

LAB: 05

Lab 5 – Understanding Transport and Network Layer using Wireshark

NAME: LIKHITH R

SECTION: K

SRN: PES1UG20CS659

ROLL NO: 12

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.

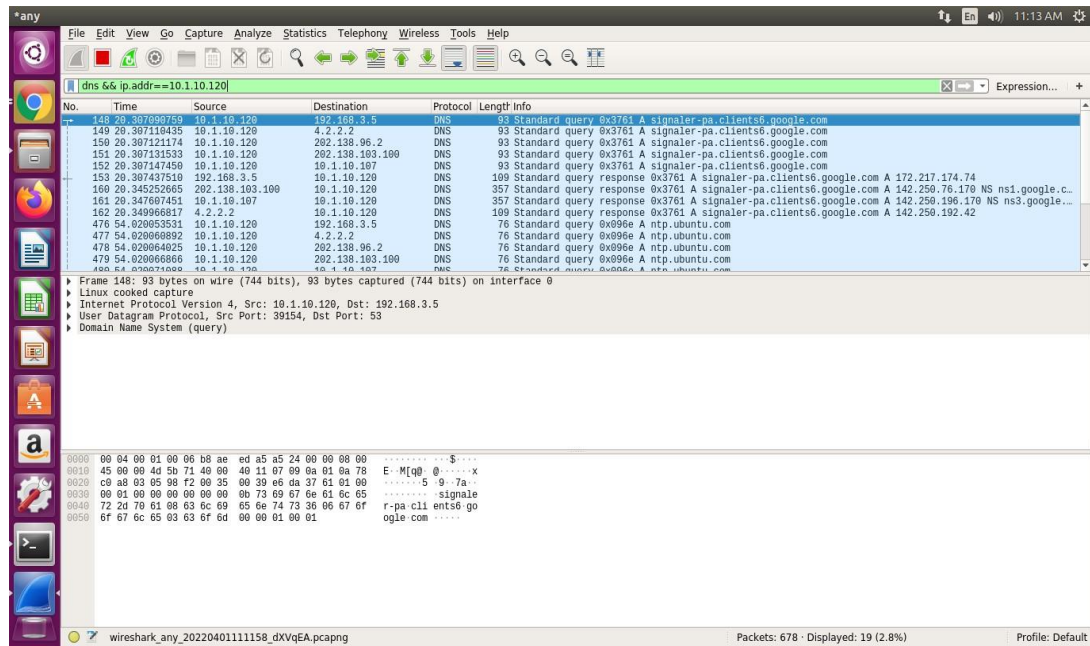
Step 1: UDP and DNS

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>**).

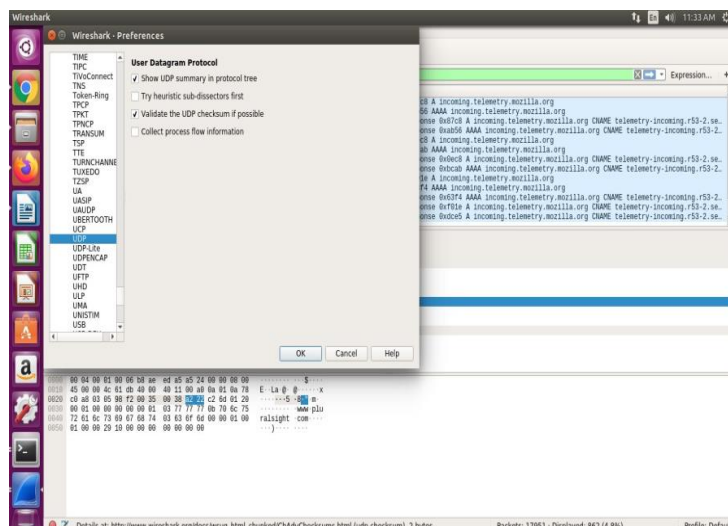
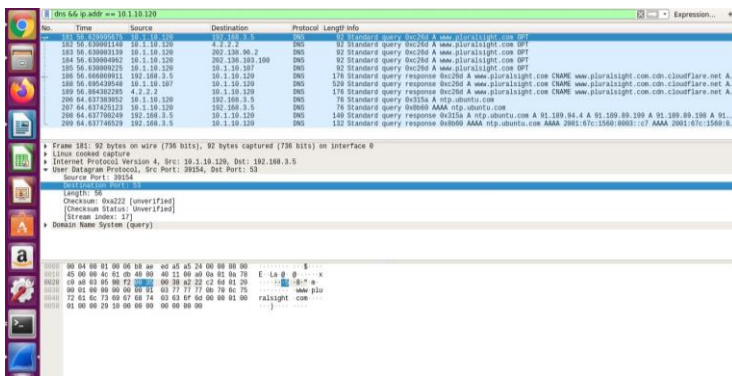
My system ip address is: 10.0.2.15



