Elijah Cordova
ejcordov
1425119

Lab 2 CMPE 150

Prelab

1. 200 : OK : A request was sucessfully fufilled.

201 : CREATED : After a POST or PUT request, a new entry in the server was sucessfully created.

404 : Not found : The server could not find the entry requested.

400 : Bad Request: The server could not process your request as it was formatted wrong / could not be read

301 : moved : The entry requested has moved to a new URI.

2. GET: A GET method requests and retrieves information from the specified URI. This method only retrieves, and does not create. The get method can return a response body.

HEAD: Similar to GET, it requests and retrieves information, and only retrieves, but does not return a response body.

POST: The POST method creates a new entry at the URI specified. It's up to the server for the method of creation, but usually the information needed within a POST request is sheathed within the request URI.

PUT: Similar to POST: it creates a new entry at the URI specified if the resource doesn't exist at the location, but if a resource already exists the method will modify already existing data.

DELETE: Delete specified entry identified by the request URI.

TRACE: The server echoes back a request. This method allows you to see what the server sees after going through potential intermediary servers.

OPTIONS: Returns the HTTP methods that the specified URL requested supports.

CONNECT: Creates a two way channel with the requested resource.
PATCH: Changes the specified resource partially.

3. wget -S -spider

return status : 200 OK
Last-Modified Fri,09 Aug 2013

4. The telnet server plays an animation of Star Wars. More generally, the telnet server is sending text via the the telnet protocol invoked by the command.

5. A DNS resource record is an implemented data type in the Domain Name System. DNS records gives DNS zones instructions on how to handle requests.

using nslookup -q=mx ucsc.edu

The mail exchanger records point to _.google.com. These servers accept emails on behalf of the ucsc.edu domain. In this case, all ucsc email is through gmail.

6. The command involves looking to find the ns records of the current directory which is invoked by (.) . So when this command is run, it returns the highest DNS level, or root servers. It lists all of the root servers : f.root-servers.net, m.rootservers.net, etc.

7. You can run multiple applications by identifying each connection by a 4-tuple. This 4-tuple Contains source / destination ports and source / destination IP addresses.

8. The whole point of the window mechanism is control. You can send or recieve a specific amount of data, giving you control over the buffer of information of data you wish to send or recieve. This makes it easy to tell whether or not something got garbled or sent incorrectly.

9. The MTU is the maximum transmission unit, meaning the largest possible unit of data you can transmit over a network in a single transaction. Unlike the MSS, a packet over the size of MTU will fragment.

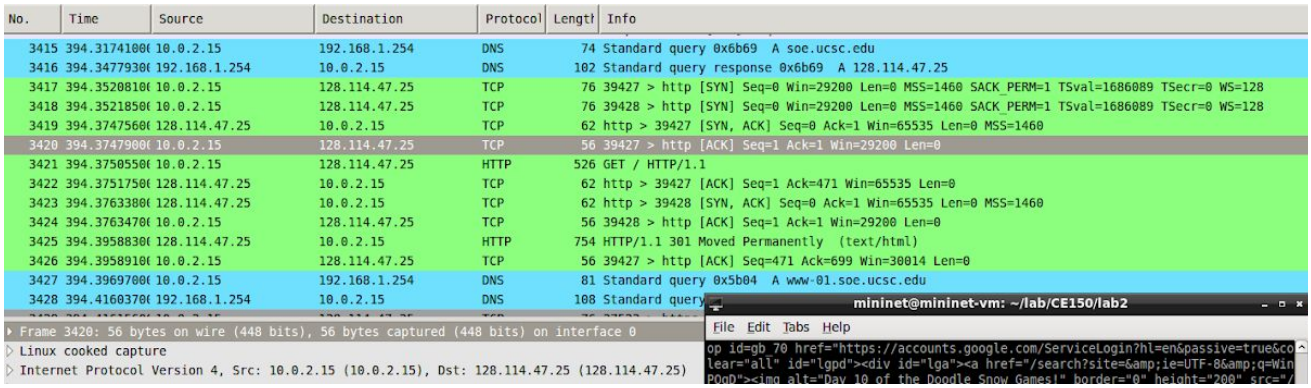
Lab2:
1.
The computer used a GET method to make this request.
URI: https://www.ucsc.edu/

[The packet is buried within an encrypted layer (https) so wireshark can't display the appropriate information.]

