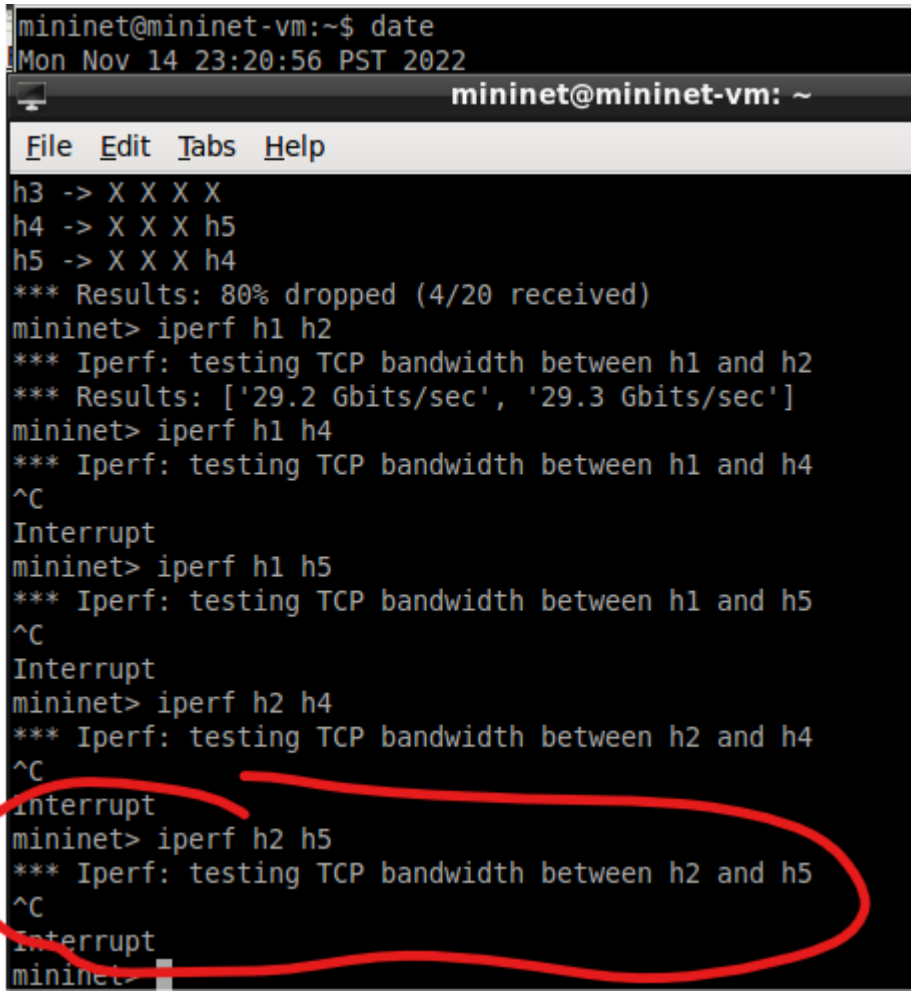
- h1 and h5


mininet@mininet-vm:~$ date
Mon Nov 14 23:20:56 PST 2022
mininet@mininet-vm: ~
File Edit Tabs Help
^C
Interrupt
mininet> exit
mininet@mininet-vm:~$ sudo python ~/lab3_topo_skel.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X
h2 -> h1 X X X
h3 -> X X X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 80% dropped (4/20 received)
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['29.2 Gbits/sec', '29.3 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
^C
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
Interrupt
mininet>

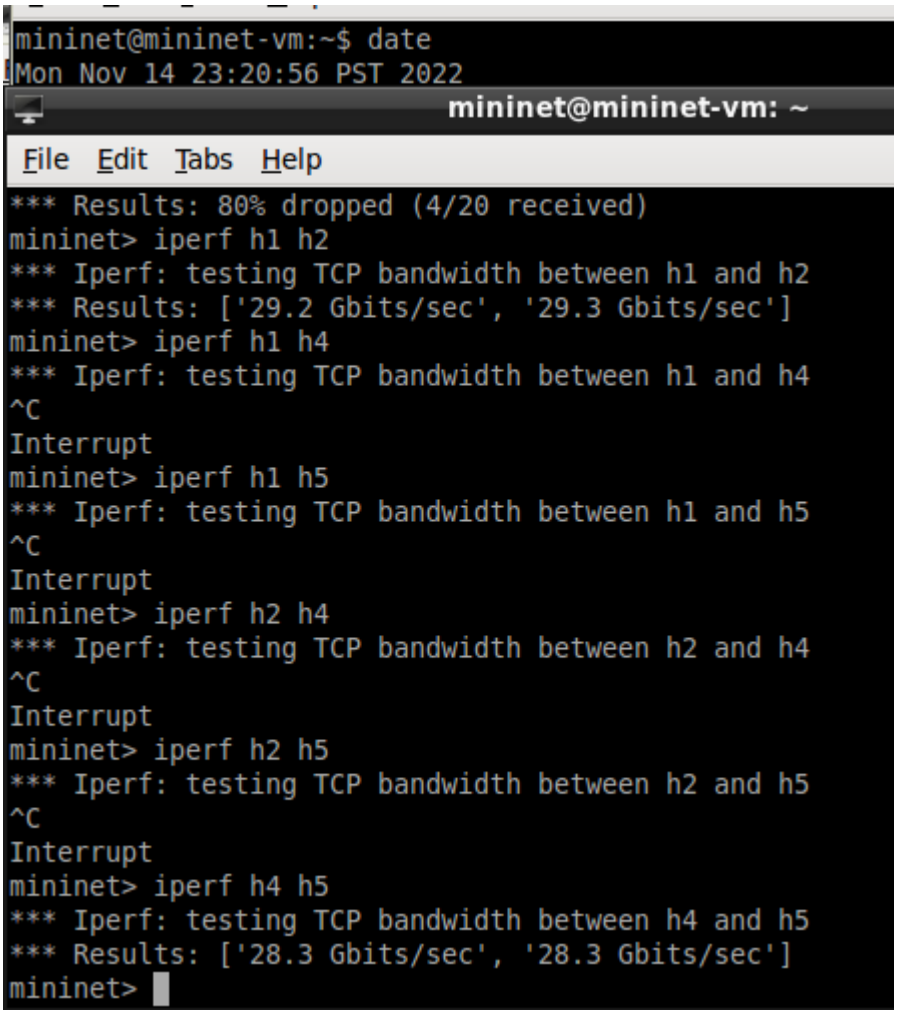
- h2 and h4


mininet@mininet-vm:~$ date
Mon Nov 14 23:20:56 PST 2022
mininet@mininet-vm: ~
File Edit Tabs Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X X X
h2 -> h1 X X X
h3 -> X X X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 80% dropped (4/20 received)
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['29.2 Gbits/sec', '29.3 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
^C
Interrupt
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
^C
mininet> iperf h2 h4
*** Iperf: testing TCP bandwidth between h2 and h4
^C
mininet>