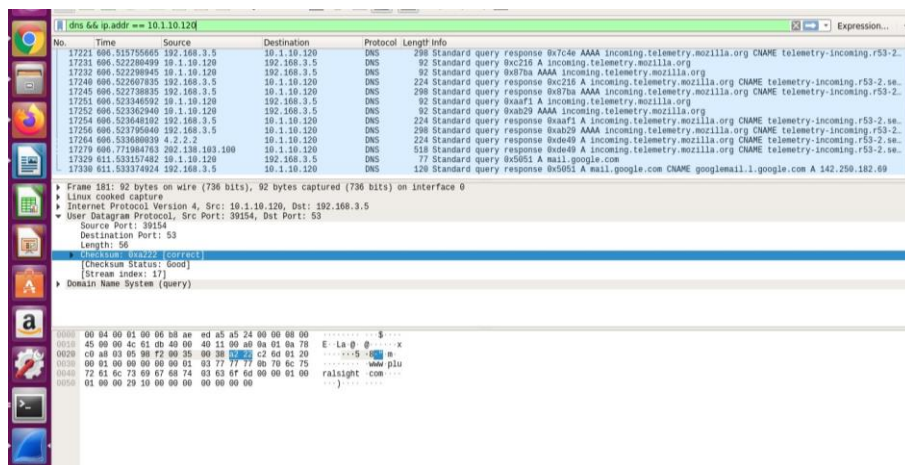
- 2) Use dig to generate a DNS query to lookup the domain name “www.pluralsight.com”. Then, stop the capture.



- 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.
- 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?
- 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?



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



If the UDP packet is fragmented, its checksum cannot be calculated unless it's reassembled, so Wireshark can't verify the checksum.

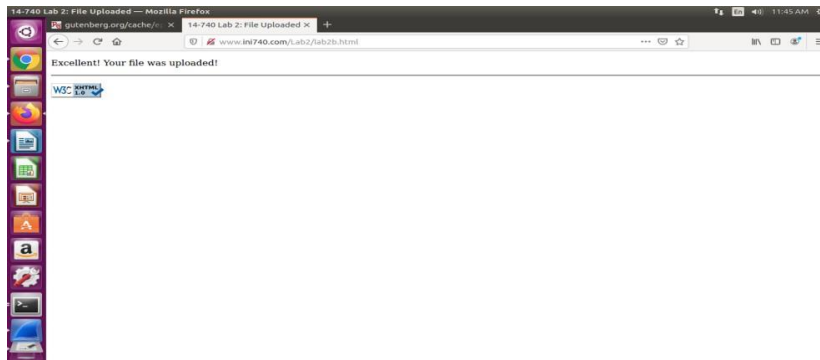
If the checksum is reported as incorrect, either it really *is* incorrect, or it's a packet sent by the machine on which the capture was done and the network adapter is doing checksum offloading (so that the copy of the packet handed to the capture program hasn't had the checksum set), or there's a bug in Wireshark.