2.
The server returns status code 200 ok.
It returns a xml file which is the webpage.

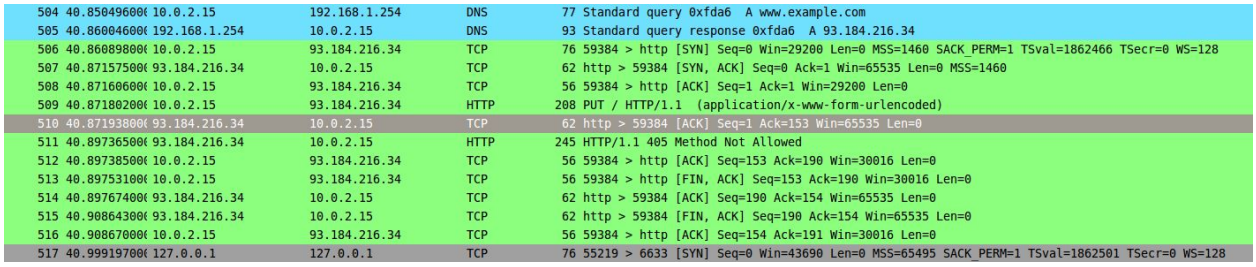
[The packet is buried within an encrypted layer (https) so wireshark can't display the appropriate information.]

3.



What's different is that the uri contains http instead of https.
The webpage redirects to the equivalent https://,so the response instead of a simple 200 is 301, or Moved Permanently.

4.

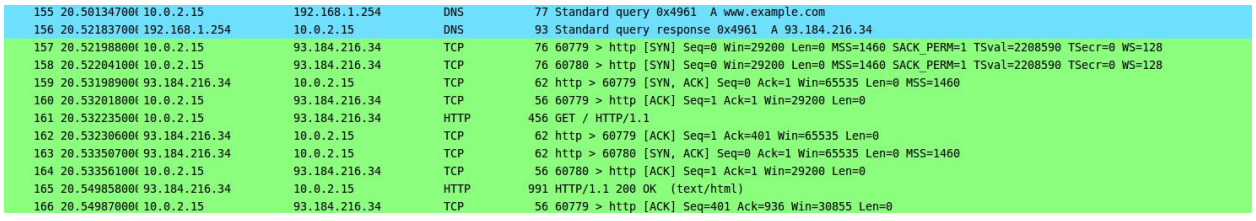


I sent a put request to www.example.com using the terminal command

curl www.example.com -X PUT -d "stuff"

this sends a PUT request with a data field "stuff". This results in a 405 method not allowed.

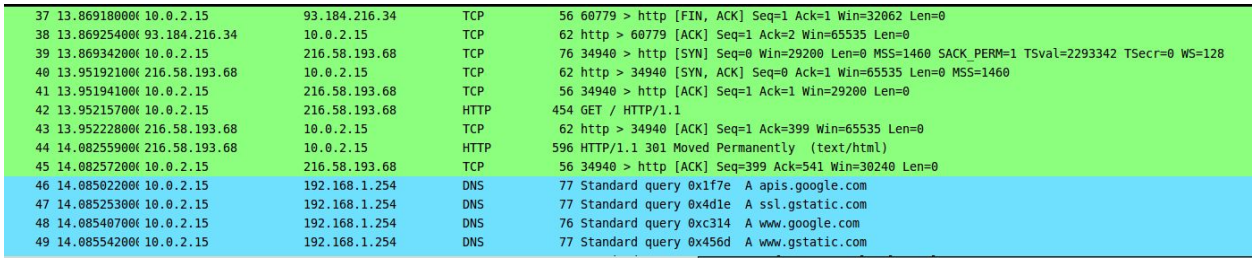
5.



As you can see by the screenshot, the DNS performs a standard query to retrieve the IP address of www.example.com. Example.com then sends a standard query response, then the

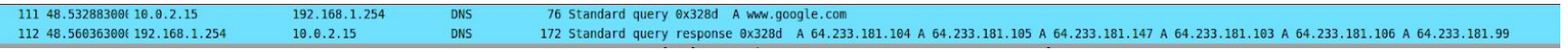
TCP connection starts. The tcp connection then sends the HTTP1.1 file and http status 200 ok. What's being illustrated above is the TCP handshake protocol, as seen by the [ACK]'s.

6.



Unlike the above request, the http status sends 301 moved permanently. This is because the address uses http:// . After redirecting to the appropriate https:// equivalent uri it starts loading a standard query, and goes through the same process is above. HTTPS is encrypted, so wireshark can't show the appropriate information after this.

