Lab 5 – Understanding Transport and Network Layer using Wireshark

Sriram Radhakrishna PES1UG20CS435 4H

Objective

In this lab, you will continue to use Wireshark, you will explore the transport and network layers. You will examine various UDP, TCP and ICMP transmissions. Write a report, to show you have executed the lab procedures. In this report, also answer any questions that are interleaved among the procedures. Feel free to also include questions, thoughts, and any interesting stuff you observed.

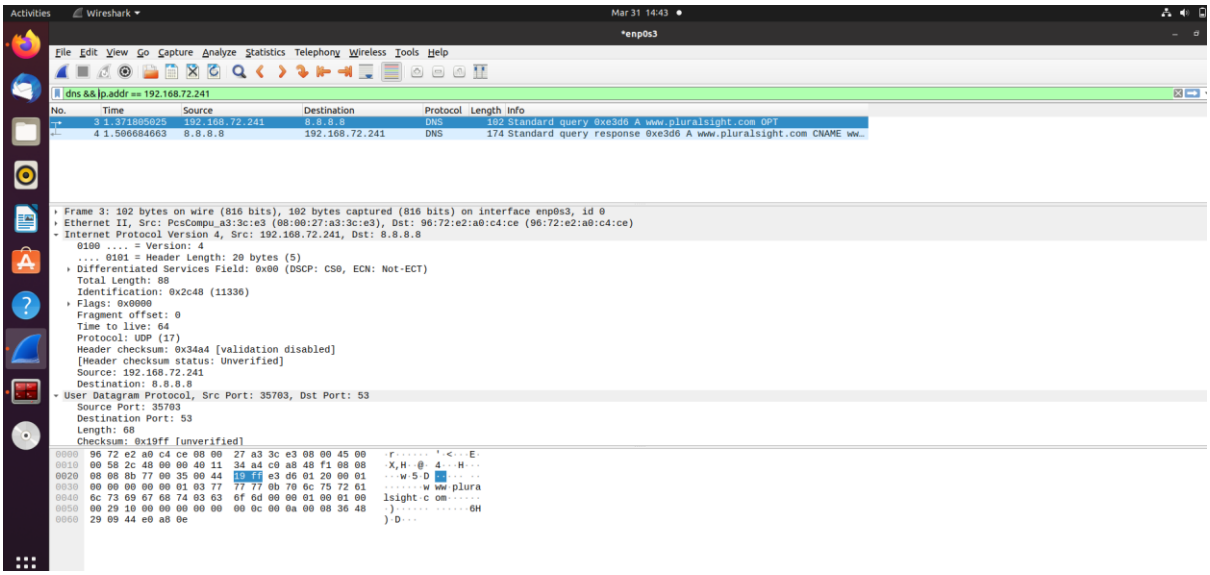
Note: Take screenshots wherever necessary.

Step 1: UDP and DNS

Let's start by examining a few UDP segments. UDP is a streamlined, no-frills transport protocol. All state information is conveyed in each individual UDP segment. In Lab 4, we used dig to generate DNS traffic with the intent of examining the DNS protocol. In this lab, we will use dig to generate DNS traffic, but with the intent of examining the UDP protocol.

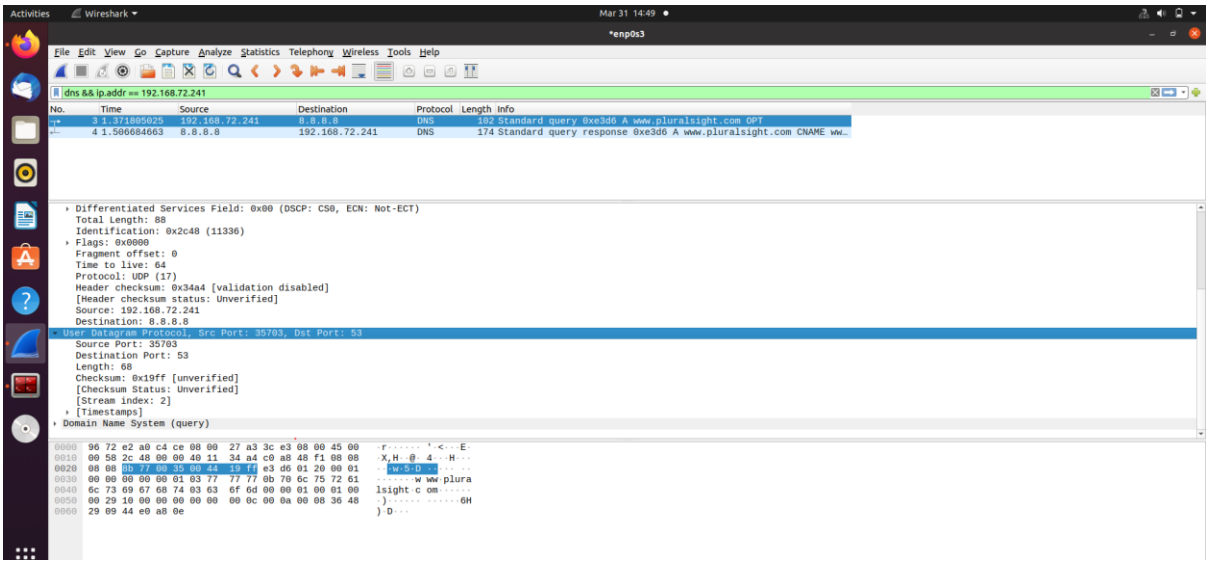
Procedures

1) Open Wireshark and set up our privacy filter so that you display only DNS traffic to or from your computer (Filter: **dns && ip.addr==<your IP address>**).
2) Use dig to generate a DNS query to lookup the domain name "**www.pluralsight.com**". Then, stop the capture.

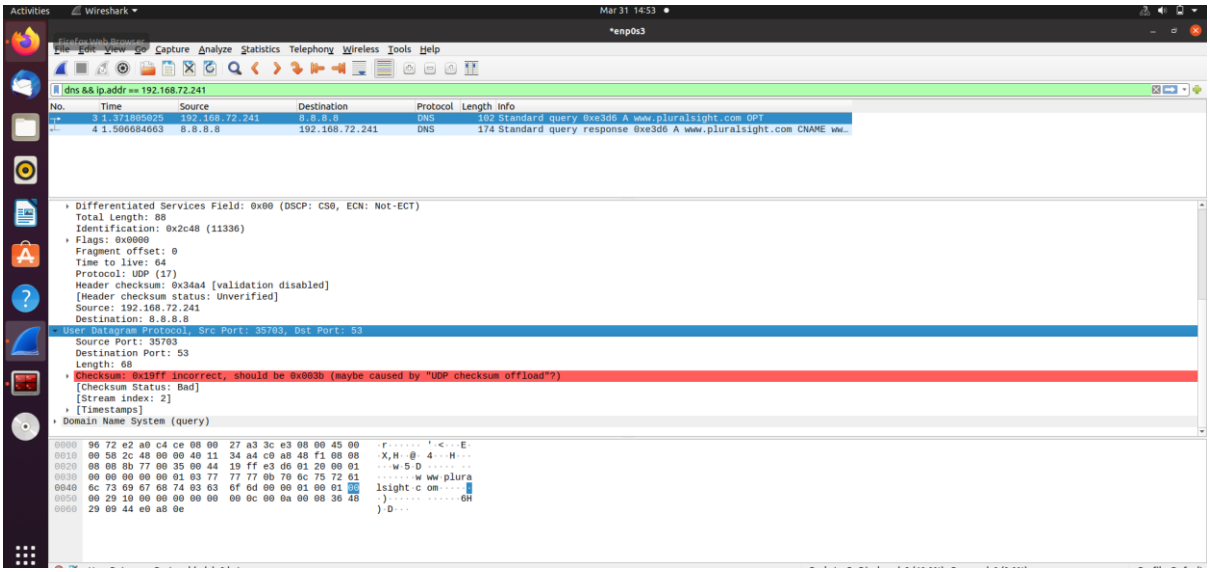


3) Before you look at the packets in Wireshark, think for a minute about what you expect to see as the UDP segment headers. What can you reasonably predict, and what could you figure out if you had some time and a calculator handy? Use your knowledge of UDP to inform your predictions.
   – **Source port, destination port, segment length.**
4) Take a look at the query packet on Wireshark. You'll see a bunch of bytes (70-75 bytes) listed as the actual packet contents in the bottom Wireshark window. The bytes at offsets up to number 33-34 are generated by the lower-level protocols. If you click on the "User Datagram Protocol" line in the packet details window, you'll see the

UDP contents get highlighted in the packet contents window. You will also see Wireshark interpret the header contents. Match up the bytes in the packet contents window with each field of the UDP header. Were your predictions correct? - **Yes**



5) Continue to examine the DNS request packet. Which fields does the UDP checksum cover? Wireshark probably shows the UDP checksum as “Validation Disabled”. Why is that?
– **Checksum field is marked as unverified. When verified, the checksum is incorrect due to UDP checksum offload.**



6) Save your capture file. Restrict the range of saved packets to only those in the DNS query.

Step 2: TCP

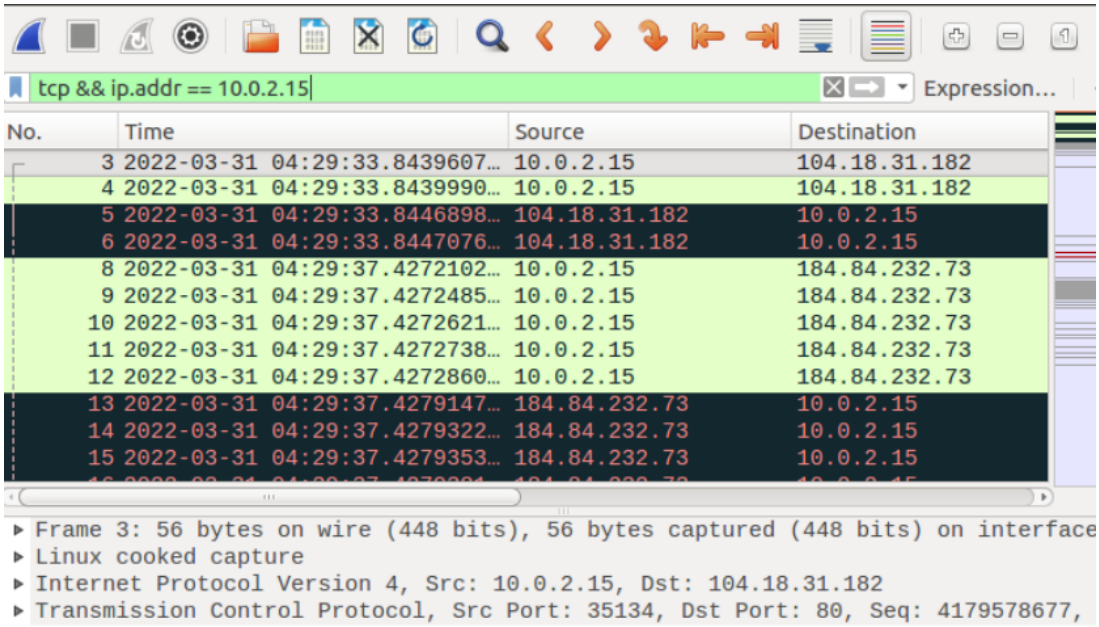
Now, let’s look at another transport protocol, TCP. We will use HTTP to invoke the sort of TCP behaviours we want to study -> I trust that you understand HTTP well enough by now.