In here, checksum showed incorrect as it is specified that it is caused by “UDP CHECKSUM OFFLOAD”

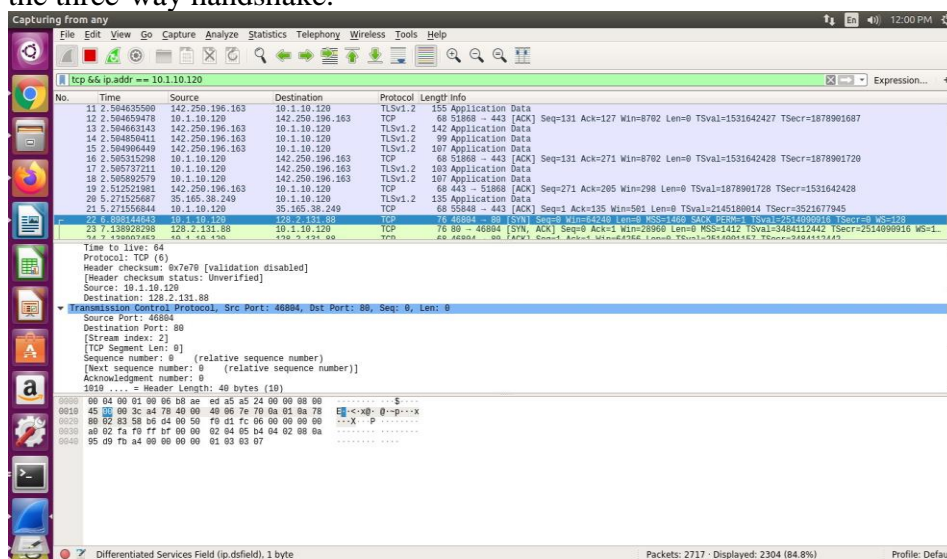
Step 2: TCP

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.



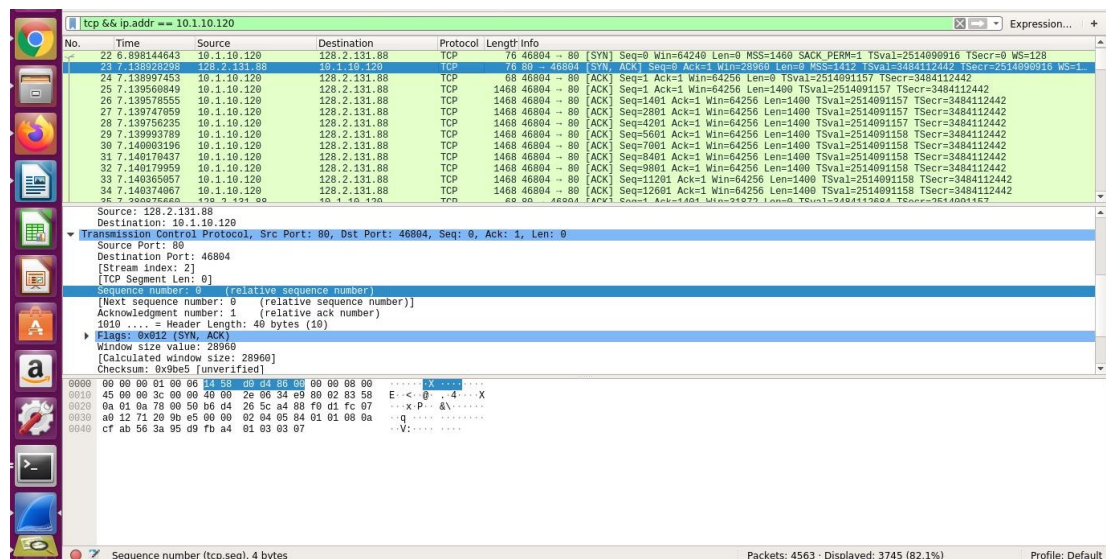
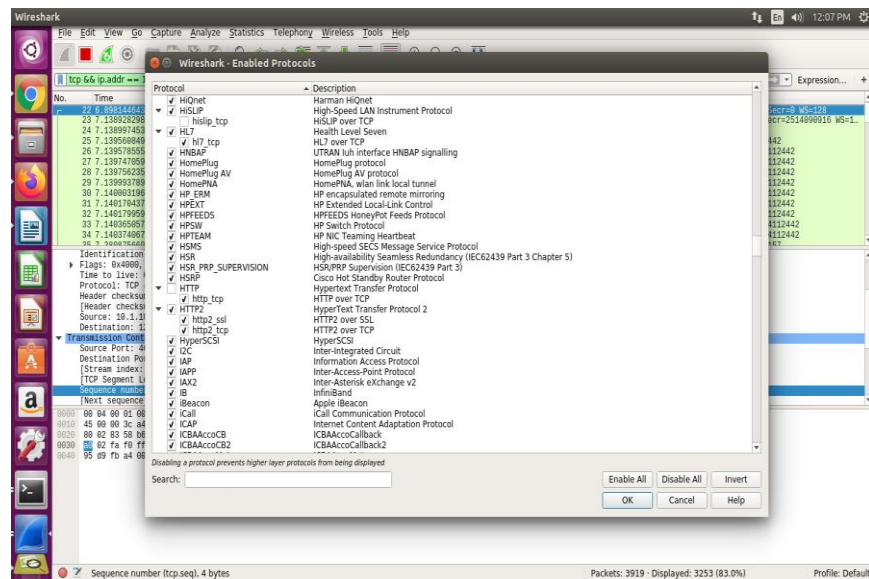
- 11) Let's look at what you captured. First, filter the results to look for TCP packets and to only look at those going to and from your computer with the filter "**tcp && ip.addr == <your IP address>**". If you have other services running on your computer, you might want to further filter so you only 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.