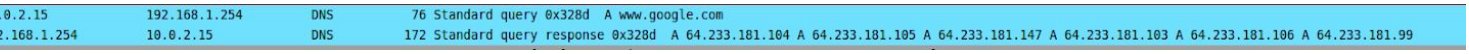
7. Use



nslookup -q=A www.google.com

The above command outputs 64.233.181.104
64.233.181.105
64.233.181.147
64.233.171.103
64.233.181.106
64.233.181.99
As seen in Wireshark as a DNS query response and in the terminal output.

8. It's recursive, as the request only has one rather than multiple
queries. The manpage for nslookup also says that recursive searches are enabled by default.



9.

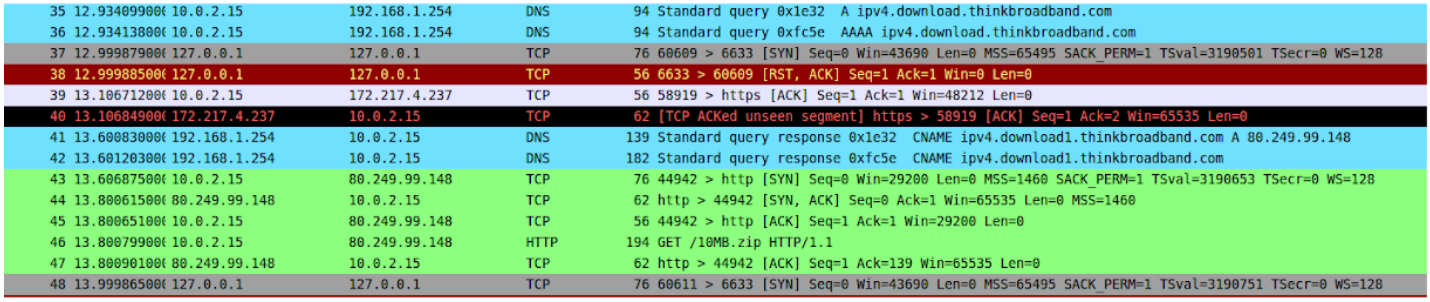
198.105.244.130
104.239.207.44

10. Using

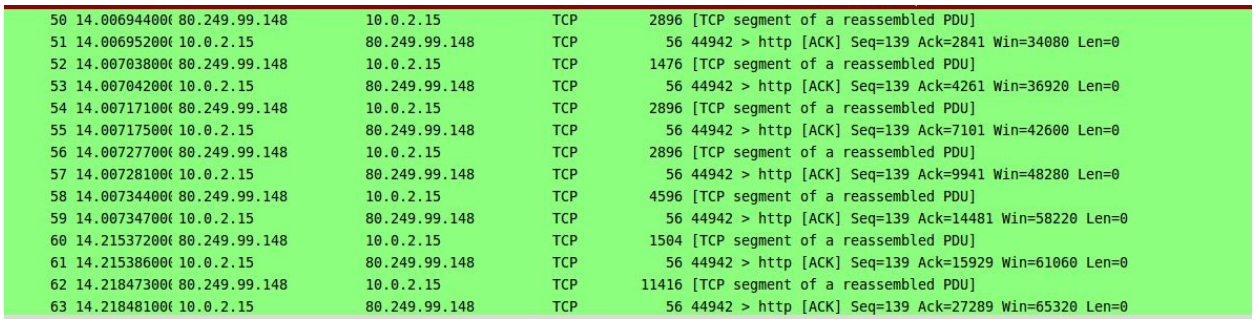
nslookup -q=a www.ucsc.edu

, we can see that the authoritative name is wcms-ucsc.aws-wcms.ucsc.edu. Connecting to the ip address queried by above verifies this.

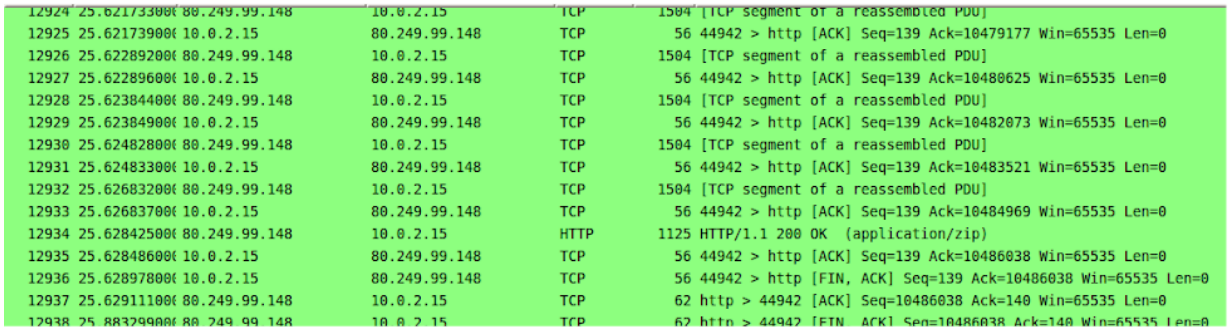
11.