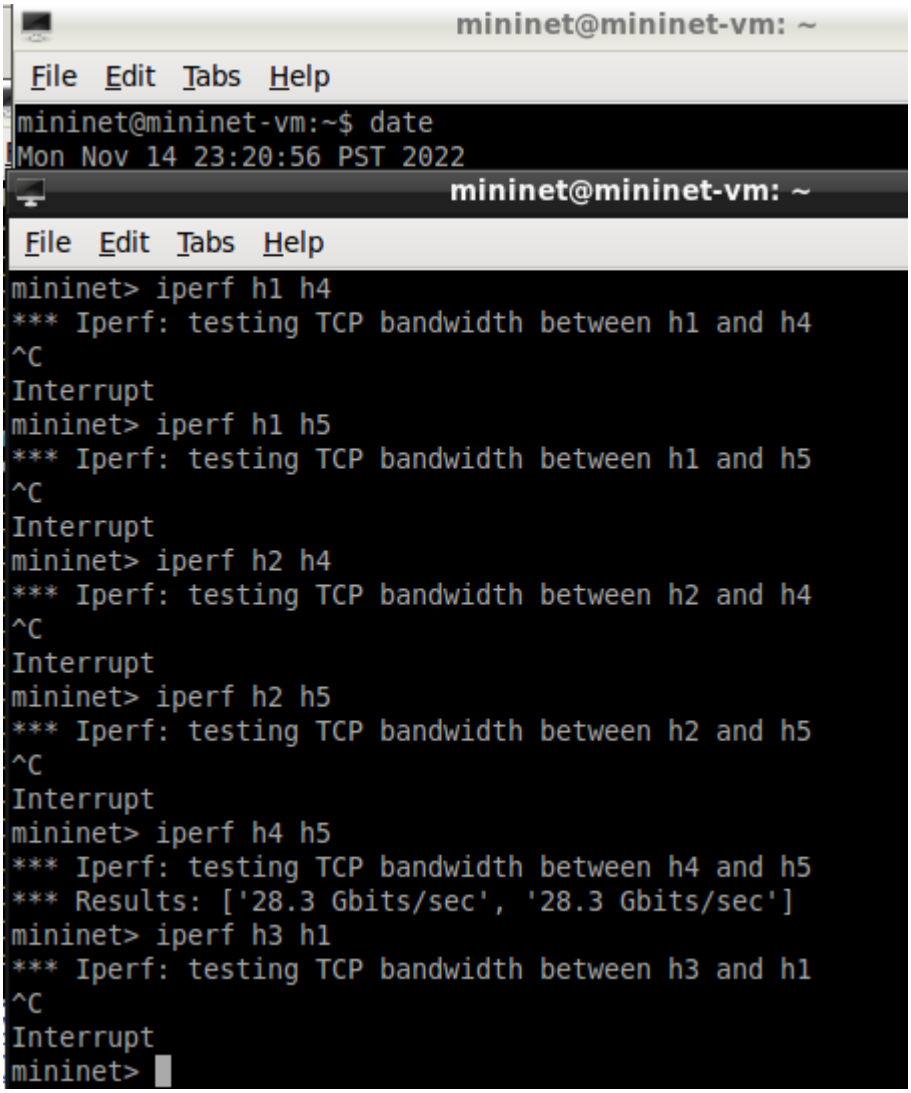
- h2 and h5


mininet@mininet-vm:~$ date
Mon Nov 14 23:20:56 PST 2022
mininet@mininet-vm: ~
File Edit Tabs Help
h3 -> X X X X
h4 -> X X X h5
h5 -> X X X h4
*** Results: 80% dropped (4/20 received)
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['29.2 Gbits/sec', '29.3 Gbits/sec']
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
^C
Interrupt
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
^C
Interrupt
mininet> iperf h2 h4
*** Iperf: testing TCP bandwidth between h2 and h4
^C
Interrupt
mininet> iperf h2 h5
*** Iperf: testing TCP bandwidth between h2 and h5
^C
Interrupt

- h4 and h5



- h3 and any other host



Important: If *iperf* seems to take forever (longer than around 15 seconds), this usually means the hosts can't reach each other; cancel it by pressing ctrl-c but include the result in your screenshot anyway.

TCP: [37 pts]

In this section we will explore the inner workings of TCP!

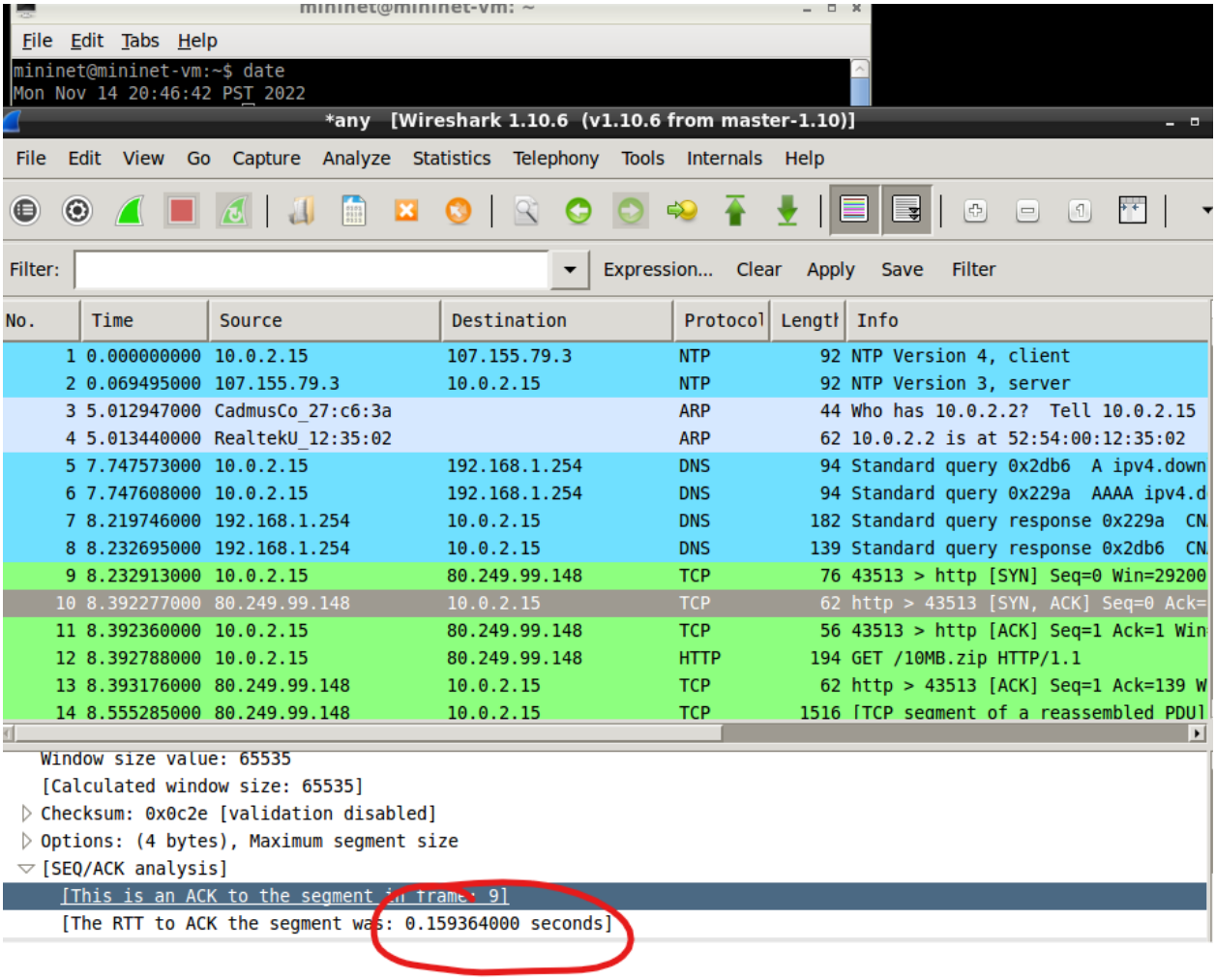
For the following questions, open Wireshark in the VM and listen on the 'any' interface.

In a terminal window use wget to download this file:
http://ipv4.download.thinkbroadband.com/10MB.zip

1) [4 pts] Answer the following questions based on the Wireshark trace. Take a screenshot and highlight the requested information below.
    a) **[1 pt]** How much time elapses between the transmission of the SYN

packet and the arrival of the SYN-ACK packet?