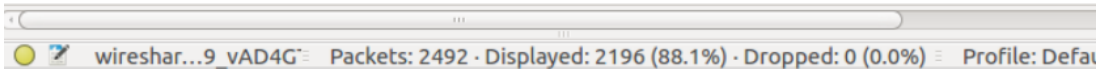
7) Download and save a copy of Geoffrey Chaucer's Canterbury Tales and Other Poems from the Project Gutenberg website1. Grab the Plain Text UTF-8 version: **http://www.gutenberg.org/ebooks/2383.txt.utf-8**

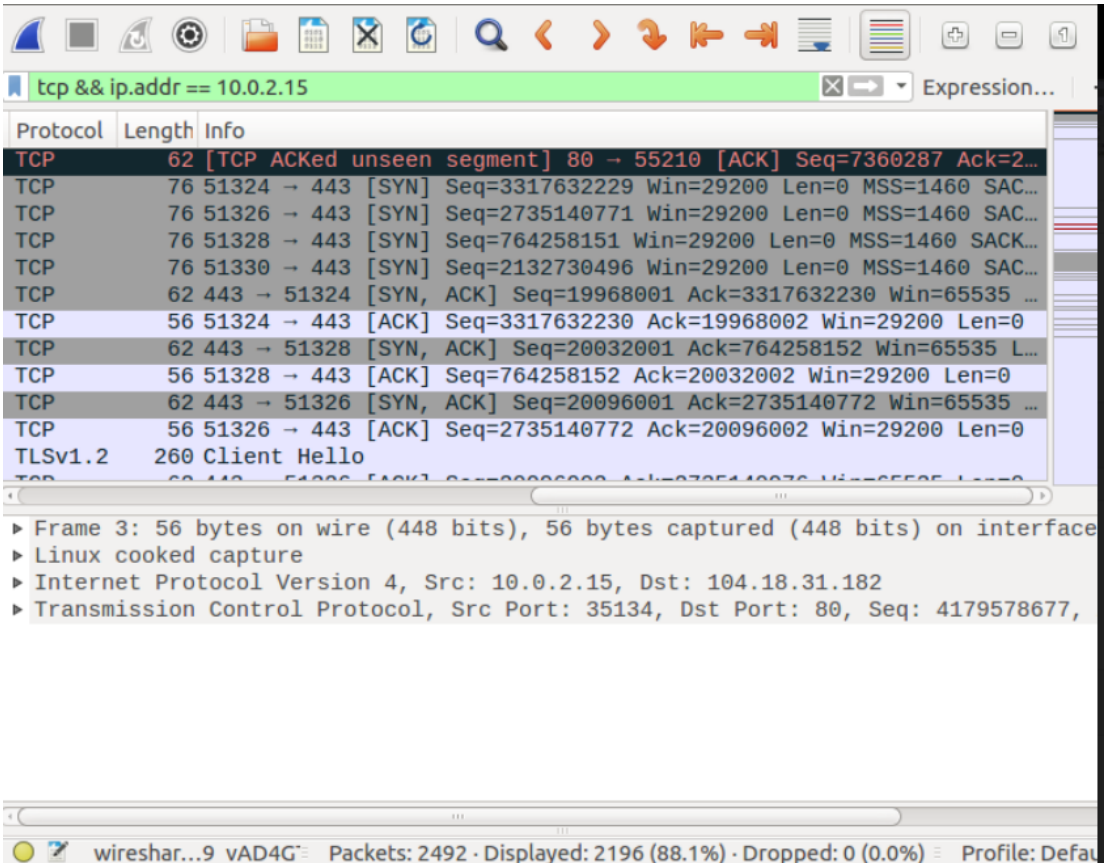
8) Clear out Wireshark and start a new capture.
9) Go to the following website. When there, use the form to choose a file (the copy of the Canterbury Tales that you've stashed away somewhere on your hard drive) and

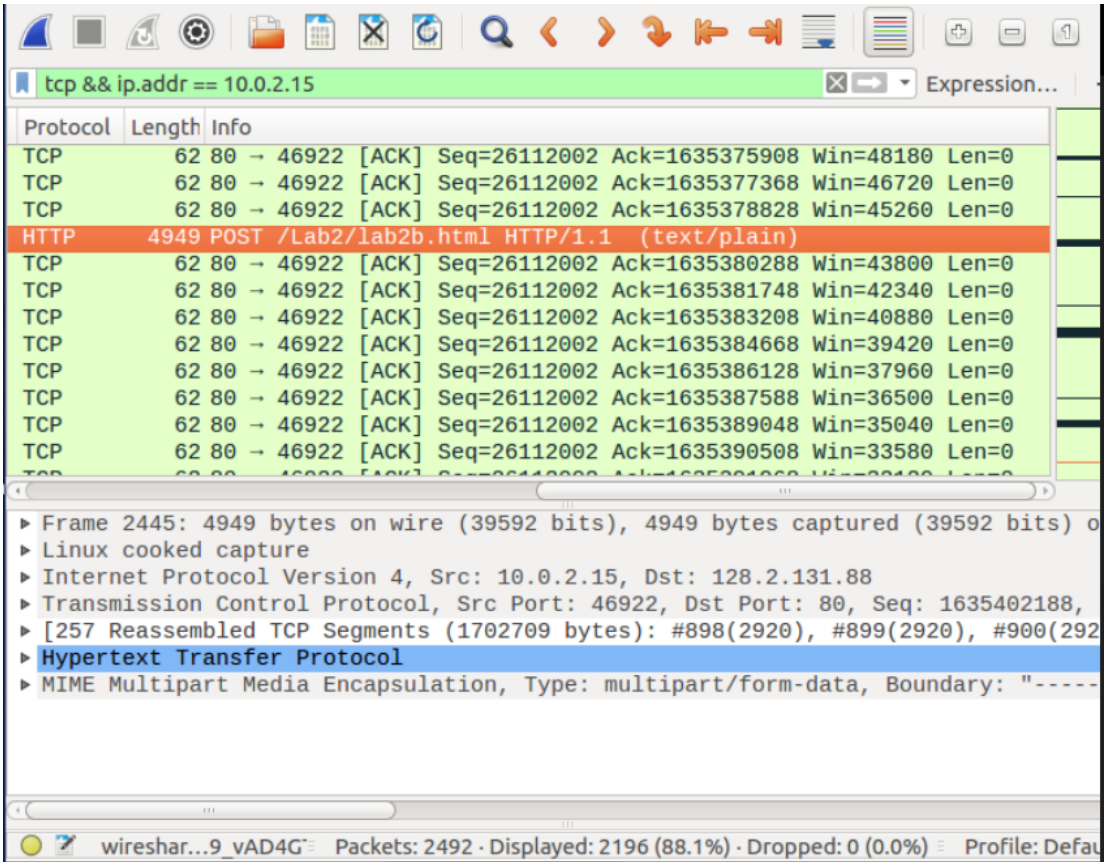
upload the file. The point of this exercise is to capture a lengthy TCP stream which originates at your computer. **http://www.ini740.com/Lab2/lab2a.html**

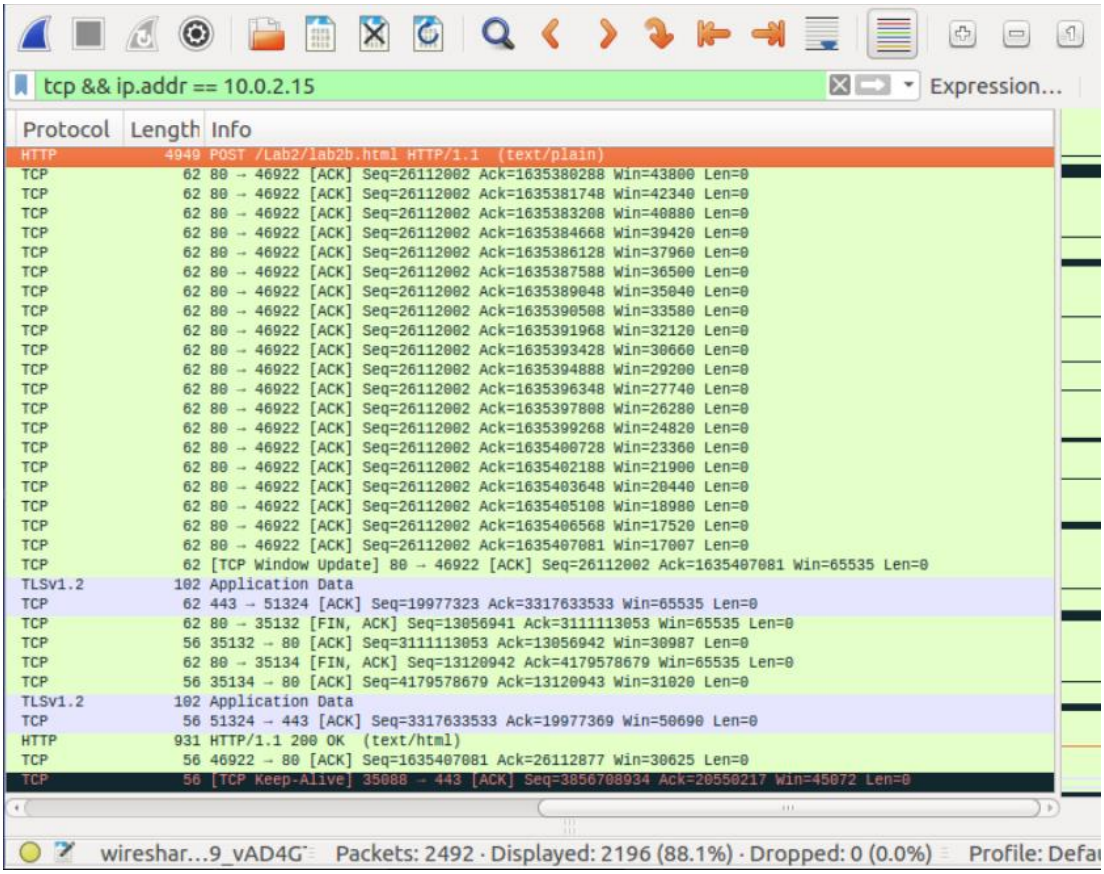
10) Stop the Wireshark capture.
11) Let's look at what you captured. First, filter the results to look for TCP packets and to only look at those going to and from your computer with the filter **"tcp && ip.addr == <your IP address>**". If you have other services running on your computer, you might want to further filter so you only display TCP packets between your computer and the ECE (Electrical and Computer Engineering department of CMU) webserver. What you should see is a series of TCP and HTTP messages between your computer and www.ece.cmu.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message and a series of "HTTP Continuation" messages being sent from your computer to the server. HTTP Continuation messages are Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. You should also see TCP ACK segments being returned from the server to your computer. Take a screenshot showing the three-way handshake.





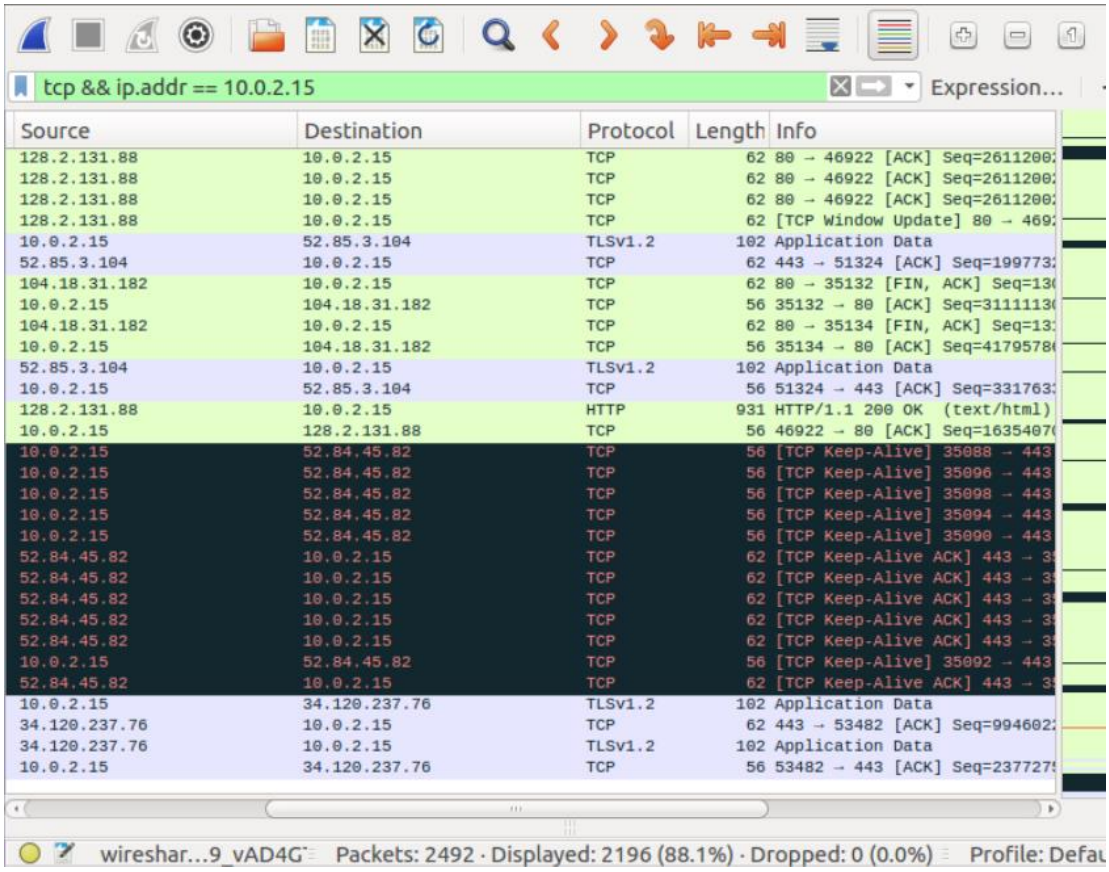

tcp && ip.addr == 10.0.2.15
Expression…
Protocol Length Info
TCP 62 [TCP ACKed unseen segment] 80 → 55210 [ACK] Seq=7360287 Ack=2…
TCP 76 51324 → 443 [SYN] Seq=3317632229 Win=29200 Len=0 MSS=1460 SAC…
TCP 76 51326 → 443 [SYN] Seq=2735140771 Win=29200 Len=0 MSS=1460 SAC…
TCP 76 51328 → 443 [SYN] Seq=764258151 Win=29200 Len=0 MSS=1460 SACK…
TCP 76 51330 → 443 [SYN] Seq=2132730496 Win=29200 Len=0 MSS=1460 SAC…
TCP 62 443 → 51324 [SYN, ACK] Seq=19968001 Ack=3317632230 Win=65535 …
TCP 56 51324 → 443 [ACK] Seq=3317632230 Ack=19968002 Win=29200 Len=0
TCP 62 443 → 51328 [SYN, ACK] Seq=20032001 Ack=764258152 Win=65535 L…
TCP 56 51328 → 443 [ACK] Seq=764258152 Ack=20032002 Win=29200 Len=0
TCP 62 443 → 51326 [SYN, ACK] Seq=20096001 Ack=2735140772 Win=65535 …
TCP 56 51326 → 443 [ACK] Seq=2735140772 Ack=20096002 Win=29200 Len=0
TLSv1.2 260 Client Hello
Frame 3: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 104.18.31.182
Transmission Control Protocol, Src Port: 35134, Dst Port: 80, Seq: 4179578677,
wireshar…9 vAD4G
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau

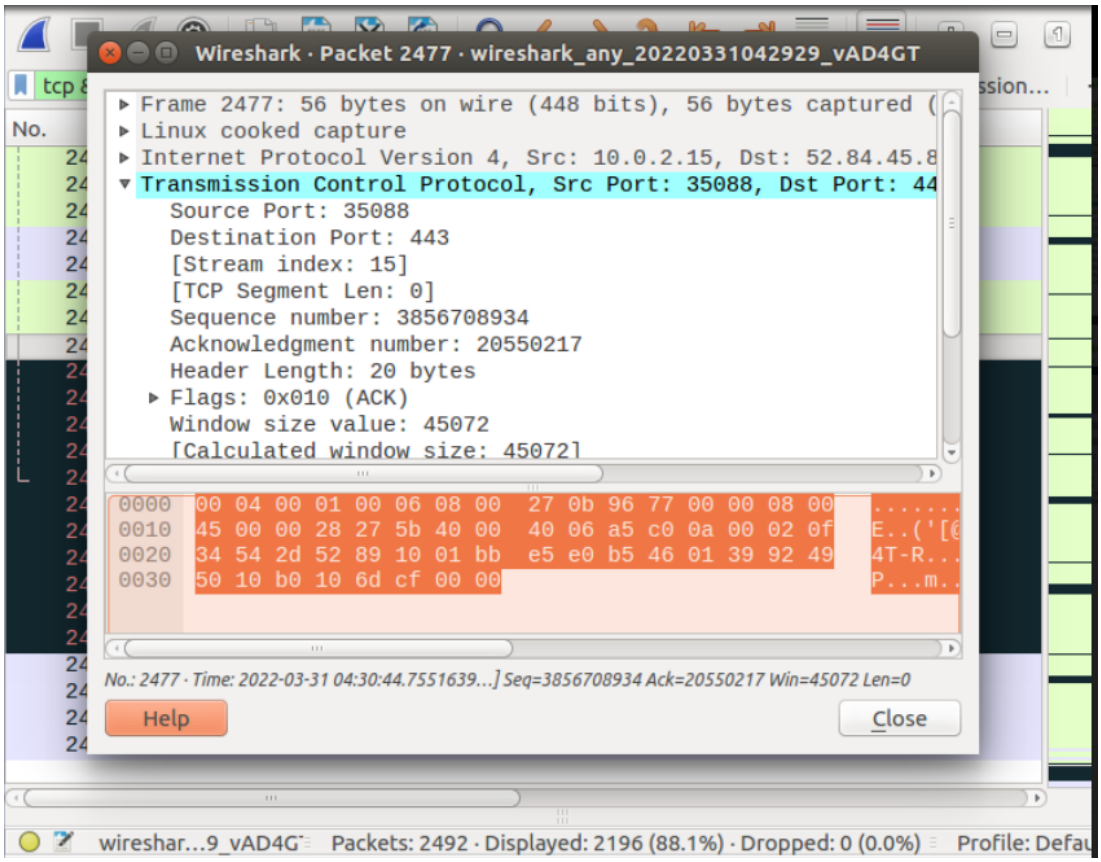

tcp && ip.addr == 10.0.2.15
Expression…
Protocol Length Info
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635375908 Win=48180 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635377368 Win=46720 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635378828 Win=45260 Len=0
HTTP 4949 POST /Lab2/lab2b.html HTTP/1.1 (text/plain)
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635380288 Win=43800 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635381748 Win=42340 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635383208 Win=40880 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635384668 Win=39420 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635386128 Win=37960 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635387588 Win=36500 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635389048 Win=35040 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635390508 Win=33580 Len=0
Frame 2445: 4949 bytes on wire (39592 bits), 4949 bytes captured (39592 bits) o
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 128.2.131.88
Transmission Control Protocol, Src Port: 46922, Dst Port: 80, Seq: 1635402188,
[257 Reassembled TCP Segments (1702709 bytes): #898(2920), #899(2920), #900(292
Hypertext Transfer Protocol
MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "-----
wireshar…9_vAD4G
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau



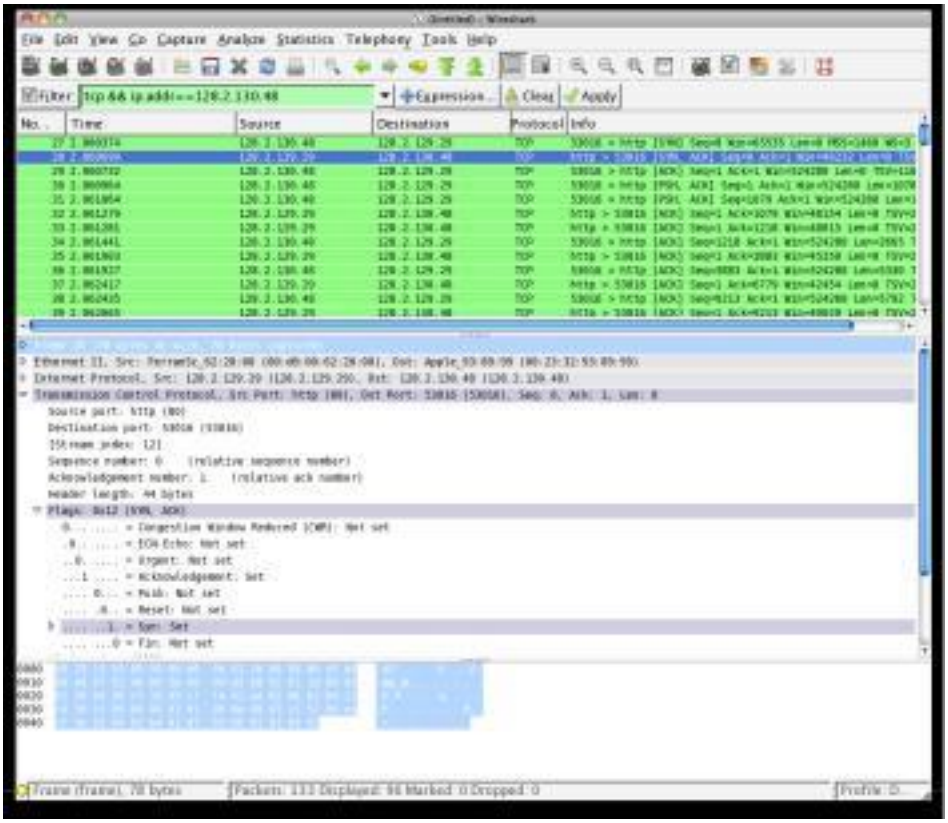
12) What is the IP address and TCP port number used by your computer (client) to transfer the file? What is the IP address of the server? On what port number is it sending and receiving TCP segments for this transfer of the file?

- **Client IP address & port : 10.0.2.15, 35088**
  Server IP address & port : 52.84.45.8, 443


tcp && ip.addr == 10.0.2.15
Expression…
Source
Destination
Protocol
Length
Info
128.2.131.88 10.0.2.15 TCP 62 80 → 46922 [ACK] Seq=2611200
128.2.131.88 10.0.2.15 TCP 62 80 → 46922 [ACK] Seq=2611200
128.2.131.88 10.0.2.15 TCP 62 80 → 46922 [ACK] Seq=2611200
128.2.131.88 10.0.2.15 TCP 62 [TCP Window Update] 80 → 469
10.0.2.15 52.85.3.104 TLSv1.2 102 Application Data
52.85.3.104 10.0.2.15 TCP 62 443 → 51324 [ACK] Seq=199773
104.18.31.182 10.0.2.15 TCP 62 80 → 35132 [FIN, ACK] Seq=13
10.0.2.15 104.18.31.182 TCP 56 35132 → 80 [ACK] Seq=3111113
104.18.31.182 10.0.2.15 TCP 62 80 → 35134 [FIN, ACK] Seq=13
10.0.2.15 104.18.31.182 TCP 56 35134 → 80 [ACK] Seq=4179578
52.85.3.104 10.0.2.15 TLSv1.2 102 Application Data
10.0.2.15 52.85.3.104 TCP 56 51324 → 443 [ACK] Seq=331763
128.2.131.88 10.0.2.15 HTTP 931 HTTP/1.1 200 OK (text/html)
10.0.2.15 128.2.131.88 TCP 56 46922 → 80 [ACK] Seq=1635407
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35088 → 443
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35096 → 443
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35098 → 443
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35094 → 443
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35090 → 443
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
10.0.2.15 52.84.45.82 TCP 56 [TCP Keep-Alive] 35092 → 443
52.84.45.82 10.0.2.15 TCP 62 [TCP Keep-Alive ACK] 443 → 3
10.0.2.15 34.120.237.76 TLSv1.2 102 Application Data
34.120.237.76 10.0.2.15 TCP 62 443 → 53482 [ACK] Seq=994602
34.120.237.76 10.0.2.15 TLSv1.2 102 Application Data
10.0.2.15 34.120.237.76 TCP 56 53482 → 443 [ACK] Seq=237727
wireshar…9_vAD4G
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau


Wireshark · Packet 2477 · wireshark_any_20220331042929_vAD4GT
Frame 2477: 56 bytes on wire (448 bits), 56 bytes captured (
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 52.84.45.8
Transmission Control Protocol, Src Port: 35088, Dst Port: 44
Source Port: 35088
Destination Port: 443
[Stream index: 15]
[TCP Segment Len: 0]
Sequence number: 3856708934
Acknowledgment number: 20550217
Header Length: 20 bytes
Flags: 0x010 (ACK)
Window size value: 45072
[Calculated window size: 45072]
0000 00 04 00 01 00 06 08 00 27 0b 96 77 00 00 08 00
0010 45 00 00 28 27 5b 40 00 40 06 a5 c0 0a 00 02 0f
0020 34 54 2d 52 89 10 01 bb e5 e0 b5 46 01 39 92 49
0030 50 10 b0 10 6d cf 00 00
No.: 2477 · Time: 2022-03-31 04:30:44.7551639…] Seq=3856708934 Ack=20550217 Win=45072 Len=0
Help
Close
wireshar…9_vAD4G
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau

13) Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select Analyze → Enabled Protocols. Then uncheck the HTTP box and select OK. You should now see a Wireshark window that looks like:



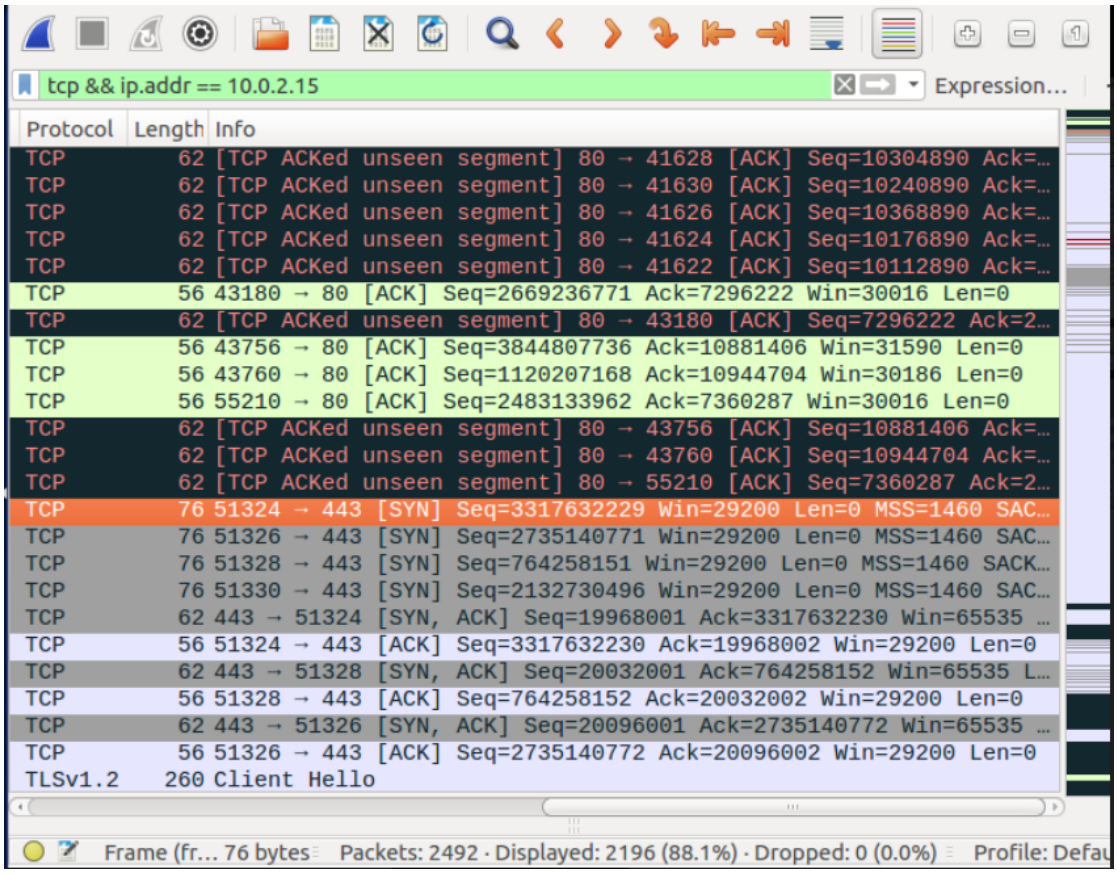
This is what we're looking for - a series of TCP segments sent between your computer and www.ece.cmu.edu. We will use the packet trace that you have captured to study TCP behaviour in the rest of this lab.