35	12.93409900(	10.0.2.15	192.168.1.254	DNS	94 Standard query 0x1e32 A ipv4.download.thinkbroadband.com
36	12.93413800(	10.0.2.15	192.168.1.254	DNS	94 Standard query 0xfc5e AAAA ipv4.download.thinkbroadband.com
37	12.99987900(	127.0.0.1	127.0.0.1	TCP	76 60609 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3190501 TSecr=0 WS=128
38	12.99988500(	127.0.0.1	127.0.0.1	TCP	56 6633 > 60609 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
39	13.10671200(	10.0.2.15	172.217.4.237	TCP	56 58919 > https [ACK] Seq=1 Ack=1 Win=48212 Len=0
40	13.10684900(	172.217.4.237	10.0.2.15	TCP	62 [TCP ACKed unseen segment] https > 58919 [ACK] Seq=1 Ack=2 Win=65535 Len=0
41	13.60083000(	192.168.1.254	10.0.2.15	DNS	139 Standard query response 0x1e32 CNAME ipv4.download1.thinkbroadband.com A 80.249.99.148
42	13.60120300(	192.168.1.254	10.0.2.15	DNS	182 Standard query response 0xfc5e CNAME ipv4.download1.thinkbroadband.com
43	13.60687500(	10.0.2.15	80.249.99.148	TCP	76 44942 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3190653 TSecr=0 WS=128
44	13.80061500(	80.249.99.148	10.0.2.15	TCP	62 http > 44942 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
45	13.80065100(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0
46	13.80079900(	10.0.2.15	80.249.99.148	HTTP	194 GET /10MB.zip HTTP/1.1
47	13.80090100(	80.249.99.148	10.0.2.15	TCP	62 http > 44942 [ACK] Seq=1 Ack=139 Win=65535 Len=0
48	13.99986500(	127.0.0.1	127.0.0.1	TCP	76 60611 > 6633 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3190751 TSecr=0 WS=128