.159 seconds (SCREENSHOT BELOW)

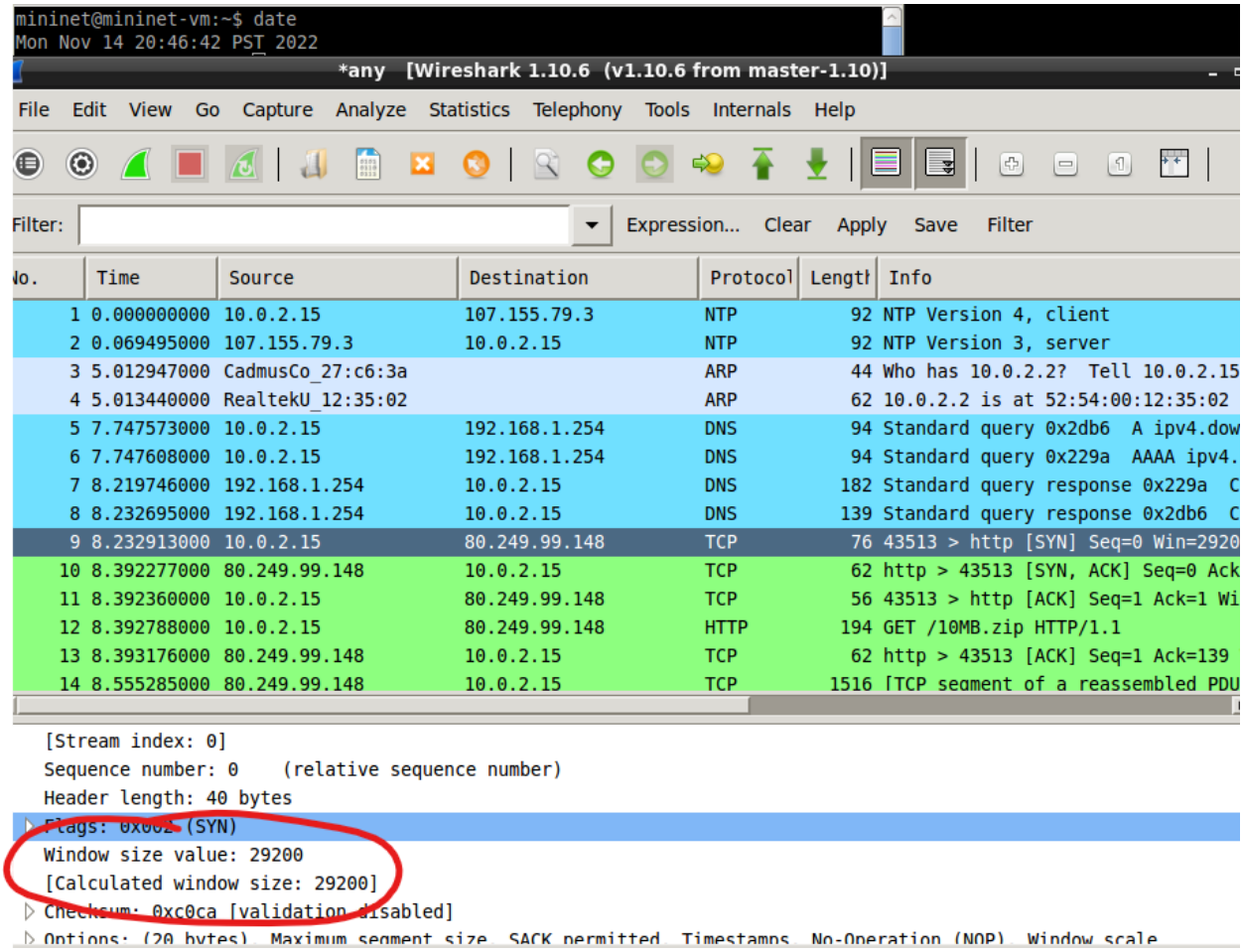


b) **[1 pt]** What does the window size indicate in the packet header? (No screenshot required)

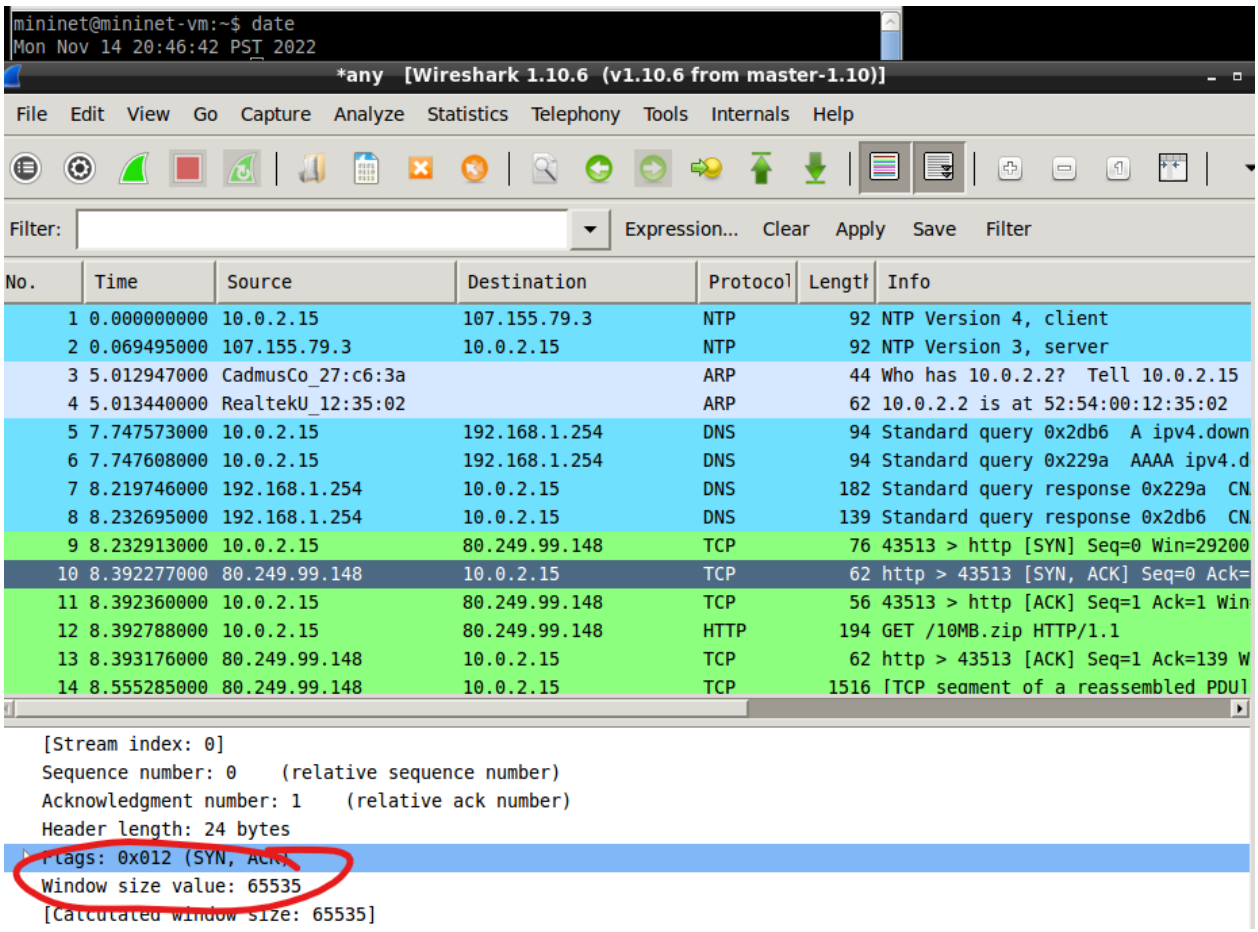
The window size indicates how much data the server is willing to receive.

c) **[1 pt]** What was the initial window size that your computer advertised to the server?

The initial window size was 29200.



d) **[1 pt]** What was the initial window size that the server advertised to you?
The initial window size was 65535.