Step 2b: TCP Basics

Answer the following questions for the TCP segments:

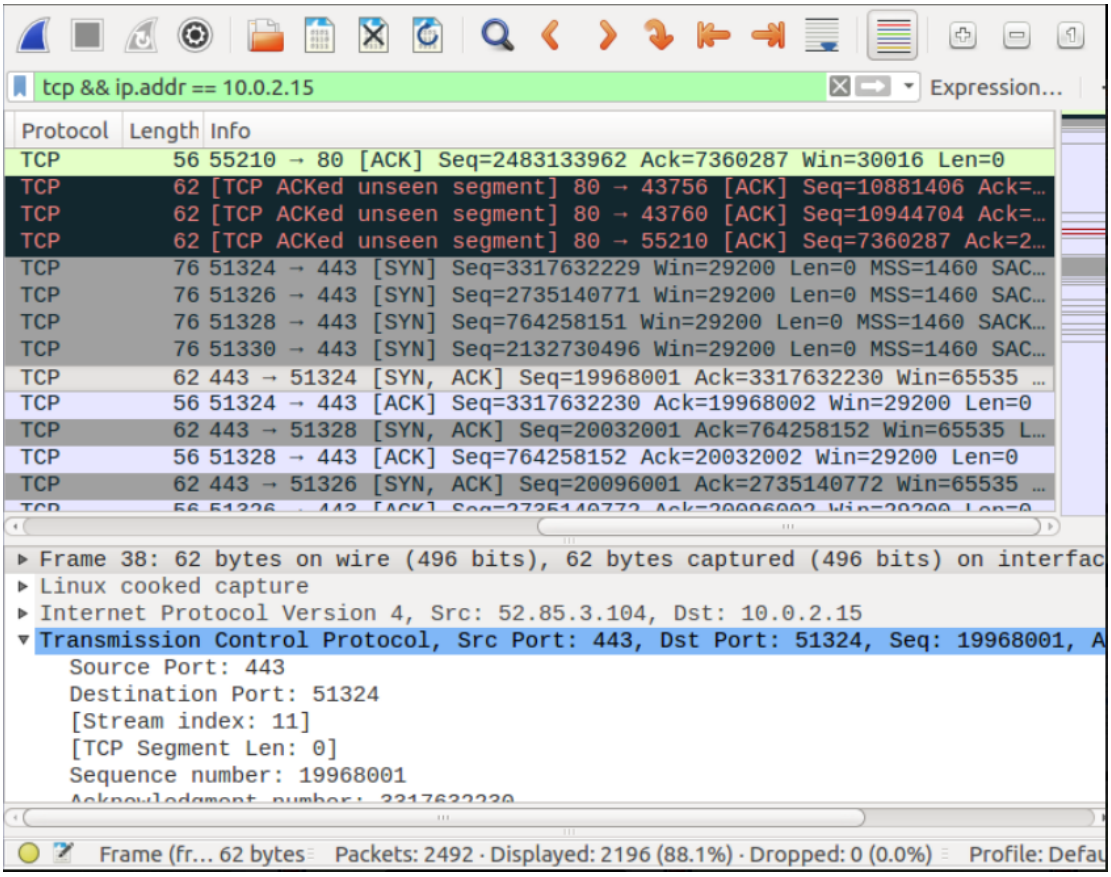
14) What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection? What element of the segment identifies it as a SYN segment? Wireshark uses relative sequence numbers by default. Can you obtain absolute sequence numbers instead? How? You can use relative sequence numbers to answer the remaining questions.

- **TCP SYN sequence number : 3317632229.**
  This segment is identifiable by the [SYN] tag in info.
  To disable relative sequence numbers and instead display them as the real absolute numbers, go to the TCP preferences and uncheck the box for relative.



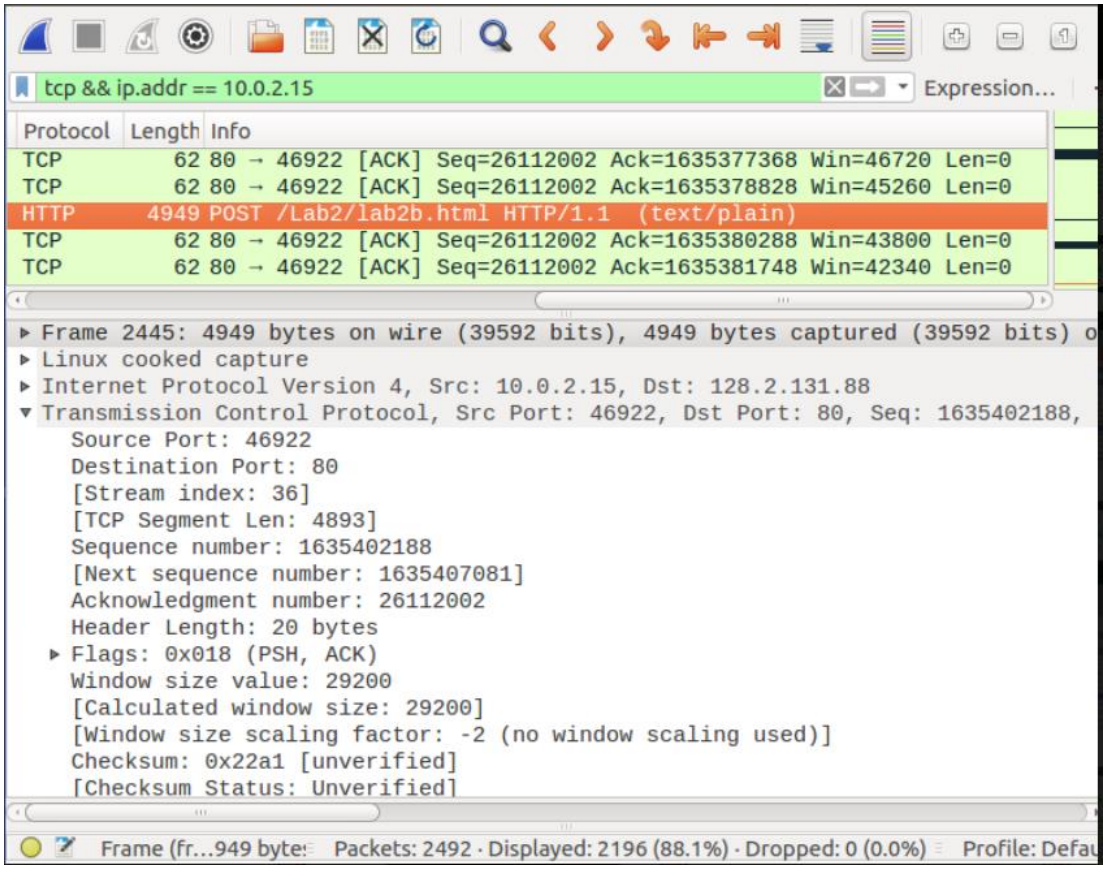
15) What is the sequence number of the SYNACK segment sent by the server in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did the server determine that value? What element in the segment identifies it as a SYNACK segment?

- **Sequence number : 19968001.**
  Acknowledgement number : 3317632230.
  The server determines the acknowledgment number value based on sequence number of the packet being acknowledged.
  The [SYN, ACK] tag identifies the segment as the same.

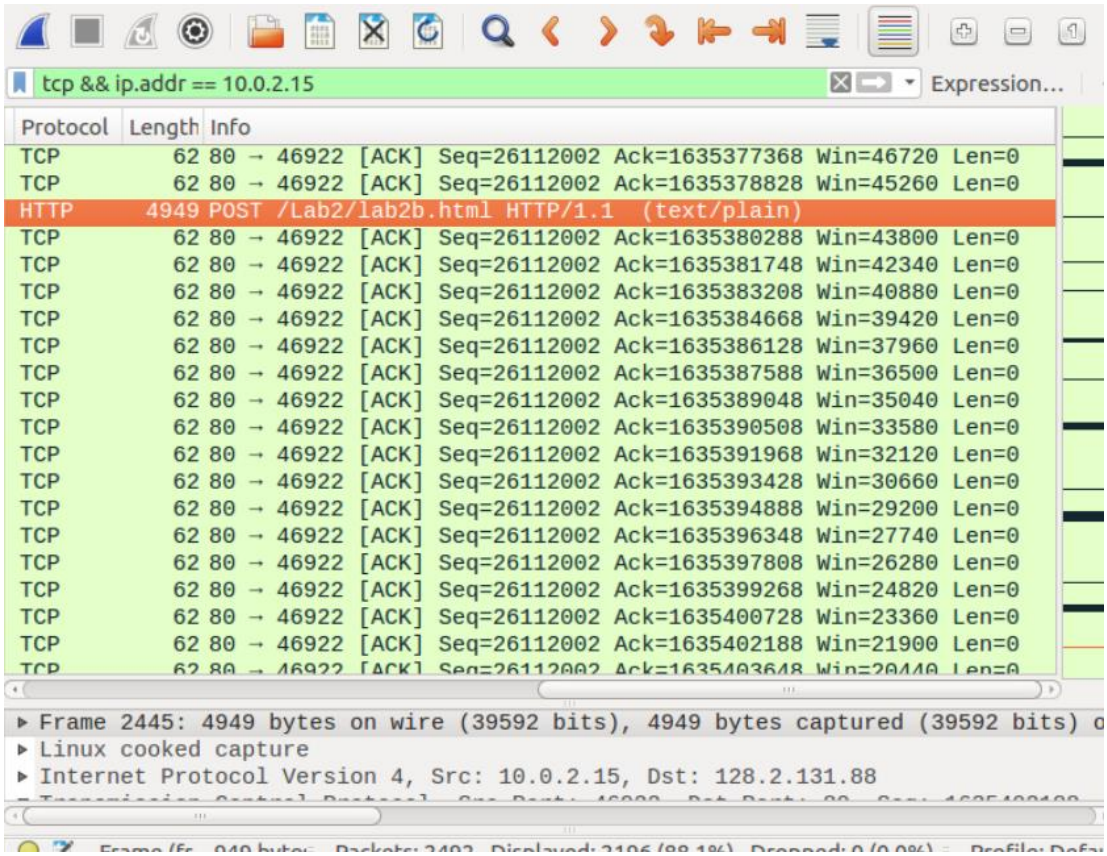


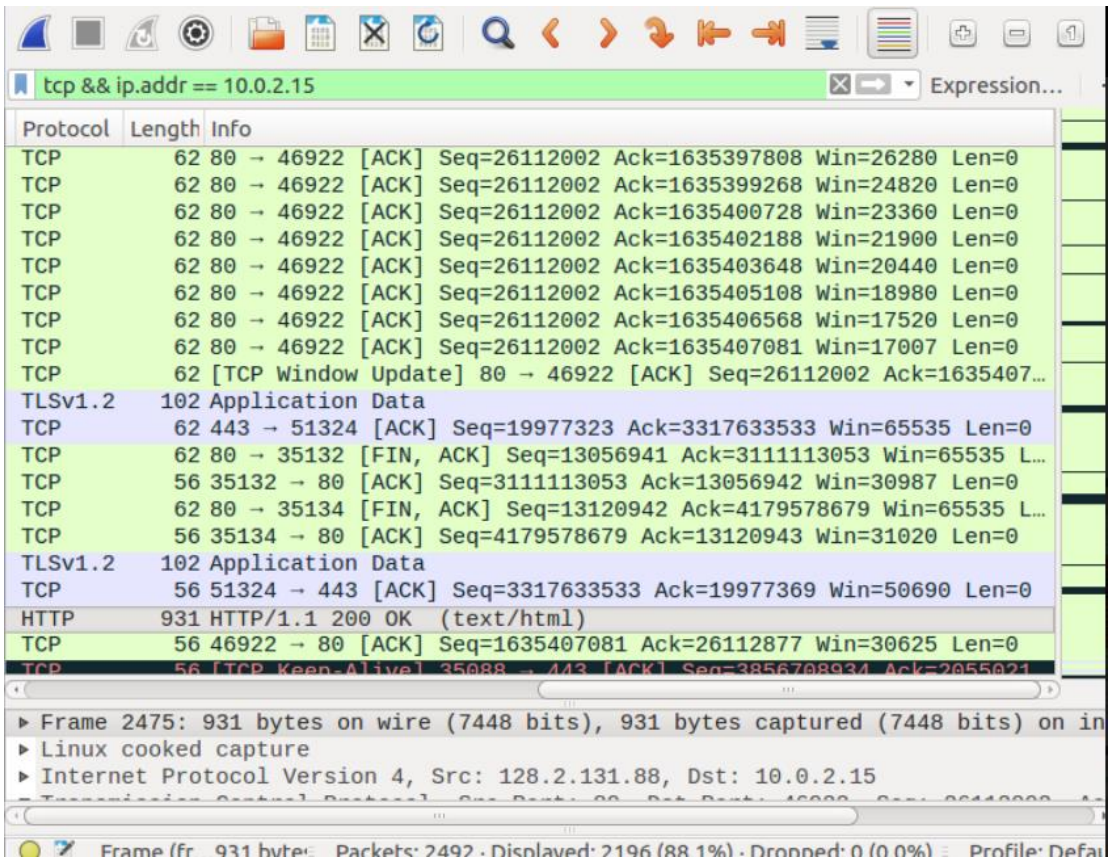
16) What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.