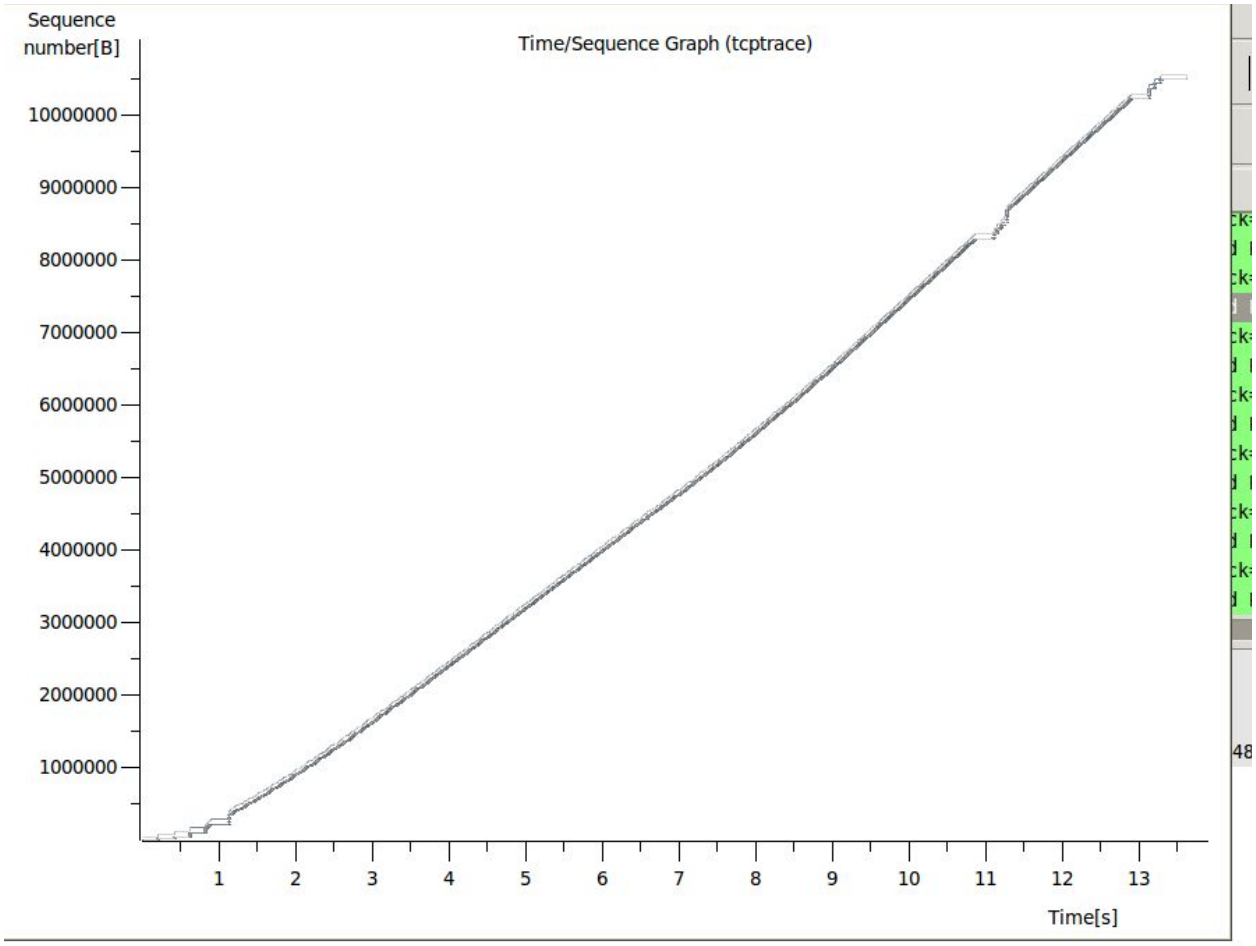
50	14.00694400(	80.249.99.148	10.0.2.15	TCP	2896 [TCP segment of a reassembled PDU]
51	14.00695200(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=2841 Win=34080 Len=0
52	14.00703800(	80.249.99.148	10.0.2.15	TCP	1476 [TCP segment of a reassembled PDU]
53	14.00704200(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=4261 Win=36920 Len=0
54	14.00717100(	80.249.99.148	10.0.2.15	TCP	2896 [TCP segment of a reassembled PDU]
55	14.00717500(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=7101 Win=42600 Len=0
56	14.00727700(	80.249.99.148	10.0.2.15	TCP	2896 [TCP segment of a reassembled PDU]
57	14.00728100(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=9941 Win=48280 Len=0
58	14.00734400(	80.249.99.148	10.0.2.15	TCP	4596 [TCP segment of a reassembled PDU]
59	14.00734700(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=14481 Win=58220 Len=0
60	14.21537200(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
61	14.21538600(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=15929 Win=61060 Len=0
62	14.21847300(	80.249.99.148	10.0.2.15	TCP	11416 [TCP segment of a reassembled PDU]
63	14.21848100(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=27289 Win=65320 Len=0



12924	25.62173300(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
12925	25.62173900(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10479177 Win=65535 Len=0
12926	25.62289200(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
12927	25.62289600(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10480625 Win=65535 Len=0
12928	25.62384400(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
12929	25.62384900(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10482073 Win=65535 Len=0
12930	25.62482800(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
12931	25.62483300(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10483521 Win=65535 Len=0
12932	25.62683200(	80.249.99.148	10.0.2.15	TCP	1504 [TCP segment of a reassembled PDU]
12933	25.62683700(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10484969 Win=65535 Len=0
12934	25.62842500(	80.249.99.148	10.0.2.15	HTTP	1125 HTTP/1.1 200 OK (application/zip)
12935	25.62848600(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [ACK] Seq=139 Ack=10486038 Win=65535 Len=0
12936	25.62897800(	10.0.2.15	80.249.99.148	TCP	56 44942 > http [FIN, ACK] Seq=139 Ack=10486038 Win=65535 Len=0
12937	25.62911100(	80.249.99.148	10.0.2.15	TCP	62 http > 44942 [ACK] Seq=10486038 Ack=140 Win=65535 Len=0
12938	25.88329900(	80.249.99.148	10.0.2.15	TCP	62 http > 44942 [FIN, ACK] Seq=10486038 Ack=140 Win=65535 Len=0