2) **[7 pts]** Now we are going to examine the transfer of packets in TCP using the tcptrace graph from Wireshark. Here is a great video to introduce you to this: https://www.youtube.com/watch?v=yUmACeSmT7o)

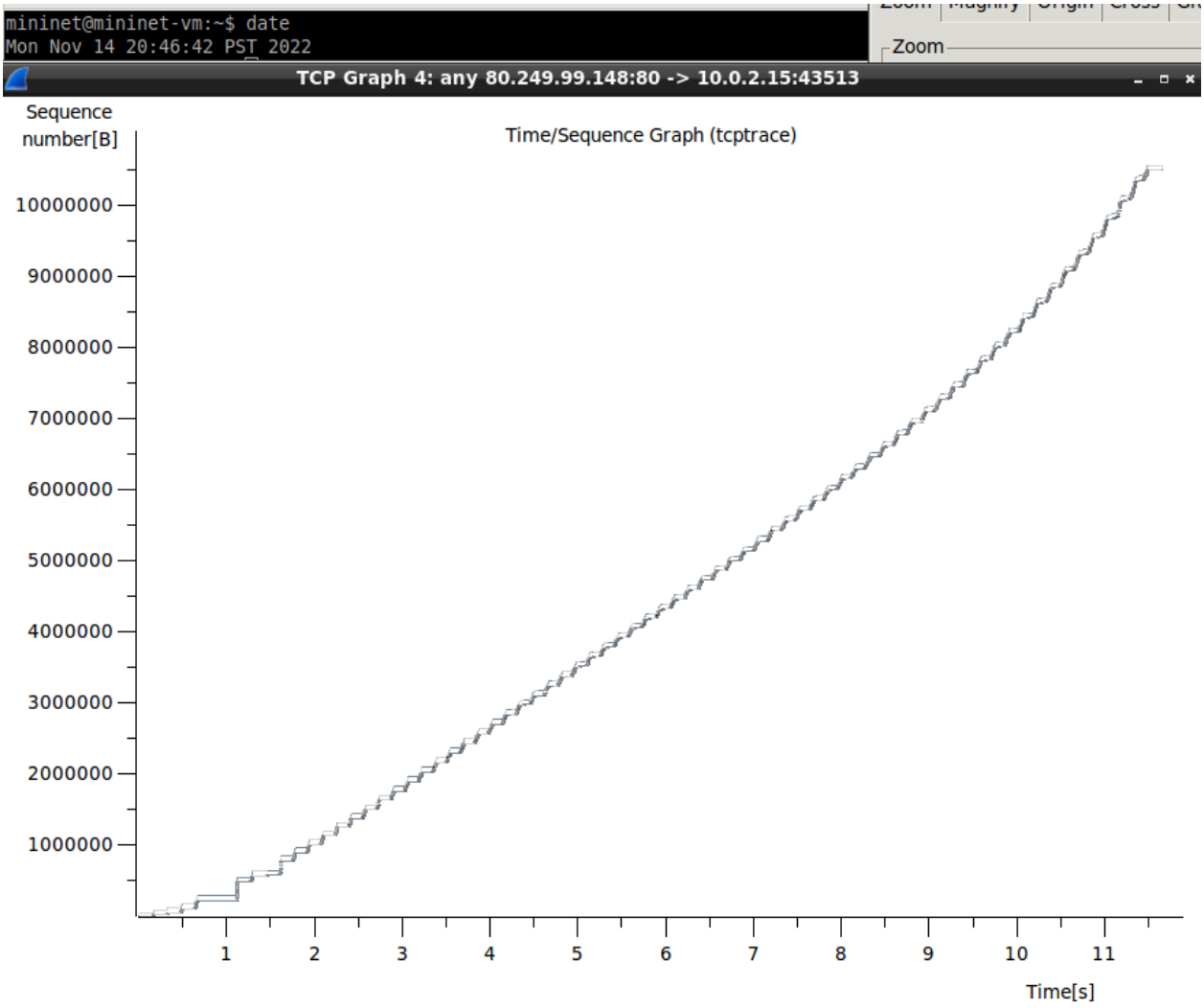
Select a TCP packet with len != 0 that has been transmitted by the server to your client:

a) **[2 pts]** What is the beginning SN of this TCP Segment?
Beginning sequence number is 2505361

b) **[2 pts]** How many bytes are in the entire TCP segment? (payload + header)

11,700 bytes

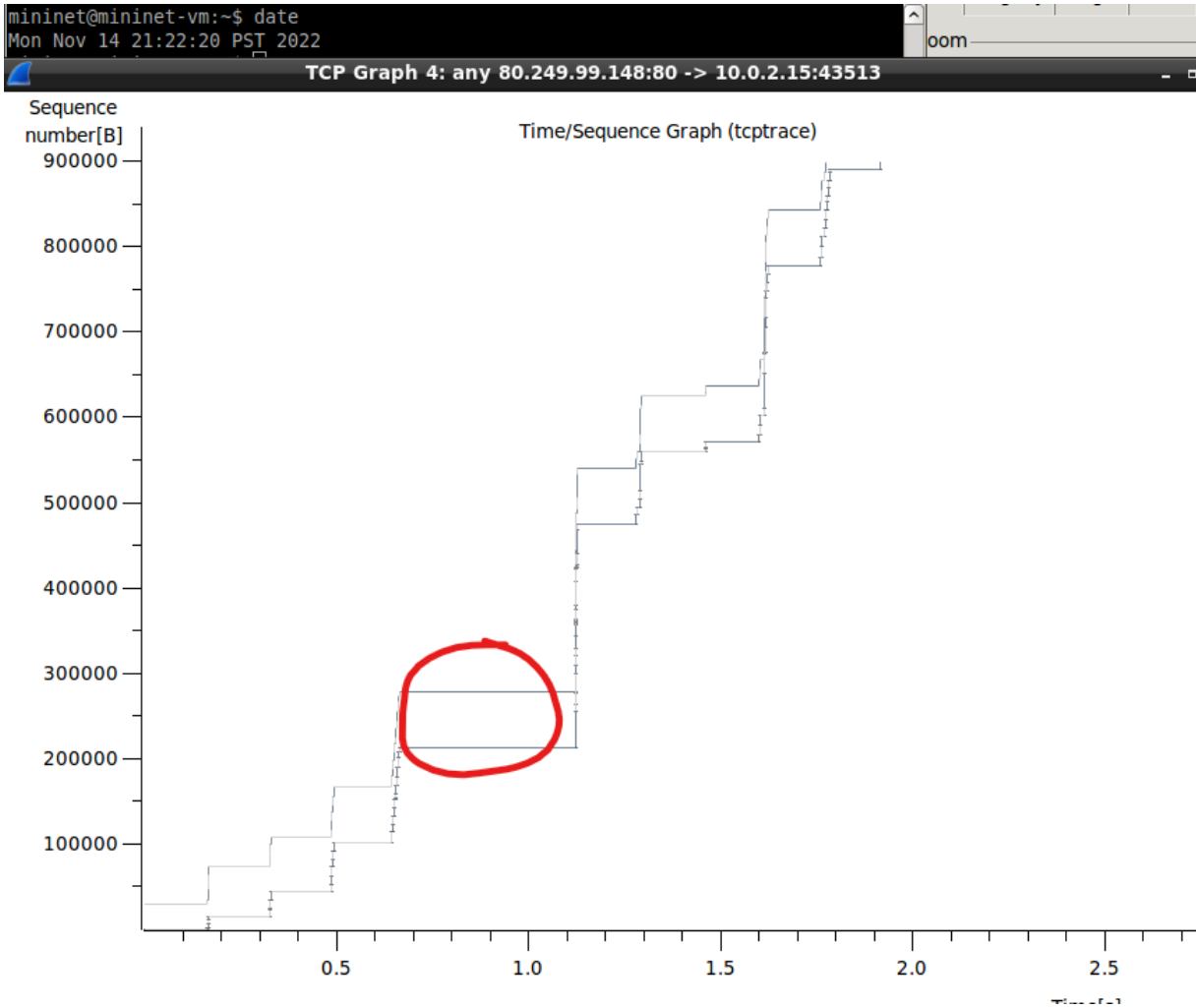
c) **[3 pts]** Create a tcptrace graph with this packet selected. Take a screenshot of the graph.



i) **[1 pt]** Explain what is displayed on the x and y-axis in the graph.

The x-axis displays time in seconds and the y-axis displays the sequence numbers.

ii) **[2 pts]** Highlight a period of slow start.



In the next section, we will be simulating loss on an interface using the command ***tc qdisc***. When the command is first used, you must use ***add dev*** for the interface being changed. It only needs to be set on the sender's side. After adding the device, use ***change dev*** to set the loss rate.

Follow the commands below to simulate loss on `eth0`:

- `sudo tc qdisc add dev eth0 root netem loss 0%`
- Change loss to 100%
  `sudo tc qdisc change dev eth0 root netem loss 100%`
- Change loss back to 0%
  `sudo tc qdisc change dev eth0 root netem loss 0%`

Read through this paragraph before starting the next step: First start Wireshark, then open 2 terminals and have these commands typed and ready before you begin: ● In one terminal, download the 10MB.zip file again.

- While the download is in progress, change loss to 100%. After a few seconds, change loss to 0%.