- 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?

IP ADDRESS OF CLIENT: 128.2.131.88 SOURCE
 PORT: 46804
 IP ADDRESS OF SERVER: 10.1.10.120
 DESTINATION PORT: 80

- 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:



?ile Edit View Go Captu re Analyze Stat is t ics Tele phony Tools He Ip

6 Filre r. tcp && ip.addr== T28.2.T 3 0.48

@ Ex press io n... Clear . Apply

29.2.060732	128.2.130.48	128.2.129.29	TCP 53016 -> http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSV=116
30.2.060954	128.2.130.48	128.2.129.29	TCP 53016 -> http [PSH, ACK] Seq=1 Ack=1 Win=524280 Len=1078

Destination port: 53016 (53016)

Sequence number: 0 (relative sequence number)

Acknowledgement number: 1 (relative ack number)

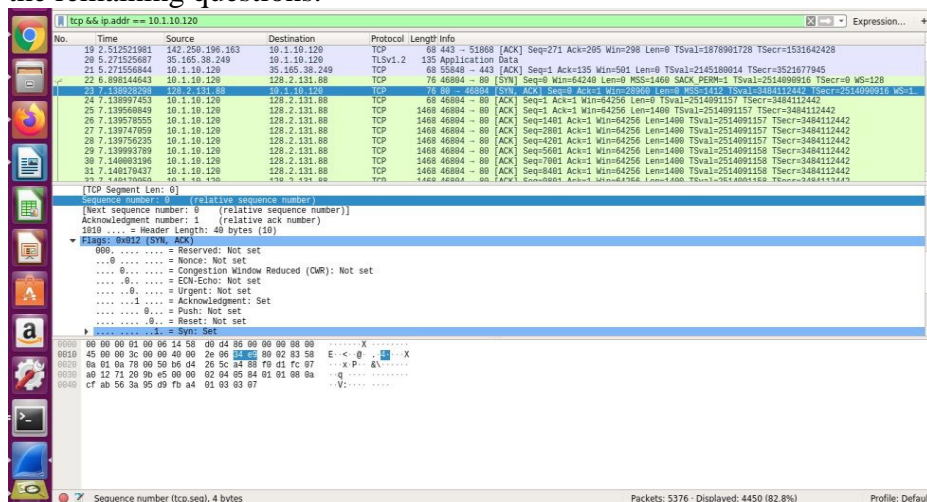
0... = Push: Not set

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.