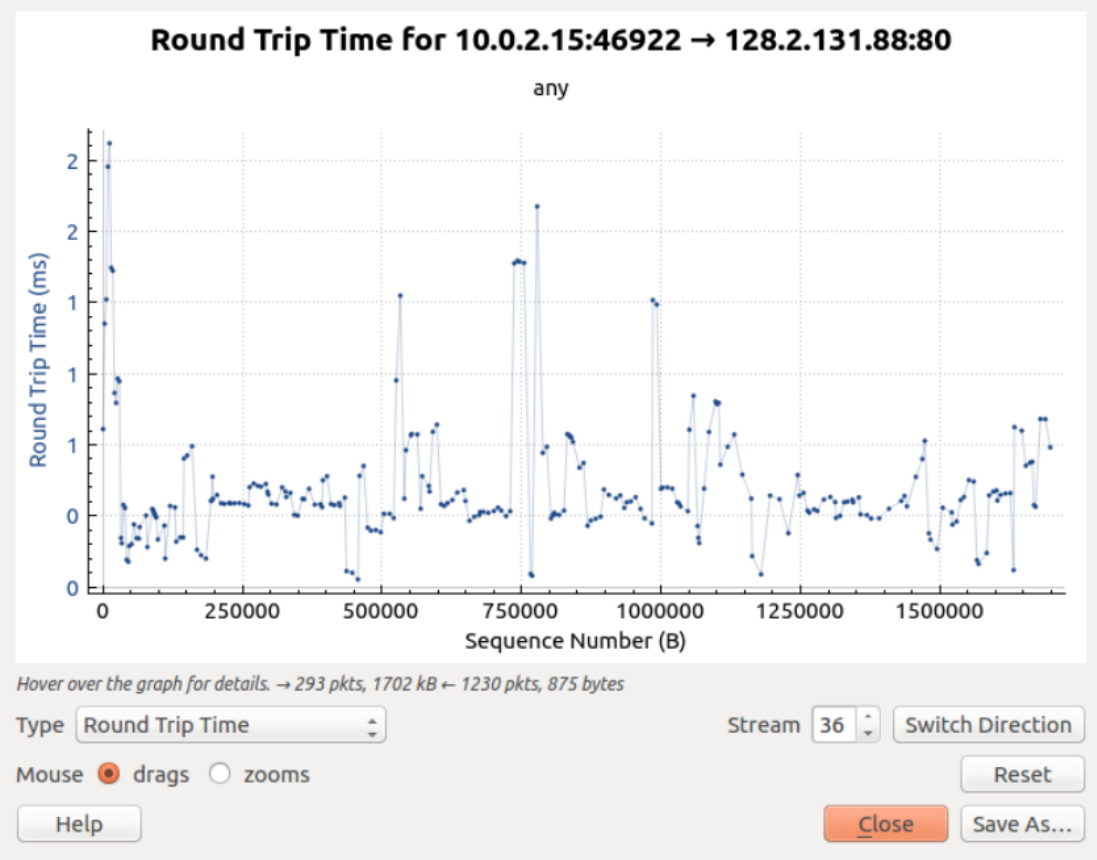
- **Sequence number : 1635402188**



17) Consider the TCP segment containing the HTTP POST as the first segment in the non-overhead part of the TCP connection.
Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the server. Then select: Statistics → TCP Stream Graph → Round Trip Time Graph.

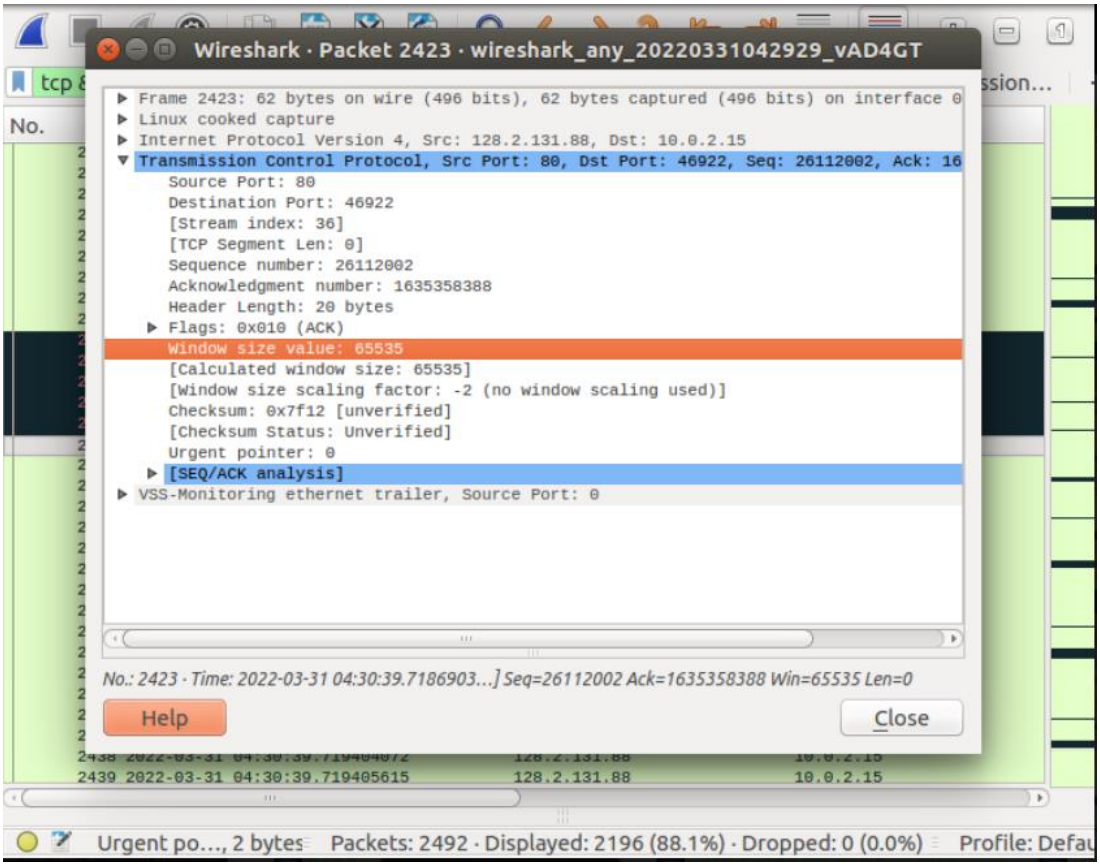

tcp && ip.addr == 10.0.2.15
Expression…
Protocol Length Info
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635377368 Win=46720 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635378828 Win=45260 Len=0
HTTP 4949 POST /Lab2/lab2b.html HTTP/1.1 (text/plain)
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635380288 Win=43800 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635381748 Win=42340 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635383208 Win=40880 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635384668 Win=39420 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635386128 Win=37960 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635387588 Win=36500 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635389048 Win=35040 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635390508 Win=33580 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635391968 Win=32120 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635393428 Win=30660 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635394888 Win=29200 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635396348 Win=27740 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635397808 Win=26280 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635399268 Win=24820 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635400728 Win=23360 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635402188 Win=21900 Len=0
▸ Frame 2445: 4949 bytes on wire (39592 bits), 4949 bytes captured (39592 bits) o
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 128.2.131.88


tcp && ip.addr == 10.0.2.15
Expression…
Protocol Length Info
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635397808 Win=26280 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635399268 Win=24820 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635400728 Win=23360 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635402188 Win=21900 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635403648 Win=20440 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635405108 Win=18980 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635406568 Win=17520 Len=0
TCP 62 80 → 46922 [ACK] Seq=26112002 Ack=1635407081 Win=17007 Len=0
TCP 62 [TCP Window Update] 80 → 46922 [ACK] Seq=26112002 Ack=1635407…
TLSv1.2 102 Application Data
TCP 62 443 → 51324 [ACK] Seq=19977323 Ack=3317633533 Win=65535 Len=0
TCP 62 80 → 35132 [FIN, ACK] Seq=13056941 Ack=3111113053 Win=65535 L…
TCP 56 35132 → 80 [ACK] Seq=3111113053 Ack=13056942 Win=30987 Len=0
TCP 62 80 → 35134 [FIN, ACK] Seq=13120942 Ack=4179578679 Win=65535 L…
TCP 56 35134 → 80 [ACK] Seq=4179578679 Ack=13120943 Win=31020 Len=0
TLSv1.2 102 Application Data
TCP 56 51324 → 443 [ACK] Seq=3317633533 Ack=19977369 Win=50690 Len=0
HTTP 931 HTTP/1.1 200 OK (text/html)
TCP 56 46922 → 80 [ACK] Seq=1635407081 Ack=26112877 Win=30625 Len=0
▸ Frame 2475: 931 bytes on wire (7448 bits), 931 bytes captured (7448 bits) on in
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 128.2.131.88, Dst: 10.0.2.15
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)



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

- **Advertised buffer space : 65535**



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

- **There were a few retransmitted segments as indicated by the black label and the tag reading 'unacked segment'.**

Step 2c: Statistics

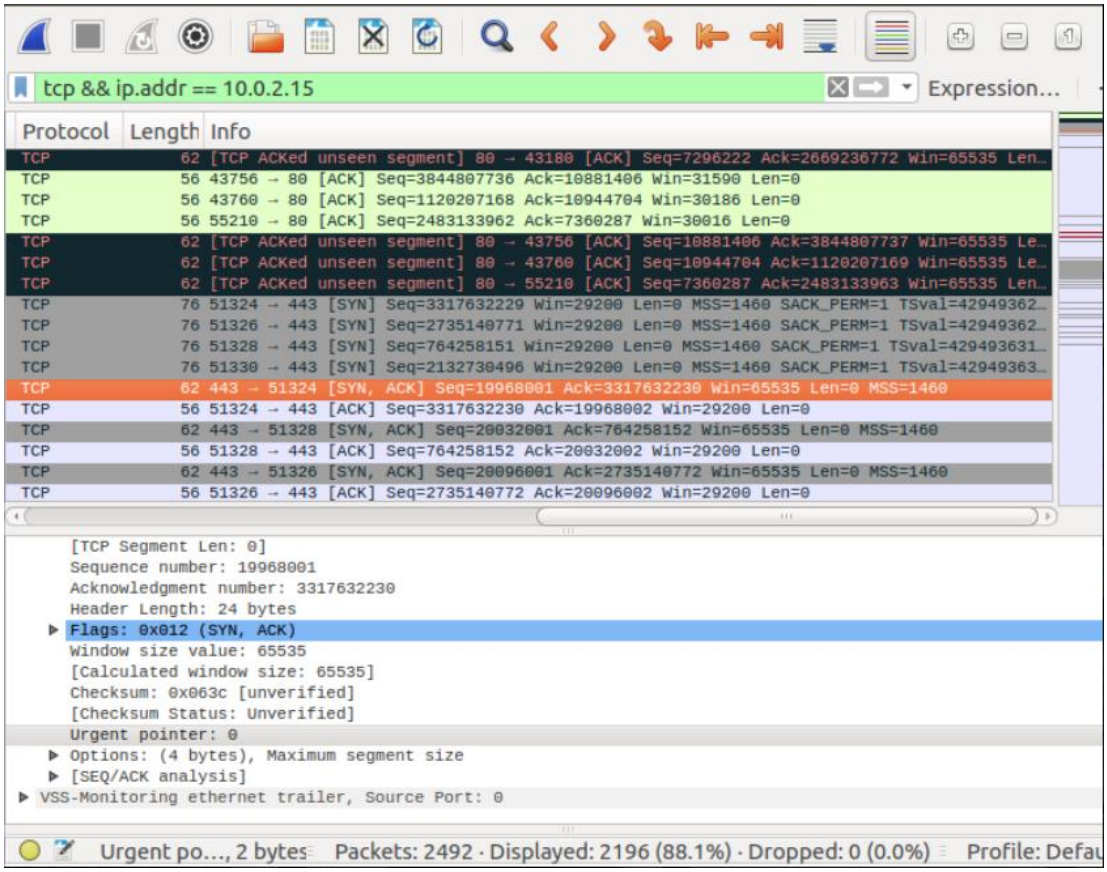
Wireshark has some fairly robust reporting abilities, most of which are accessed via the Statistics menu. Spend a few minutes messing around with the options on that menu, trying to figure out what each report is telling you. Then, answer the following questions about the Canterbury Tales capture:

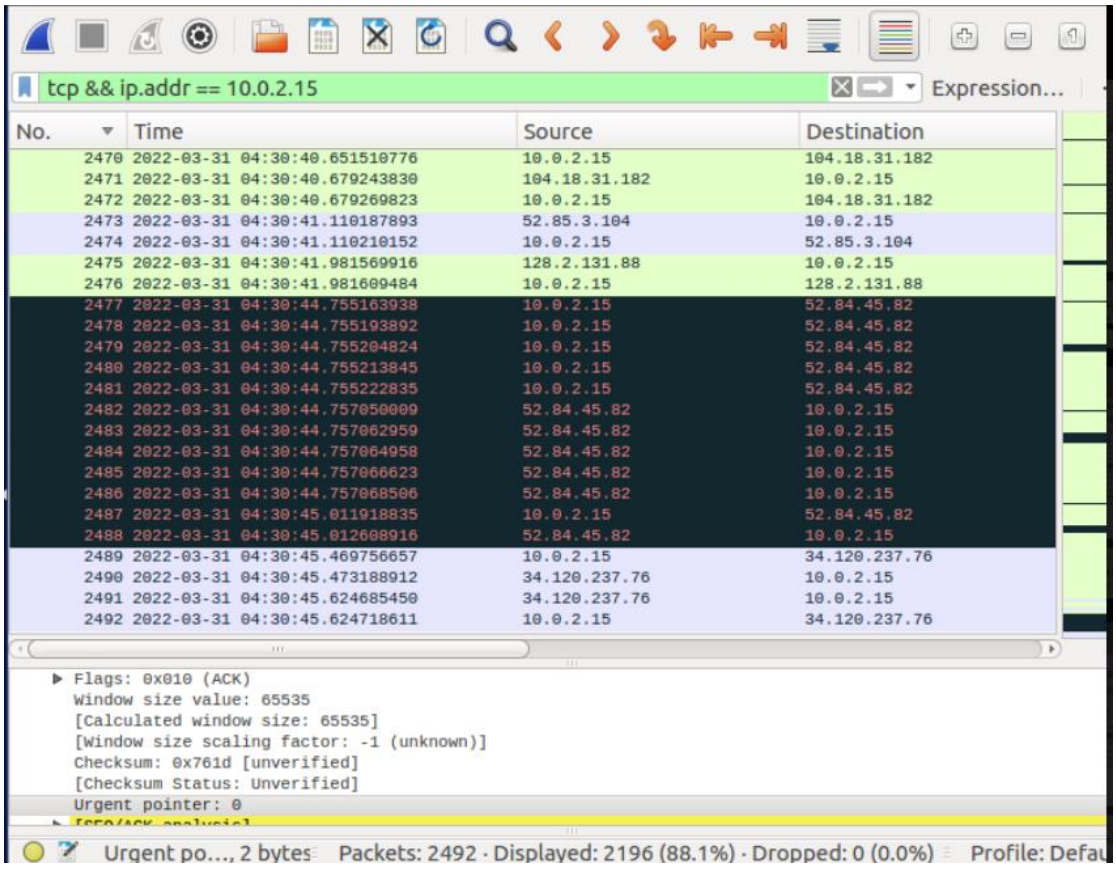
22) What is the most common TCP packet length range? What is the second most common TCP packet length range?
Describe what actions you took to get answers to these questions from Wireshark.

- **Common TCP packet size range ~65000 (around 64kB).**
  Second most common size ~512.

  These inferences could be made from the window and frame size fields.

23) A conversation represents a traffic between two hosts. With which remote host did your local host converse the most (in bytes)? How many packets were sent from your host? How many packets were sent from the remote host?

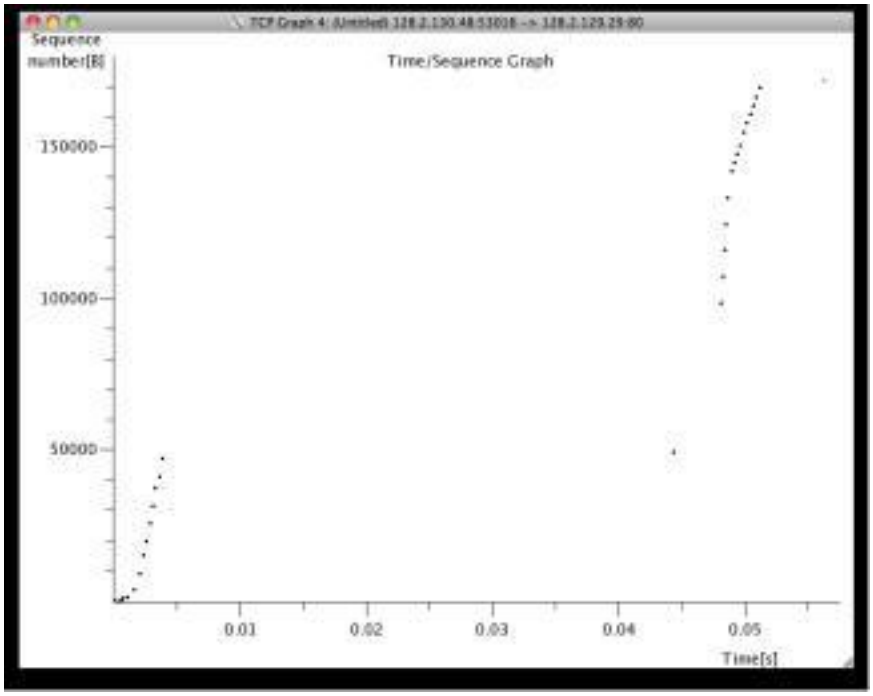

tcp && ip.addr == 10.0.2.15
Expression…
Protocol Length Info
TCP 62 [TCP ACKed unseen segment] 80 → 43180 [ACK] Seq=7296222 Ack=2669236772 Win=65535 Len…
TCP 56 43756 → 80 [ACK] Seq=3844807736 Ack=10881406 Win=31590 Len=0
TCP 56 43760 → 80 [ACK] Seq=1120207168 Ack=10944704 Win=30186 Len=0
TCP 56 55210 → 80 [ACK] Seq=2483133962 Ack=7360287 Win=30016 Len=0
TCP 62 [TCP ACKed unseen segment] 80 → 43756 [ACK] Seq=10881406 Ack=3844807737 Win=65535 Le…
TCP 62 [TCP ACKed unseen segment] 80 → 43760 [ACK] Seq=10944704 Ack=1120207169 Win=65535 Le…
TCP 62 [TCP ACKed unseen segment] 80 → 55210 [ACK] Seq=7360287 Ack=2483133963 Win=65535 Len…
TCP 76 51324 → 443 [SYN] Seq=3317632229 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=42949362…
TCP 76 51326 → 443 [SYN] Seq=2735140771 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=42949362…
TCP 76 51328 → 443 [SYN] Seq=764258151 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=429493631…
TCP 76 51330 → 443 [SYN] Seq=2132730496 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=42949363…
TCP 62 443 → 51324 [SYN, ACK] Seq=19968001 Ack=3317632230 Win=65535 Len=0 MSS=1460
TCP 56 51324 → 443 [ACK] Seq=3317632230 Ack=19968002 Win=29200 Len=0
TCP 62 443 → 51328 [SYN, ACK] Seq=20032001 Ack=764258152 Win=65535 Len=0 MSS=1460
TCP 56 51328 → 443 [ACK] Seq=764258152 Ack=20032002 Win=29200 Len=0
TCP 62 443 → 51326 [SYN, ACK] Seq=20096001 Ack=2735140772 Win=65535 Len=0 MSS=1460
TCP 56 51326 → 443 [ACK] Seq=2735140772 Ack=20096002 Win=29200 Len=0
[TCP Segment Len: 0]
Sequence number: 19968001
Acknowledgment number: 3317632230
Header Length: 24 bytes
Flags: 0x012 (SYN, ACK)
Window size value: 65535
[Calculated window size: 65535]
Checksum: 0x063c [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (4 bytes), Maximum segment size
[SEQ/ACK analysis]
VSS-Monitoring ethernet trailer, Source Port: 0
Urgent po…, 2 bytes
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau


tcp && ip.addr == 10.0.2.15
Expression…
No. Time Source Destination
2470 2022-03-31 04:30:40.651510776 10.0.2.15 104.18.31.182
2471 2022-03-31 04:30:40.679243830 104.18.31.182 10.0.2.15
2472 2022-03-31 04:30:40.679269823 10.0.2.15 104.18.31.182
2473 2022-03-31 04:30:41.110187893 52.85.3.104 10.0.2.15
2474 2022-03-31 04:30:41.110210152 10.0.2.15 52.85.3.104
2475 2022-03-31 04:30:41.981569916 128.2.131.88 10.0.2.15
2476 2022-03-31 04:30:41.981609484 10.0.2.15 128.2.131.88
2477 2022-03-31 04:30:44.755163938 10.0.2.15 52.84.45.82
2478 2022-03-31 04:30:44.755193892 10.0.2.15 52.84.45.82
2479 2022-03-31 04:30:44.755204824 10.0.2.15 52.84.45.82
2480 2022-03-31 04:30:44.755213845 10.0.2.15 52.84.45.82
2481 2022-03-31 04:30:44.755222835 10.0.2.15 52.84.45.82
2482 2022-03-31 04:30:44.757050009 52.84.45.82 10.0.2.15
2483 2022-03-31 04:30:44.757062959 52.84.45.82 10.0.2.15
2484 2022-03-31 04:30:44.757064958 52.84.45.82 10.0.2.15
2485 2022-03-31 04:30:44.757066623 52.84.45.82 10.0.2.15
2486 2022-03-31 04:30:44.757068506 52.84.45.82 10.0.2.15
2487 2022-03-31 04:30:45.011918835 10.0.2.15 52.84.45.82
2488 2022-03-31 04:30:45.012608916 52.84.45.82 10.0.2.15
2489 2022-03-31 04:30:45.469756657 10.0.2.15 34.120.237.76
2490 2022-03-31 04:30:45.473188912 34.120.237.76 10.0.2.15
2491 2022-03-31 04:30:45.624685450 34.120.237.76 10.0.2.15
2492 2022-03-31 04:30:45.624718611 10.0.2.15 34.120.237.76
Flags: 0x010 (ACK)
Window size value: 65535
[Calculated window size: 65535]
[Window size scaling factor: -1 (unknown)]
Checksum: 0x761d [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Urgent po…, 2 bytes
Packets: 2492 · Displayed: 2196 (88.1%) · Dropped: 0 (0.0%)
Profile: Defau

Step 3: Congestion Control

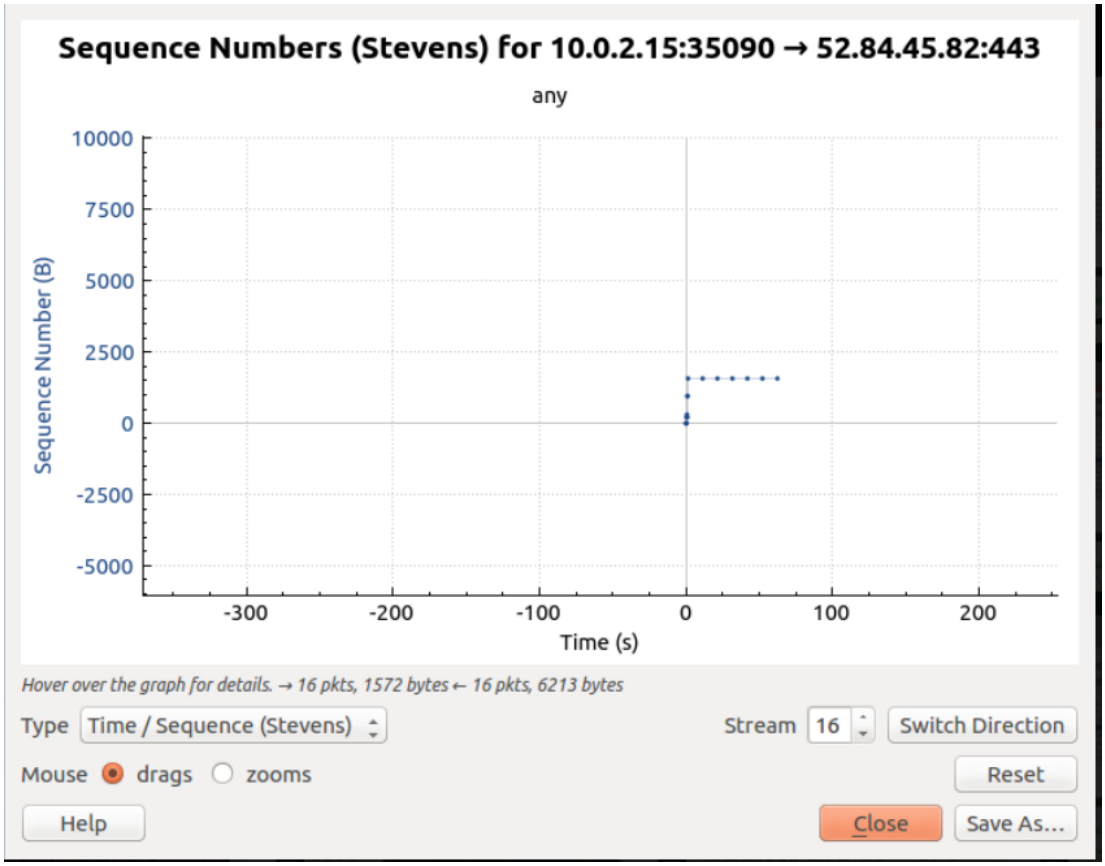
Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - Time-Sequence-Graph (Stevens) - to plot our data.