Note: We can also use the `--limit-rate` argument with wget to increase the download time, in case it is hard to type the commands during the download.

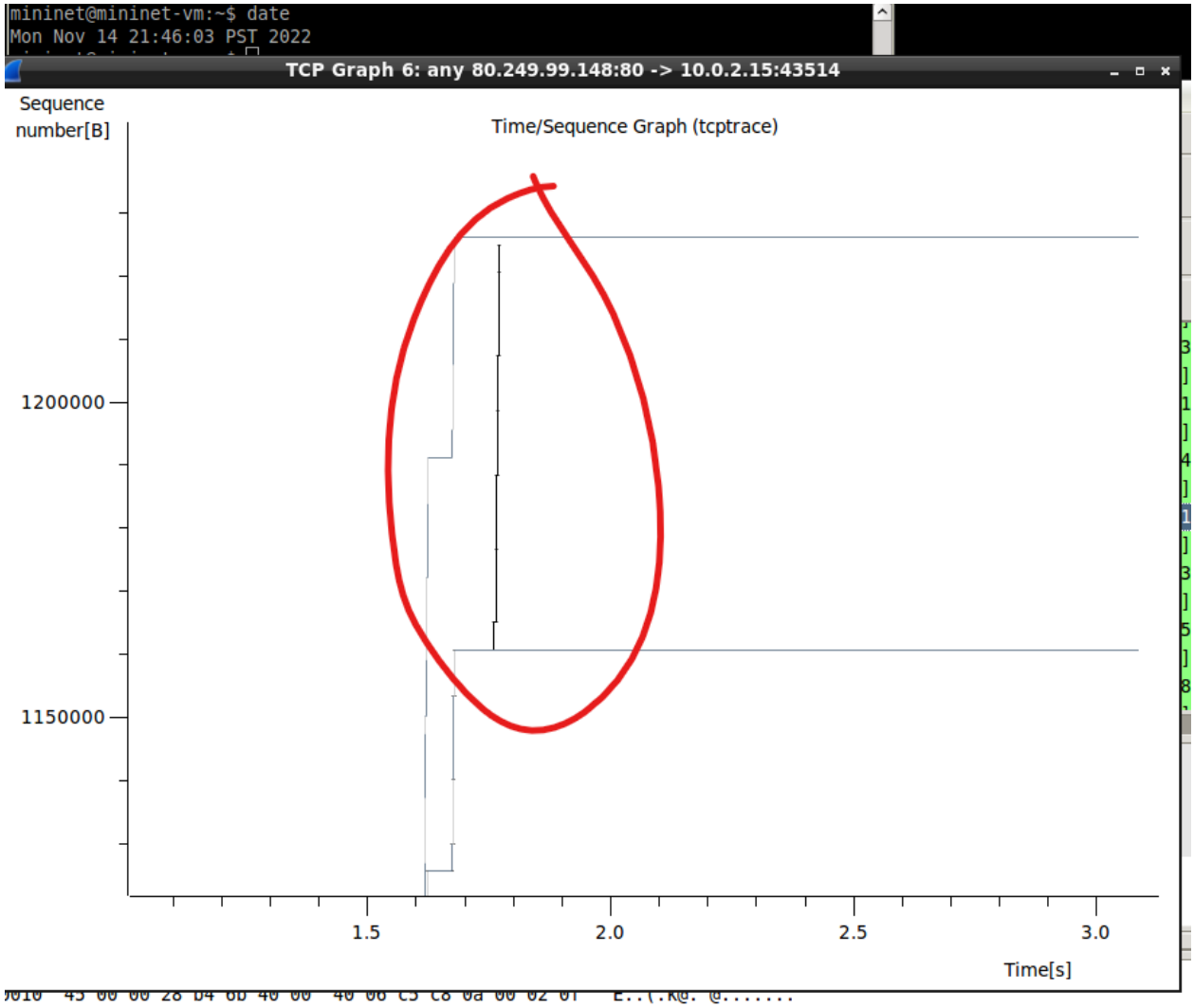
3) **[2 pts]** Do you expect that any two students in class would have exactly the same graph? Why or why not? What would be different?

I don't expect any two students in class to have the same graph because the reconnection speed when changing loss to 100% and then 0% would greatly differ, thus making the TCP graphs differ as well.

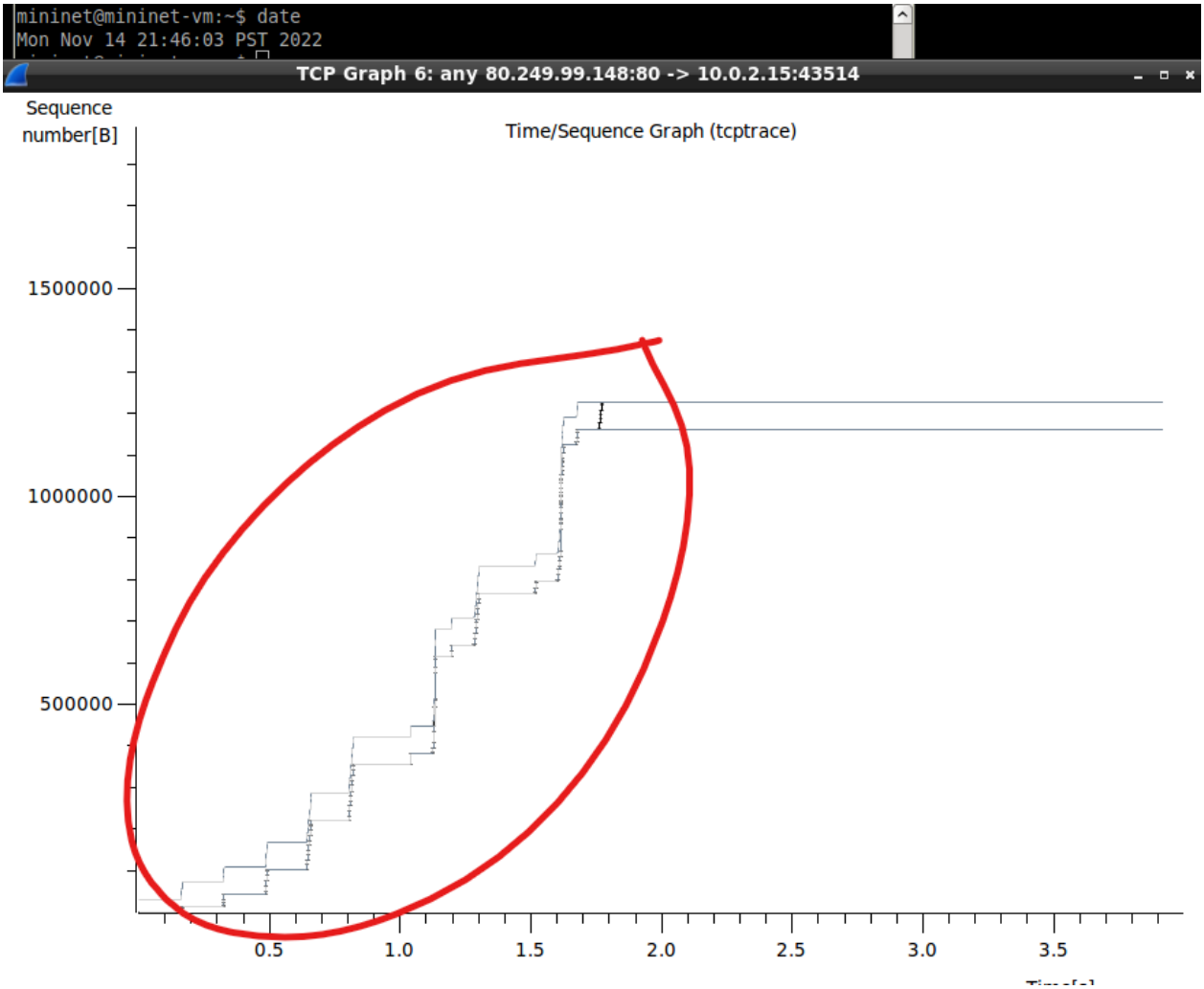
4) **[18 pts]** You might notice there is a list of TCP packets with len != 0 that have the server address as the source address and your computer's address as the destination address. Find a packet from the list and create a tcptrace graph with this packet selected.

a) **[2 pts]** At what time does packet loss begin? Attach a screenshot and highlight the region where 100% loss occurs in the graph.