As we can see, SYN is set.

The sequence number of the TCP SYN segment is 0 since it is used to imitate the TCP connection between the client computer and gaia.cs.umass.edu. According to above figure, in the Flags section, the Syn flag is set to 1 which indicates that this segment is a SYN segment

- 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 (tcp.seq), 4 bytes

SEQUENCE NUMBER: 0

Sequence number (tcp.seq), 4 bytes

The value of the acknowledgement field in the SYNACK segment is 1.

The value of the ACKnowledgement field in the SYNACK segment is determined by the server. The server adds 1 to the initial sequence number of SYN segment from the client computer. For this case, the initial sequence number of SYN segment from the client computer is 0, thus the value of the ACKnowledgement field in the SYNACK segment is 1

A segment will be identified as a SYNACK segment if both SYN flag and Acknowledgement in the segment are set to 1.

- 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.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	192.168.1.8	128.119.245.12	TCP	78	60706 > http [SYN] Seq=0 win=65535 Len=0 MSS=1460 WS=16
4	0.26949200	128.119.245.12	192.168.1.8	TCP	74	http > 60706 [SYN, ACK] Seq=0 Ack=1 win=5792 Len=0 MSS=1
5	0.26960900	192.168.1.8	128.119.245.12	TCP	66	60706 > http [ACK] Seq=1 Ack=1 win=131760 Len=0 TSval=85
6	0.27125700	192.168.1.8	128.119.245.12	TCP	644	60706 > http [PSH, ACK] Seq=1 Ack=1 win=131760 Len=578
7	0.27142500	192.168.1.8	128.119.245.12	TCP	203	60706 > http [PSH, ACK] Seq=579 Ack=1 win=131760 Len=137
8	0.27179700	192.168.1.8	128.119.245.12	TCP	1514	60706 > http [ACK] Seq=716 Ack=1 win=131760 Len=1448 TSV

Frame 6: 644 bytes on wire (5152 bits), 644 bytes captured (5152 bits) on interface 0

Ethernet II, Src: Apple1f:d4:56 (b8:e8:56:1f:d4:56), Dst: Tp-LinkT_f8:6d:f9 (a0:f3:c1:f8:6d:f9)

Internet Protocol Version 4, Src: 192.168.1.8 (192.168.1.8), Dst: 128.119.245.12 (128.119.245.12)

Transmission Control Protocol, Src Port: 60706 (60706), Dst Port: http (80), Seq: 1, Ack: 1, Len: 578

Source port: 60706 (60706)

Destination port: http (80)

[Stream index: 0]

Sequence number: 1 (relative sequence number)

[Next sequence number: 579 (relative sequence number)]

Acknowledgment number: 1 (relative ack number)

Header length: 32 bytes

Flags: 0x018 (PSH, ACK)

000. = Reserved: Not set

...0 = Nonce: Not set

....0 = Congestion Window Reduced (CWR): Not set

....0 = ECN-Echo: Not set

....0 = Urgent: Not set

....1 = Acknowledgment: Set

....1 = Push: Set

....0 = Reset: Not set

0000 a0 f3 c1 f8 6d f9 b8 e8 56 1f d4 56 08 00 45 00 ...m... V..V..E.

0010 02 76 f6 5a 40 00 40 06 0a f3 c0 a8 01 08 80 77 ..V.Z@.0.W

0020 f5 0c ed 22 00 50 1f e9 a7 e8 79 47 80 0a 80 18 ...".P... ..YG...

0030 20 2b bf 08 00 00 01 01 08 0a 05 16 f8 ee 86 ca +..... ..

0040 ee 56 50 4f 53 54 20 2f 7f 69 72 65 73 68 61 72 ..VPOST / wireshar

0050 6b 2d 6c 61 62 73 2f 6c 61 62 33 2d 31 2d 72 65 k-rs5/1 ab3-l-re

0060 70 6c 79 2e 68 74 6d 20 48 54 54 50 2f 31 2e 31 ply.htm HTTP/1.1

0070 0d 0a 48 6f 73 74 3a 20 67 61 69 61 2e 63 73 2e ..Host: gaia.cs.