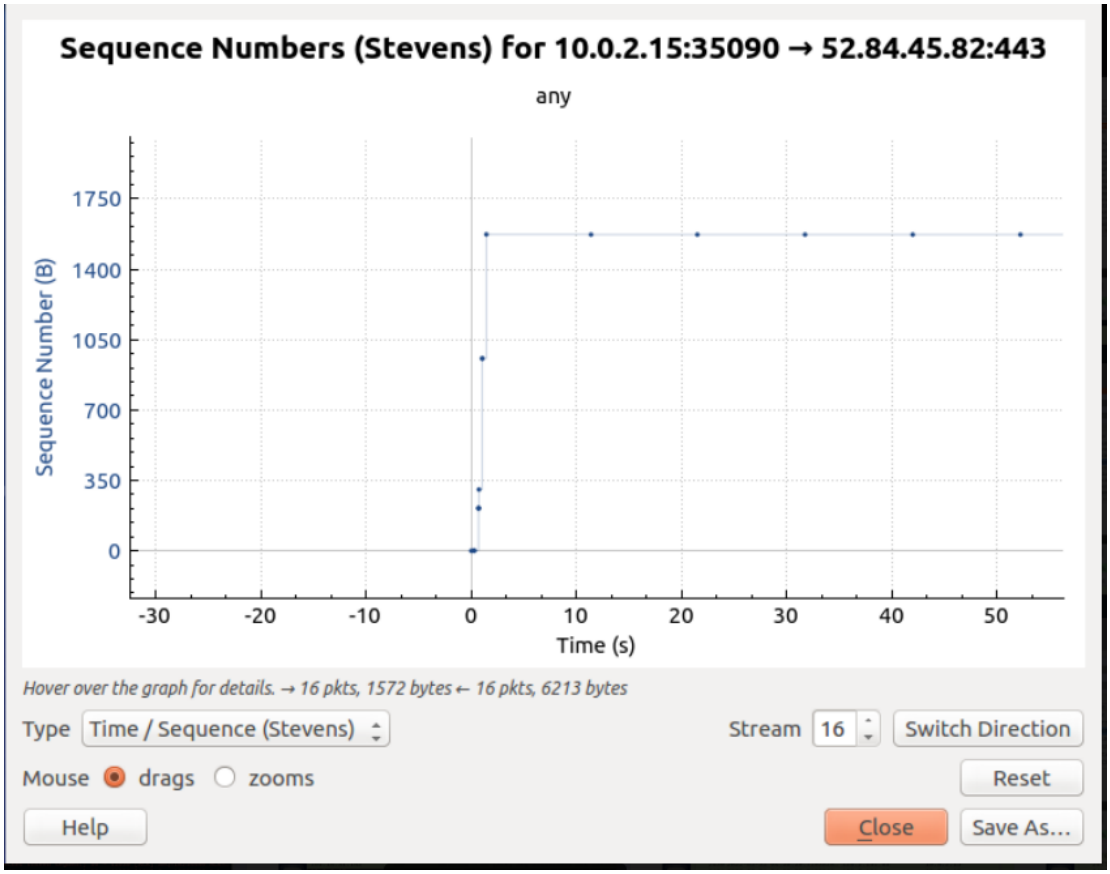
25) Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu: Statistics → TCP Stream Graph → Time-Sequence- Graph (Stevens). You should see a plot that looks like the following plot (though the individual plotted values may differ quite a bit).



Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender. Don't be distraught if your graph doesn't look like that shown above. Recall that the particular algorithms for managing congestion control can be implemented (or not) based on the OS you are running.

26) Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Make sure to include a copy of the plot in your report.

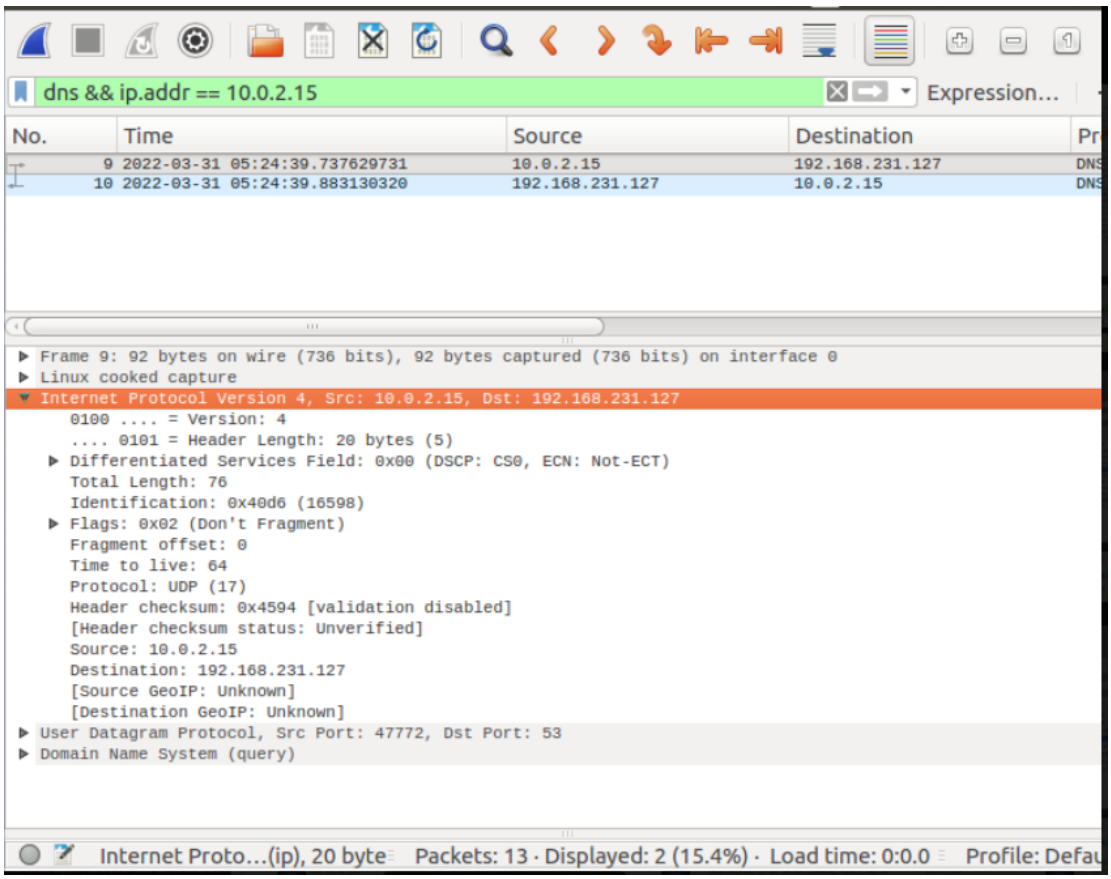




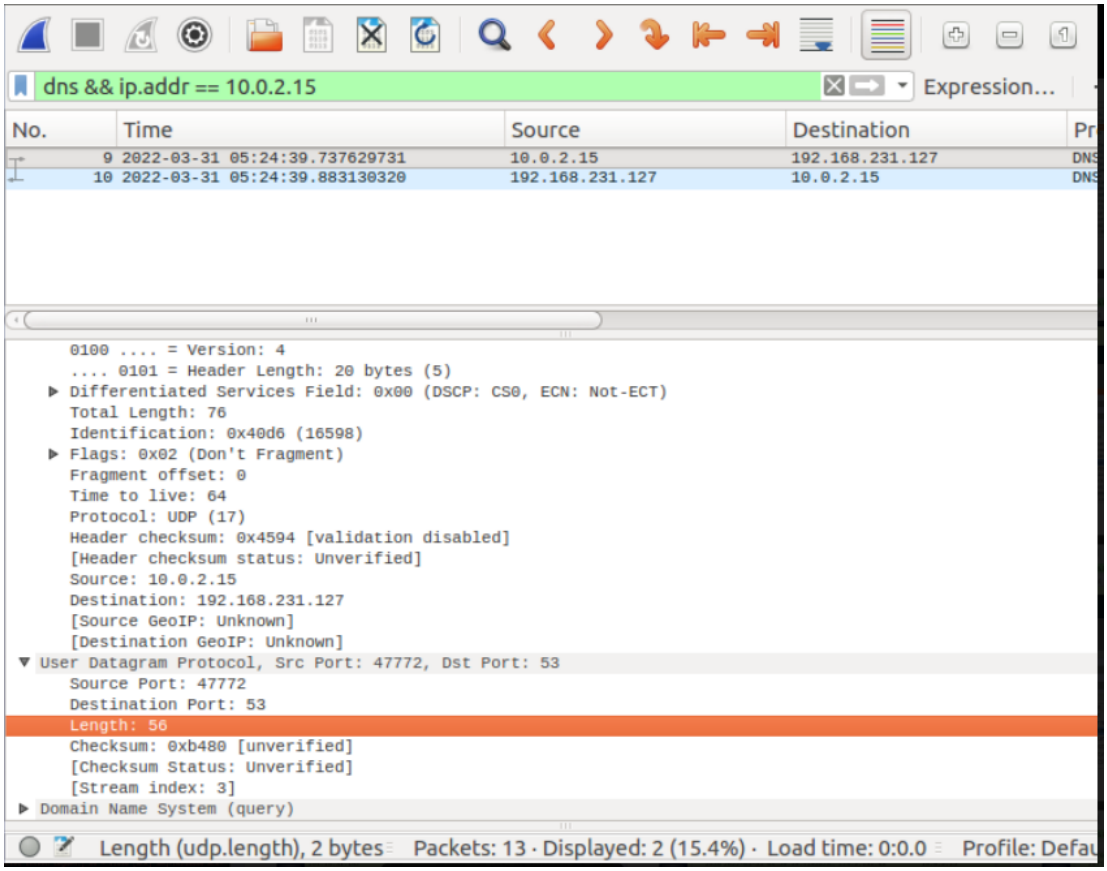
Step 4: The Network Layer

Let’s take this opportunity to check out a bit of IP traffic. We don’t have to capture any additional traffic, as everything we’ve seen today is carried over IP packets.

27) Load the capture file that you saved in step 1. Recall that this was a simple DNS query, carried in a UDP packet.
28) Take a look at the IP section of the DNS query (the packet that was generated when you used dig to request the address of **www.pluralsight.com**).
Match up the header fields with the format we discussed in class (don’t just look through Wireshark’s display -- instead, match the raw bytes with the pictures we saw in lecture, which I’ve copied on the right).