Packet loss begins at about 1.5 seconds.



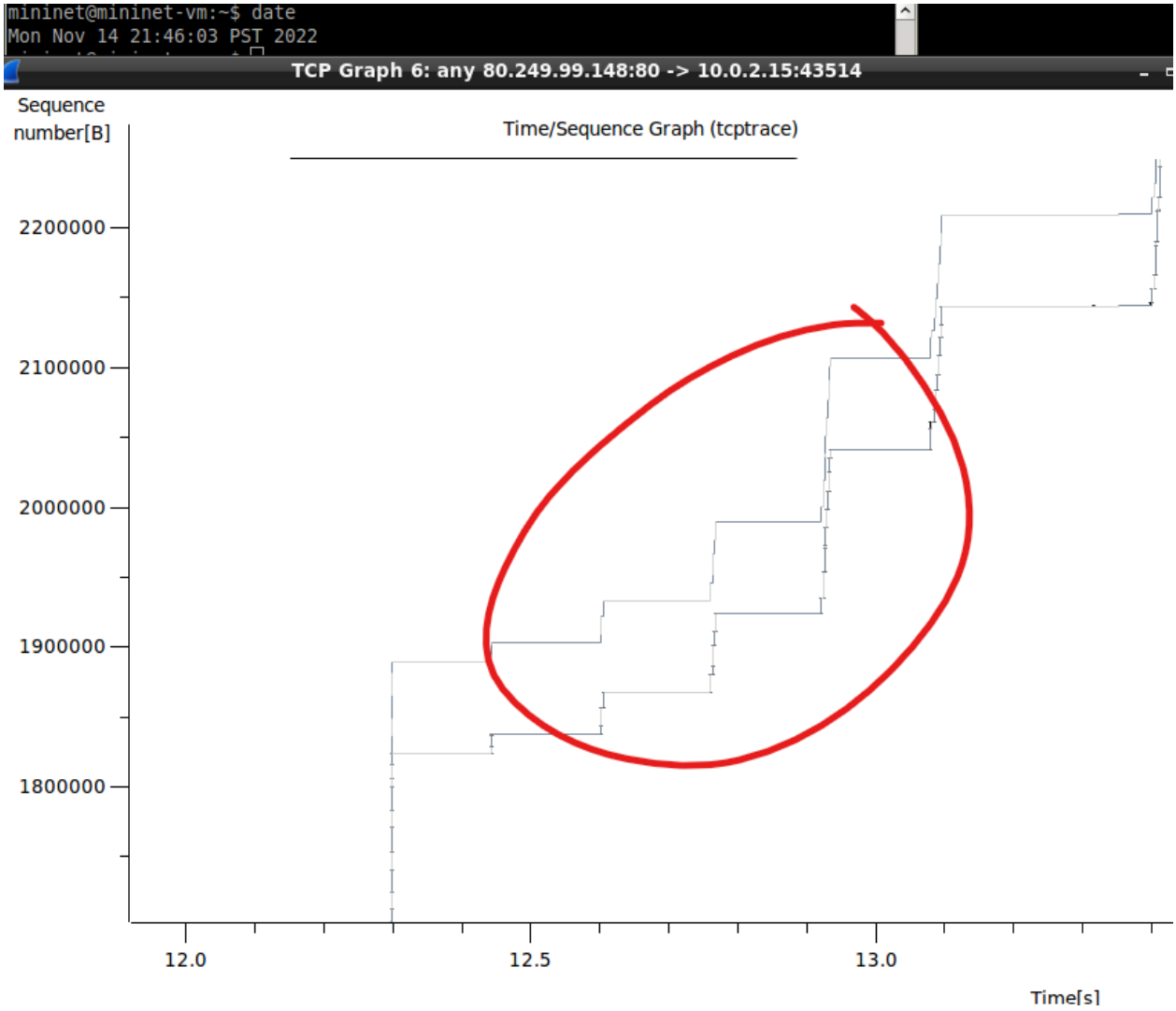
b) **[2 pts]** Attach a screenshot and highlight the region where slow start occurs at the beginning of the transfer.



c) **[2 pts]** Is the period immediately after packet loss, slow start as well? Explain and prove the same in the screenshots.

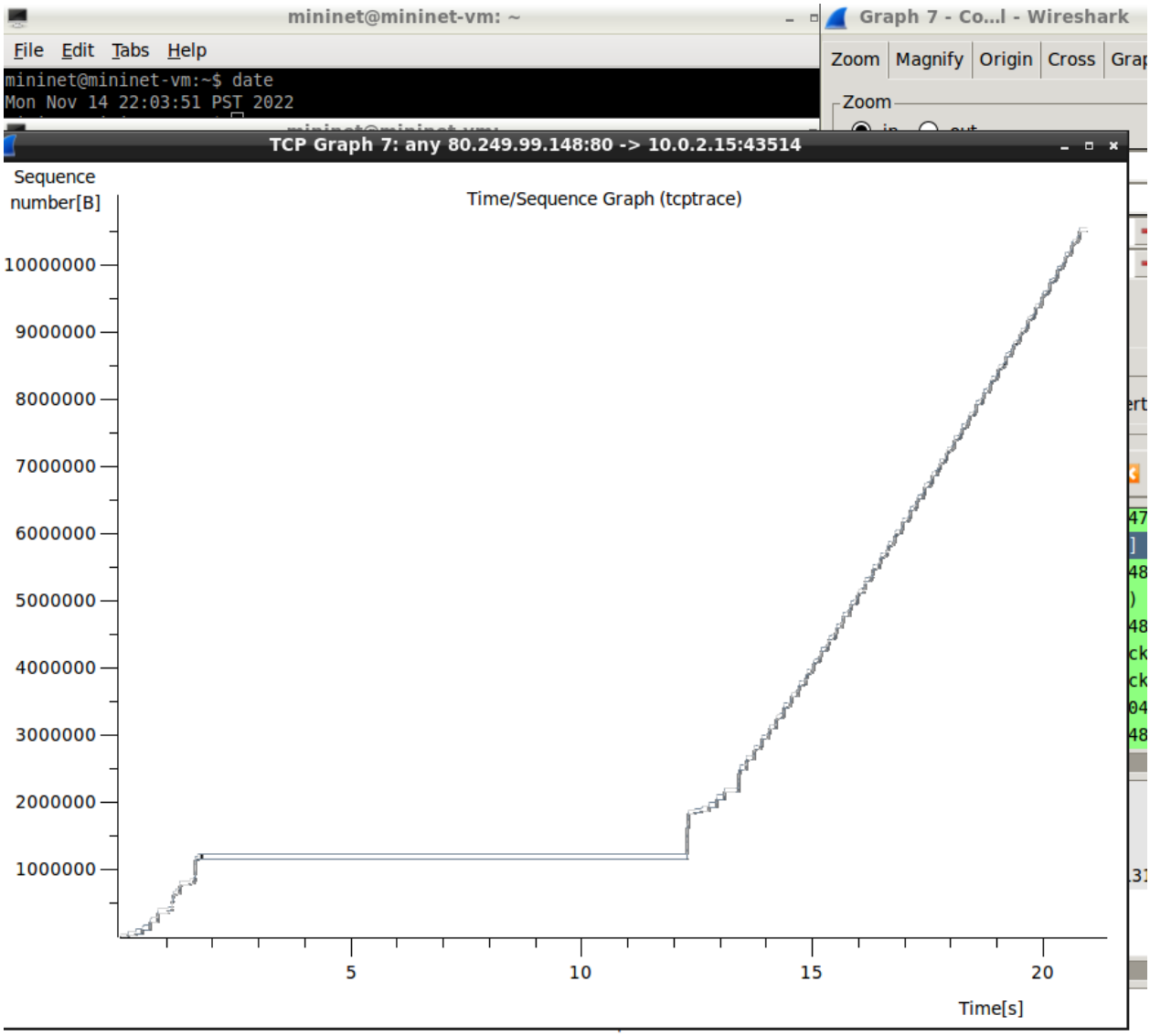
The period after packet loss is slow start as well, since the graph's slope increases exponentially.

d) **[2 pts]** Attach a screenshot and highlight the start of the region where the transmission restarted without losses.