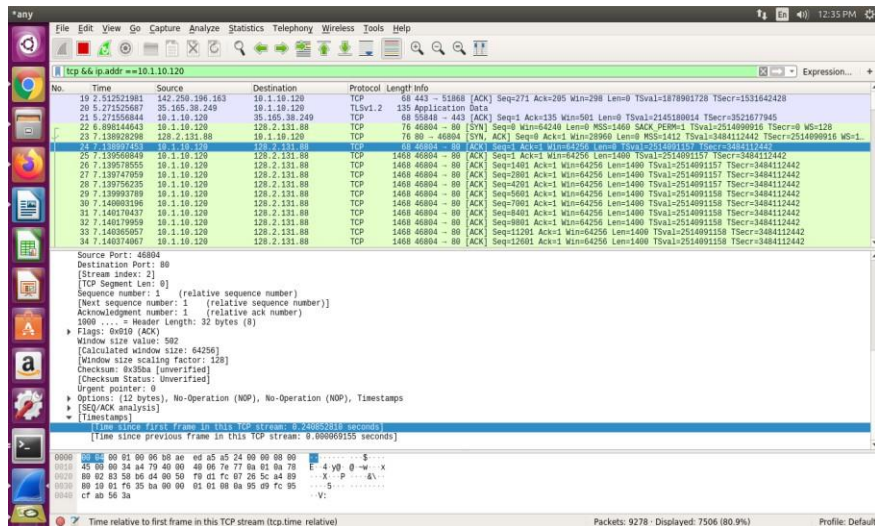
0080 75 6d 61 73 73 2e 65 64 75 0d 0a 43 6f 6e 74 65 umass.ed u.Conte

0090 6e 74 2d 54 79 70 65 3a 20 6d 75 6c 74 69 70 61 nt-Type: multipa

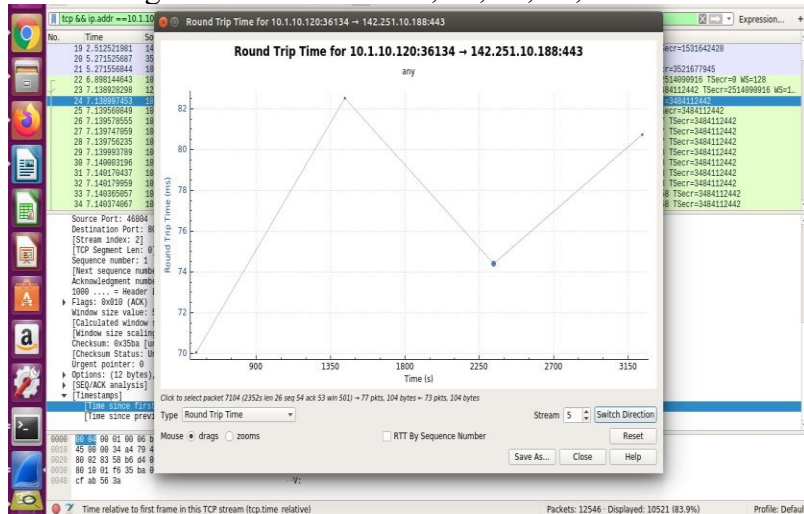
00a0 72 74 2f 66 6f 72 6d 2d 64 61 74 61 3b 20 62 6f rt/Form-data; bo

According to above figure, the segment No.6 contains the HTTP POST command, the sequence number of this segment is 1.

- 17) Consider the TCP segment containing the HTTP POST as the first segment in the non-overhead part of the TCP connection. For the segments which follow, put together a table with one row per segment (and columns for whatever data you think is useful) until you have enough segments to calculate four SampleRTT values according to the RTT estimation techniques discussed in class. Calculate what those SampleRTT values are, as well as the EstimatedRTT after each Sample is collected. Discuss this calculation, including what your initial EstimatedRTT was, your choice of parameters, and any segments that weren't used in the calculation. 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.