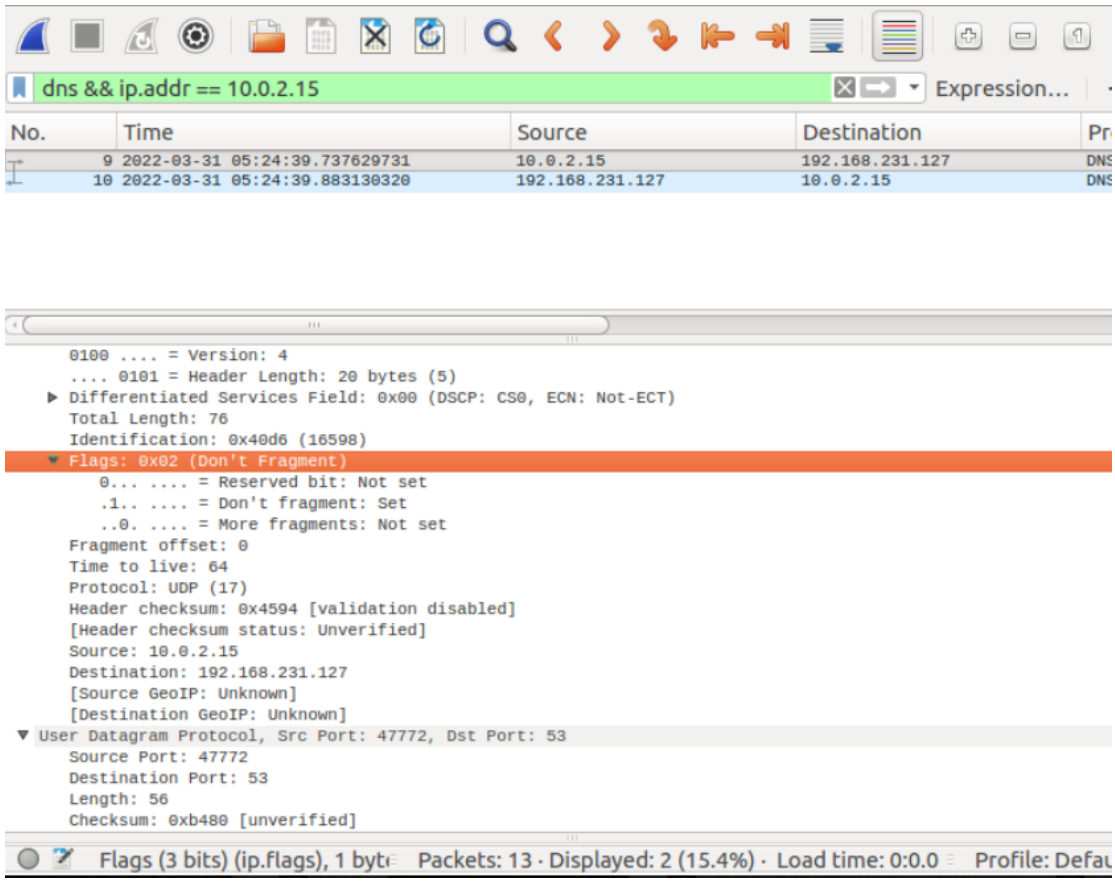


29) Most of the fields should match up and make perfect sense. Verify the Datagram Length, Upper-layer protocol and the IP address fields.

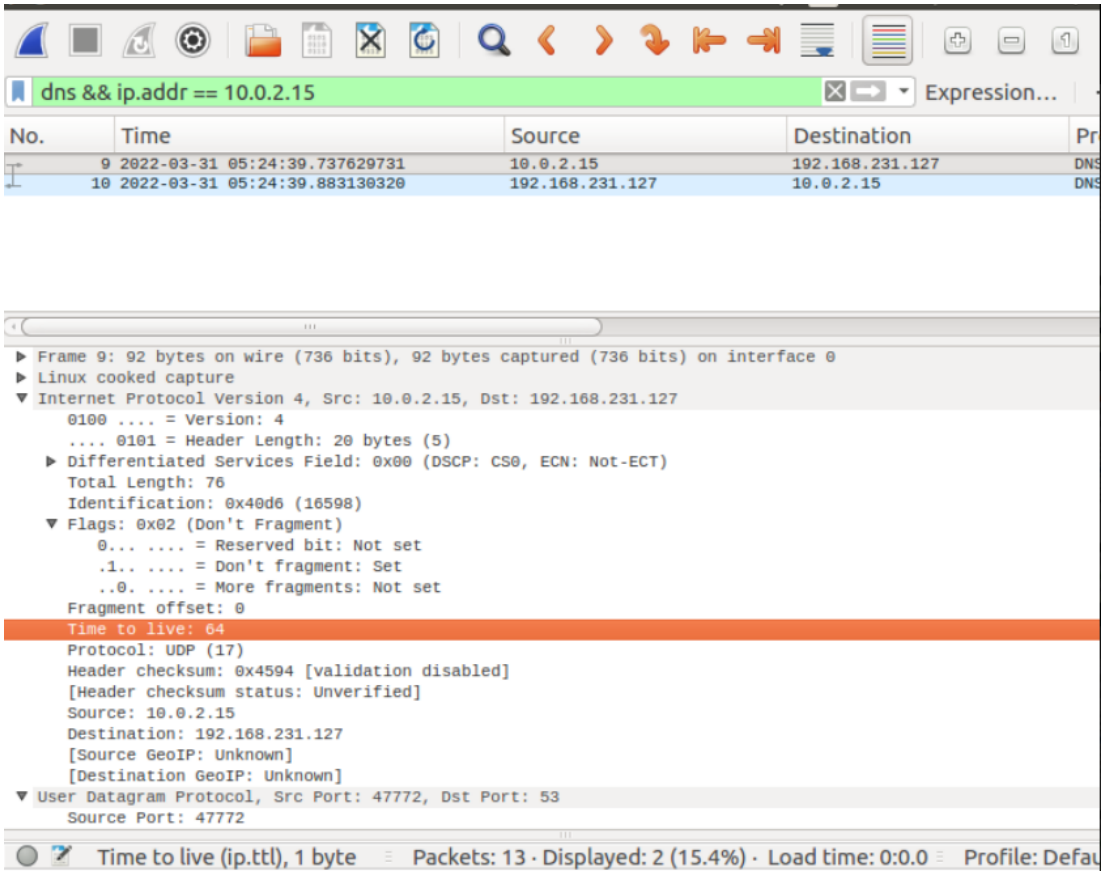


30) Are there any interesting features of the data in the identifier/flags/offset fields?

- **The fields indicate that the segment is not meant to be fragmented**



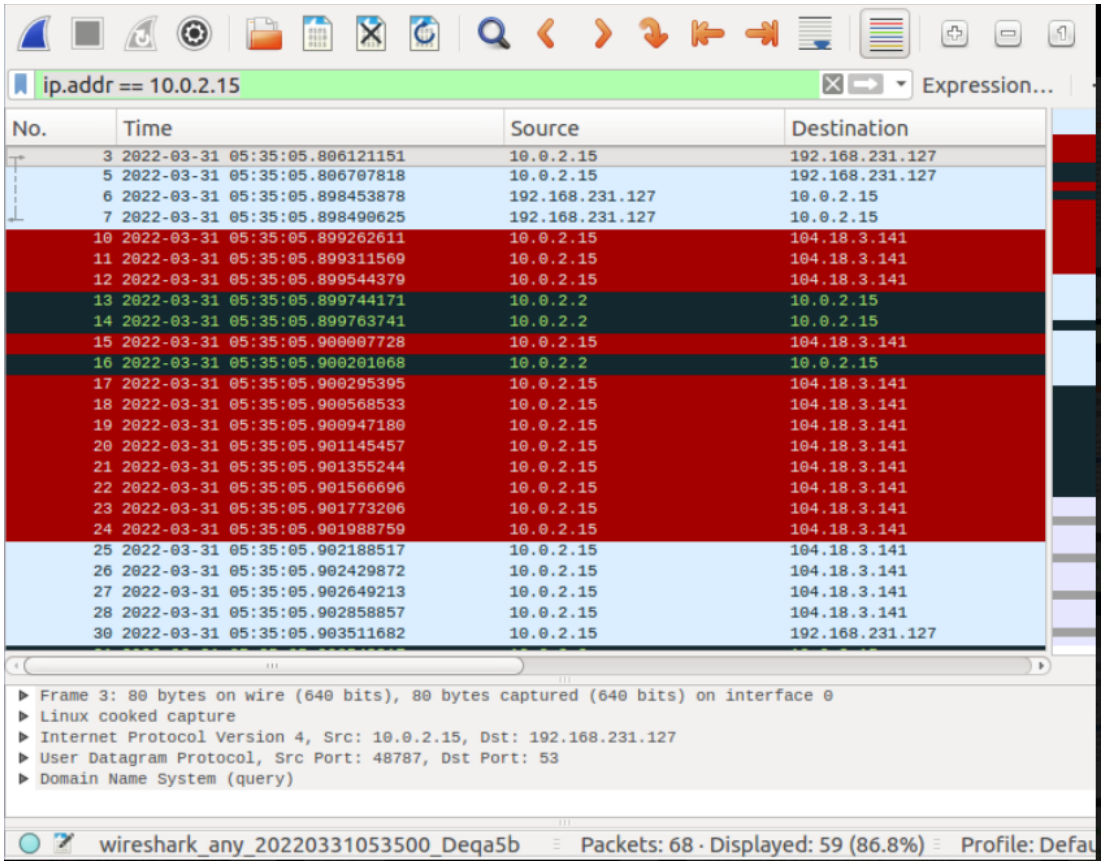
31) In class, we discussed the TTL field and determined that we didn't know a good way to set this. What does your OS set this field to? BTW, please document in this question what your OS and OS version are.



Step 5: ICMP

The Network Layer uses ICMP to send information about the network. Some would say that ICMP is a higher-layer protocol, as the actual ICMP packet is carried inside an IP packet. Let's take a look at how that works.

32) Start a new capture, with the display filter showing only packets sent to or from your computer (i.e. "**ip.addr==<ip address>**")



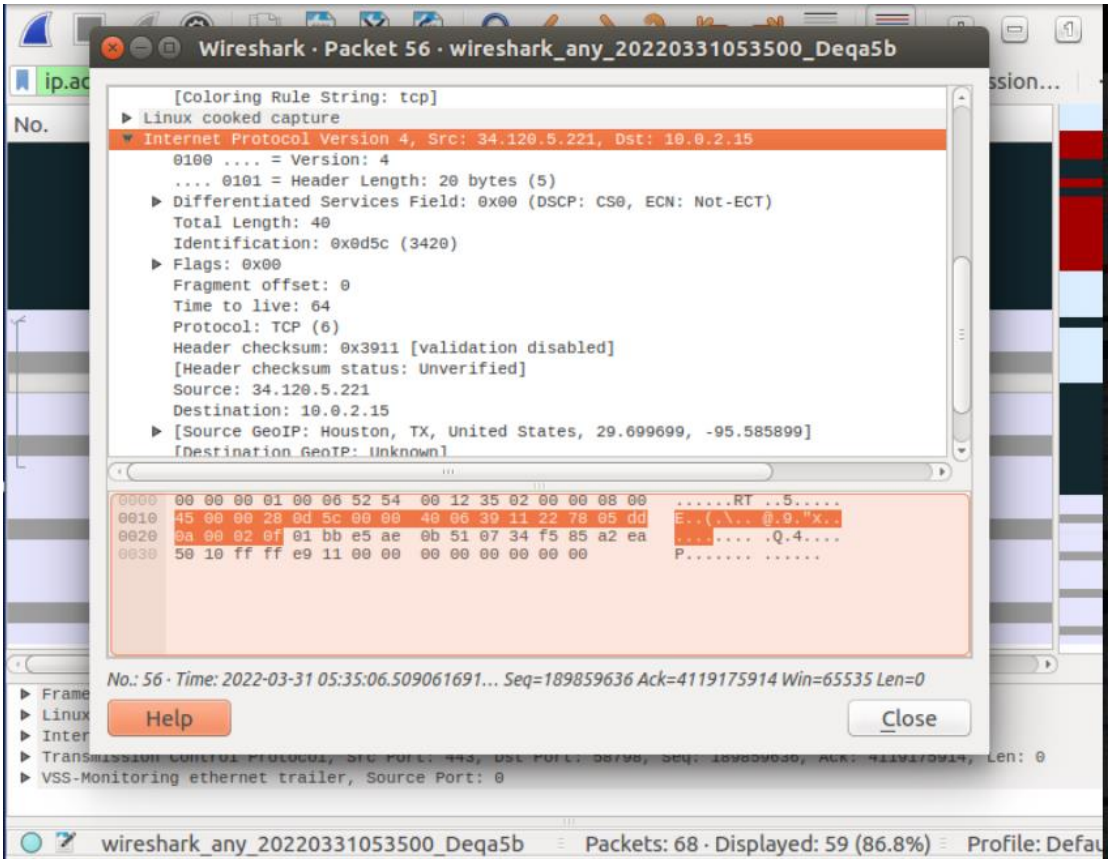
33) In a terminal window, execute the traceroute utility to trace from your computer to **www.cmuj.jp** or **www.regjeringen.no** or some other far-away destination (like we did in our class). If you are having trouble with the weird traceroutes, try this from a non-campus location (your home, a restaurant, etc). Do whatever you can to get a traceroute consisting of about a dozen steps.

34) Stop the capture and take a look at what you found.

35) What are the transmitted segments like? Describe the important features of the segments you observe. In particular, examine the destination port field. What characteristics do you observe about this port number and why would it be chosen so?

- **The destination port was found to be 53.**

36) What about the return packets? What are the values of the various header fields?



37) The ICMP packets carry some interesting data. What is it? Can you show the relationship to the sent packets?

- **The packets carry ICMP in the data section.**

38) Lab1 asserted that ping operates in a similar fashion to traceroute. What differences and similarities are there between the network traffic of ping versus traceroute?

- **Traceroute causes higher network traffic as it tracks the entire path of the packet, which requires multiple pings.**

Finishing up

The report should consist of proper explanation for each question and all the necessary screenshots.

Plagiarism Check

This lab report needs to be uploaded to the repository link going to be provided by the librarian for plagiarism check. The link will be shared very soon. The plagiarism report will then be submitted to your Edmodo Classroom of your respective sections.