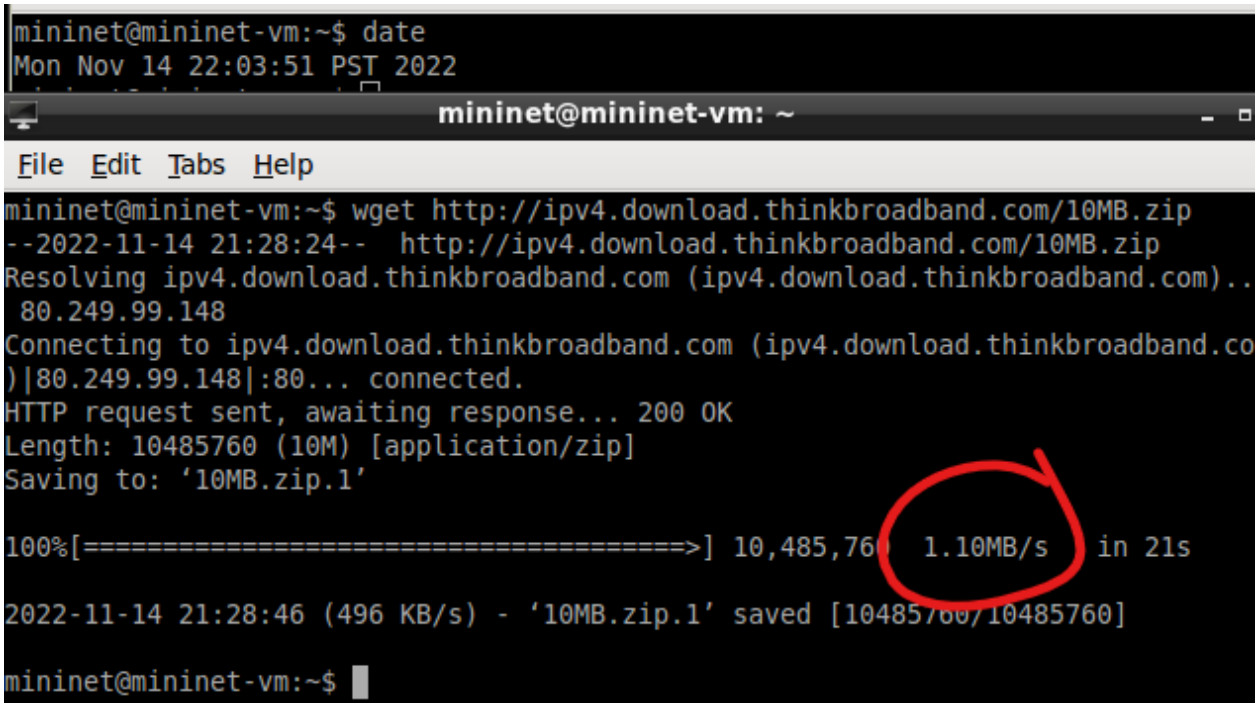


e) **[5 pts]** What is the average throughput as seen by the receiver WITH and WITHOUT packet loss? Show your calculations. Explain why there is a difference in the average throughput. Calculate average throughput from the tcptrace graph and compare your results from the wget output. Report values from both methods. Provide screenshots of TCP tracegraph and the wget (highlight the throughput in the wget).

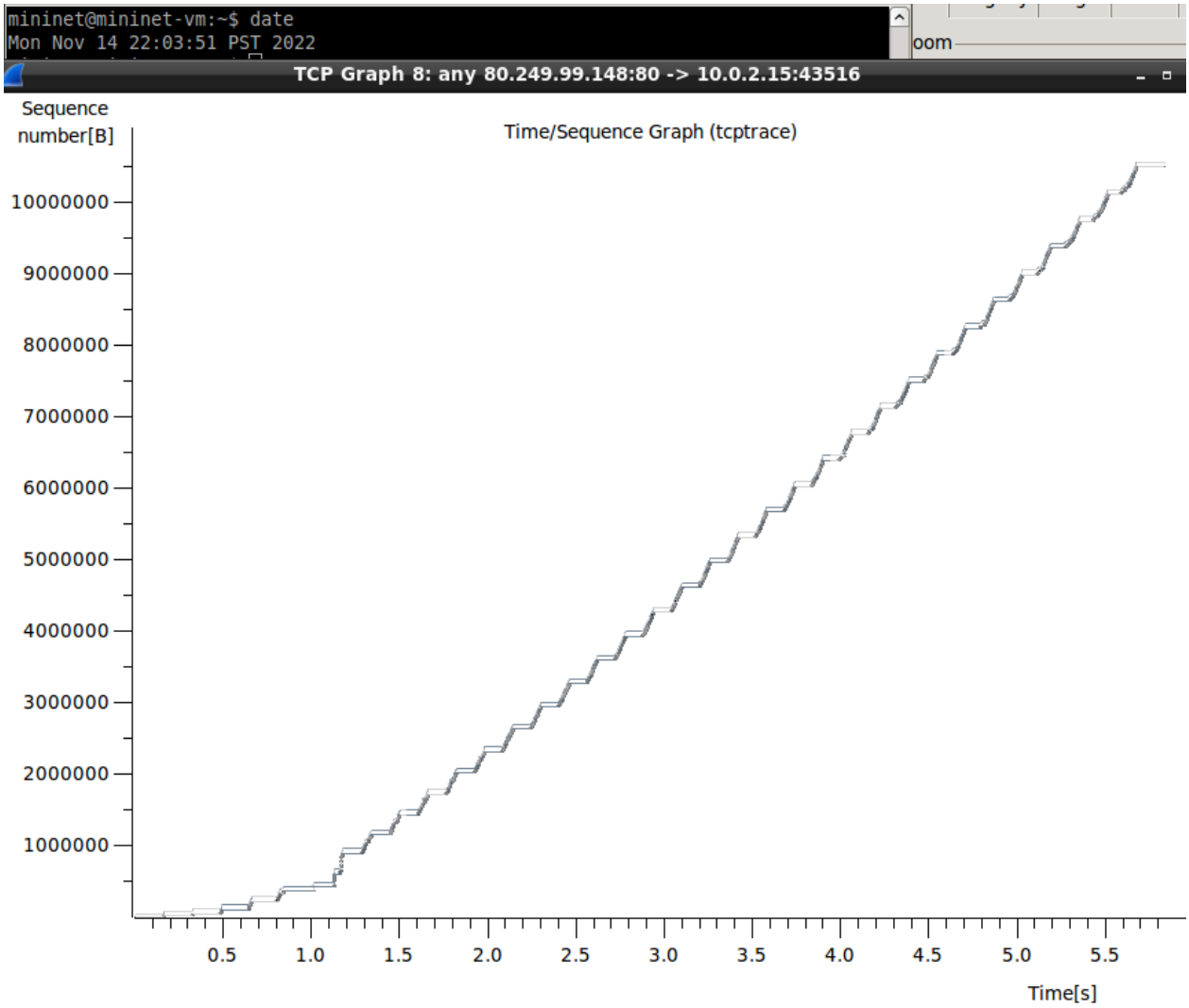
Throughput with packet loss tcptrace graph.