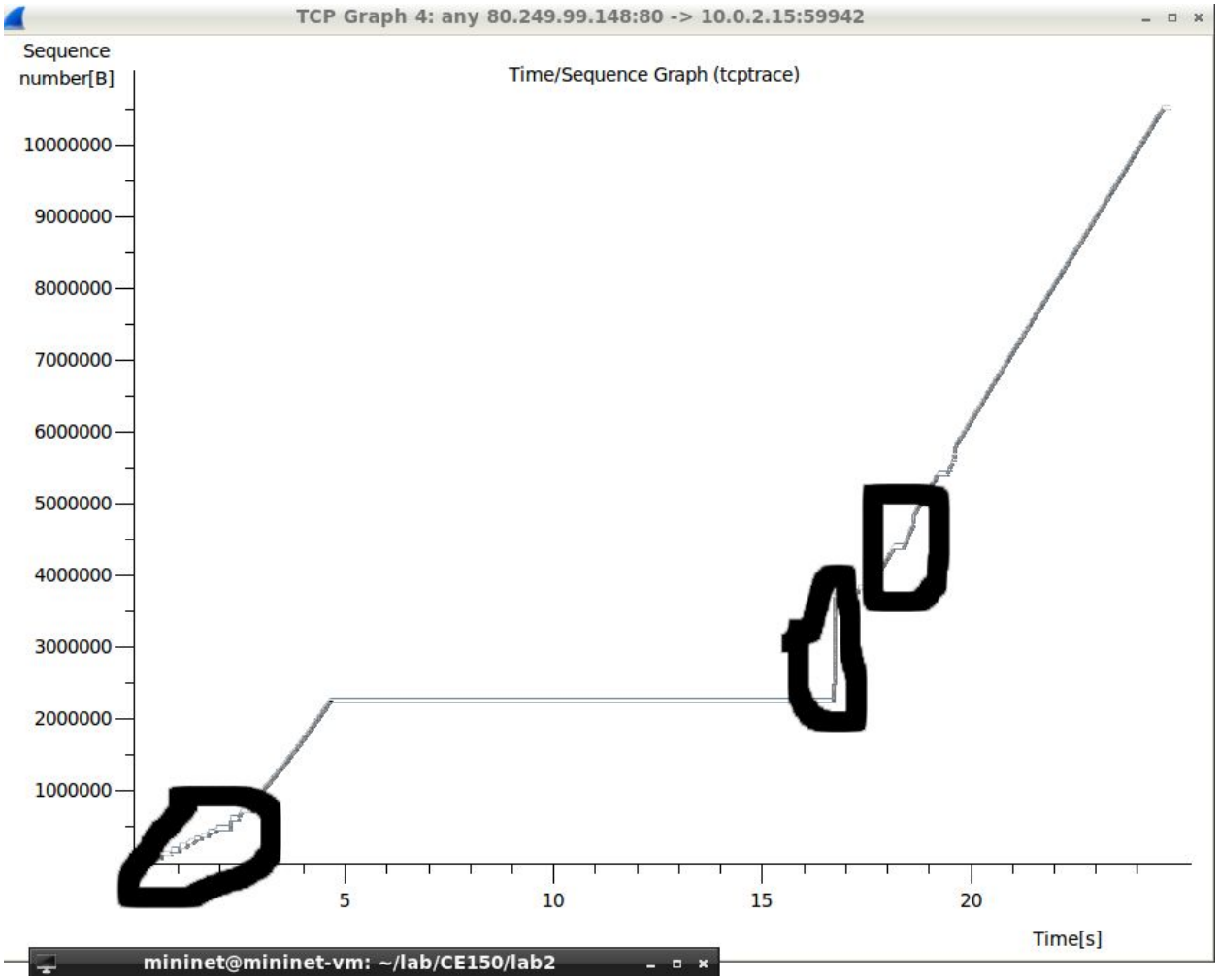
The first syn before the syn-ack (initial offering) = 29200.
The syn-ack (or the response to the intial offering) = 65535.

12.



The sequence number increases as time goes on, which means that, based on the slope of the graph, the download speed was consistent. Sequence number increases by 1 for every byte sent.

13.



This graph is similar to the above graph, but instead all connections are cut off and reconnected in the middle.
The completely flat part is a result of the command

sudo tc qdisc add dev eth0 root netem loss 100 %

Which cuts off all download speed. The commands

sudo tc qdisc del dev eth2 root netem delay 100ms 10ms 100 %
sudo tc qdisc add dev eth0 root netem loss 0 %

is where the steep upward growth after the 100% loss is from. The slow starts are the areas where the slope of the graph is less steep compared to the rest of the graph. After the slow start we have congestion avoidance.