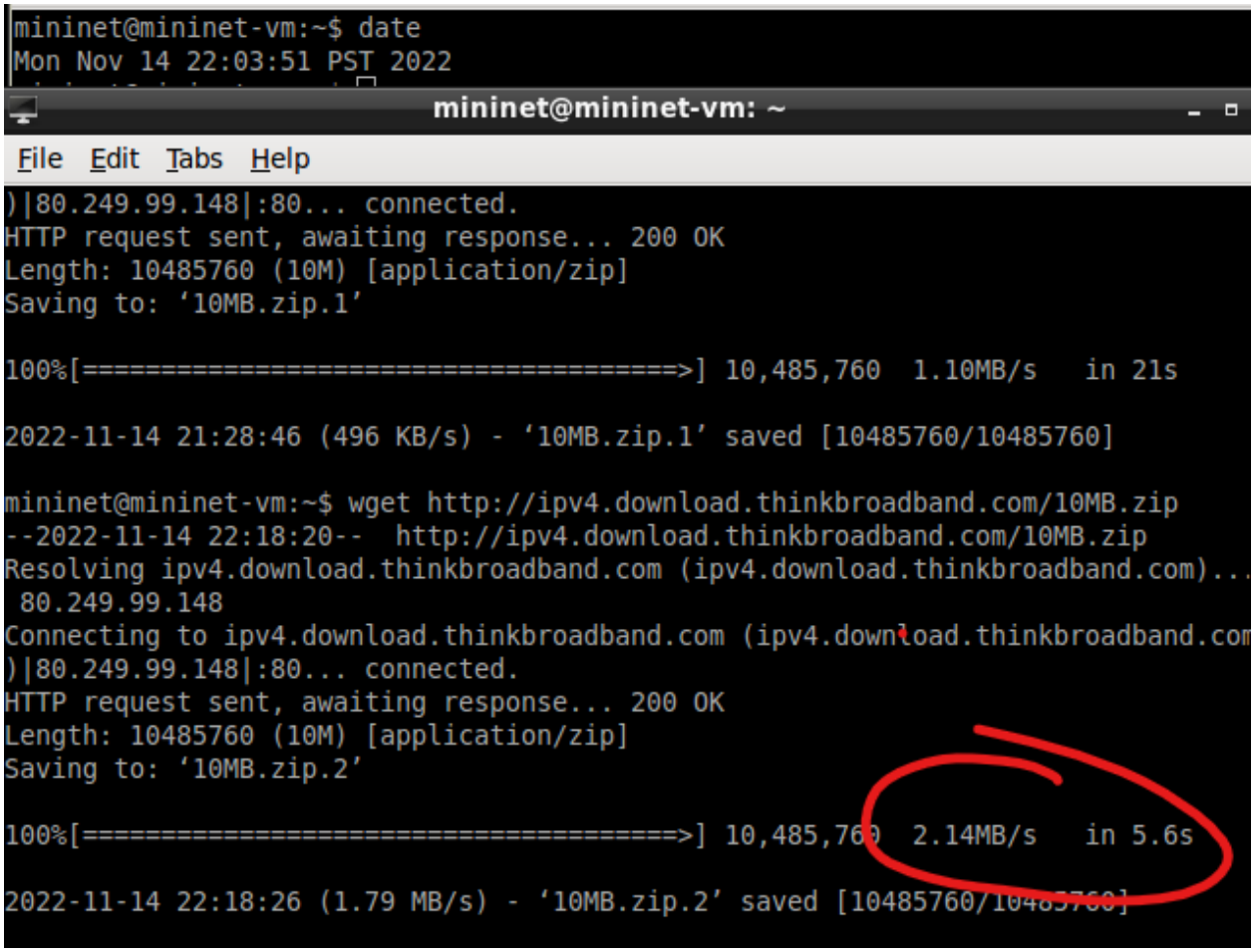
WGET with packet loss throughput



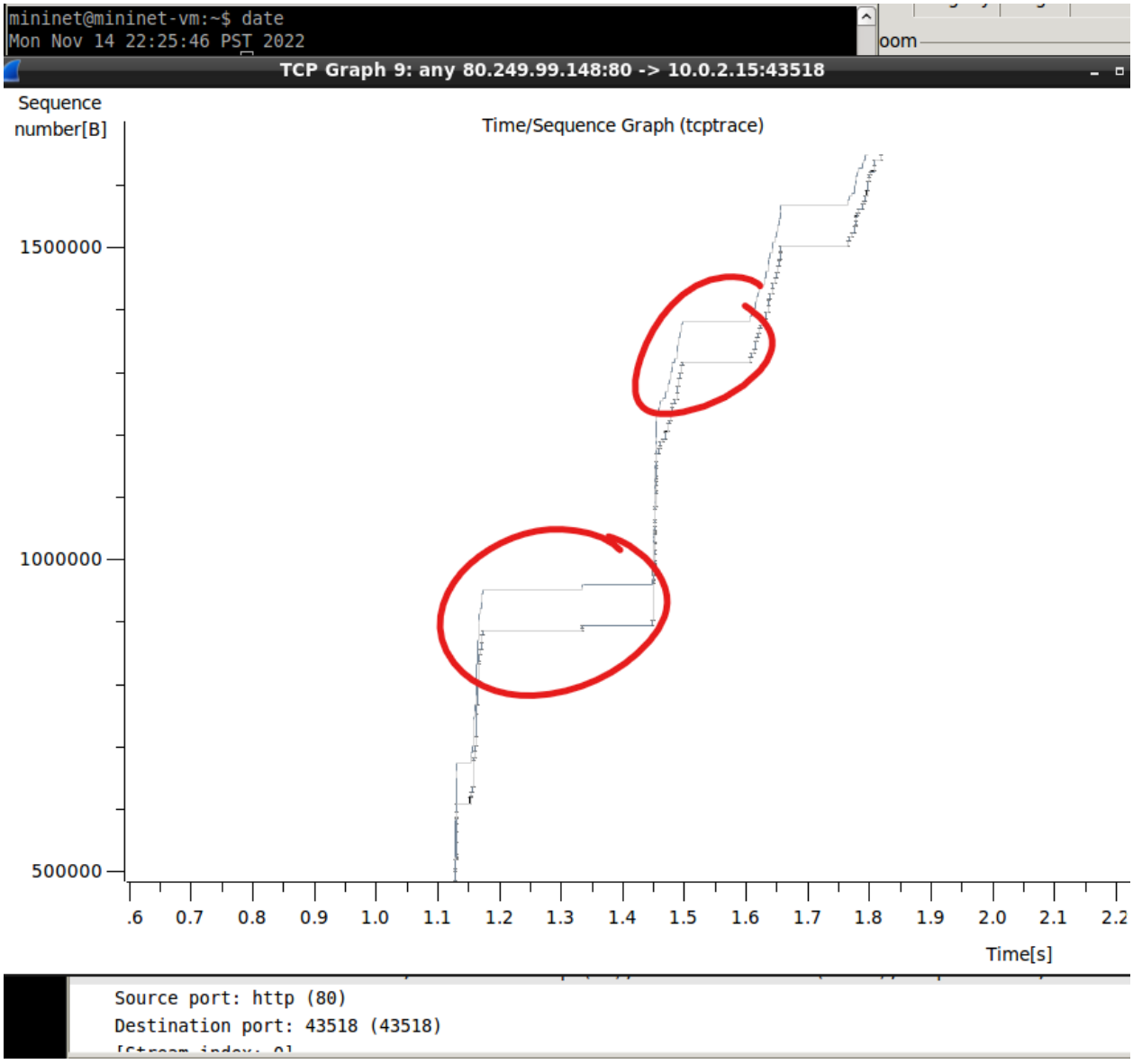
TCP tracegraph without packet loss



Wget Throughput without packet loss



f) **[5 pts]** Retry the experiment, this time with loss 25%, and attach a screenshot of the trace and highlight the periods of loss. (Note: you do not need to complete a) through e).)



5) **[6 pts]** Assume UDP was used for this packet transfer.

a) **[2 pts]** How do you expect the average end-to-end delay to be affected WITHOUT packet loss in comparison with TCP?

I think with UDP the average end to end delay would be lower since UDP establishes a transmission of data before an agreement is provided by the receiver.

b) **[2 pts]** What change would you expect to observe in the average throughput seen by the receiver WITHOUT packet loss if UDP is used for downloading the 10MB file. How would that compare to the throughput in the TCP scenario?

The average throughput would increase if UDP was used but there

might be a lot of packet loss. TCP guarantees the retransmission of lost packets.

c) **[2 pts]** How does UDP behave if there is a packet loss during the file transfer?

If there is packet loss, UDP discards the packet as soon an error occurs. It cannot retransmit lost packets.

Deliverables

1. **cruzid-lab3.pdf** : PDF containing required solutions and screenshots for lab3.
2. **cruzid-lab3_topo.py** : Your network topology for the router.
3. **cruzid-lab3_controller.py** : Your remote controller code for the router.
4. **README.txt** : A readme file explaining your submission.