According to above figures, the segments 1-6 are No. 24,25,26,27,28 and 29. The ACK of segments 1-6 are No. 10, 12, 13, 16, 19 and 22.



	Sent time
SEGMENT 1	0.226361740
SEGMENT 2	0.227254542
SEGMENT 3	0.227486752
SEGMENT 4	0.227693336
SEGMENT 5	0.227889134
SEGMENT 6	0.228106280

According to the formula: EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * SampleRTT


```

Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  No-Operation (NOP)
  No-Operation (NOP)
  Timestamps: TSval 85391598, TSecr 2261446230
    Kind: Timestamp (8)
    Length: 10
    Timestamp value: 85391598
    Timestamp echo reply: 2261446230
[SEQ/ACK analysis]
Data (578 bytes)
  Data: 504f5354202f77697265736861726b2d6c6162732f6c6162...
  [Length: 578]

```

The length of the first TCP segment is 578 bytes, the length of the second TCP segment is 137 bytes. The length of each of the following five TCP segments is 1448 bytes.

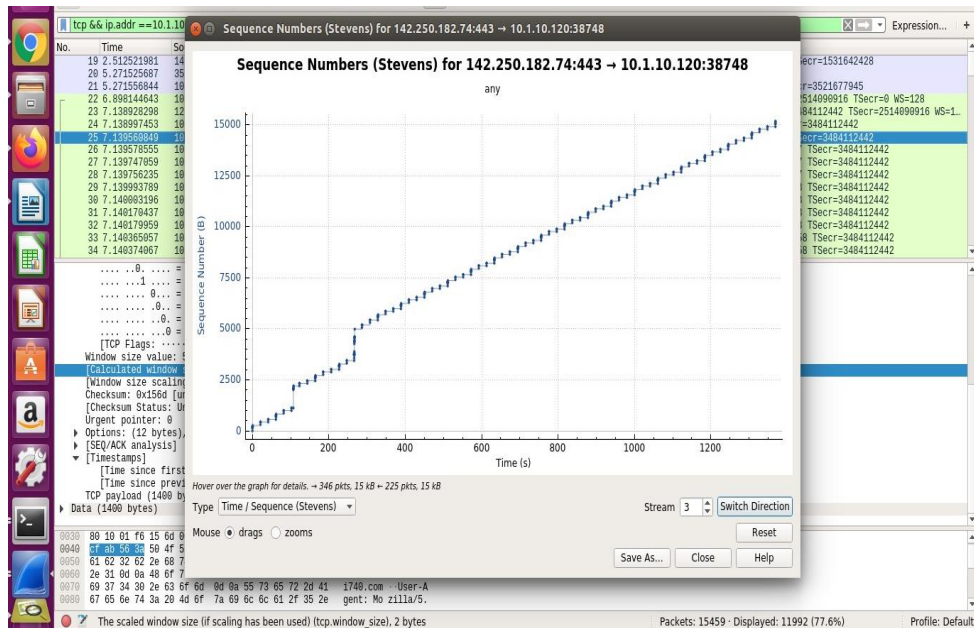
- 18) What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

The screenshot shows a Wireshark packet capture of a TCP connection. The packet list pane displays several segments. The first segment (No. 26) is a SYN segment with a length of 137 bytes and a window size of 502. The subsequent segments (Nos. 27-34) are data segments with a length of 1448 bytes and a window size of 64256. The packet details pane for the selected segment (No. 26) shows the 'Window size value: 502' and 'Calculated window size: 64256'. The packet bytes pane shows the data payload starting with 'POST /lab2/1'.

The minimum amount of available buffer space advertised at the receiver for the entire trace is indicated first ACK from the server, its value is 64256 bytes.

This receiver window grows until it reaches the maximum receiver buffer size of 62780 bytes. According to the trace, the sender is never throttled due to lack of receiver buffer space.

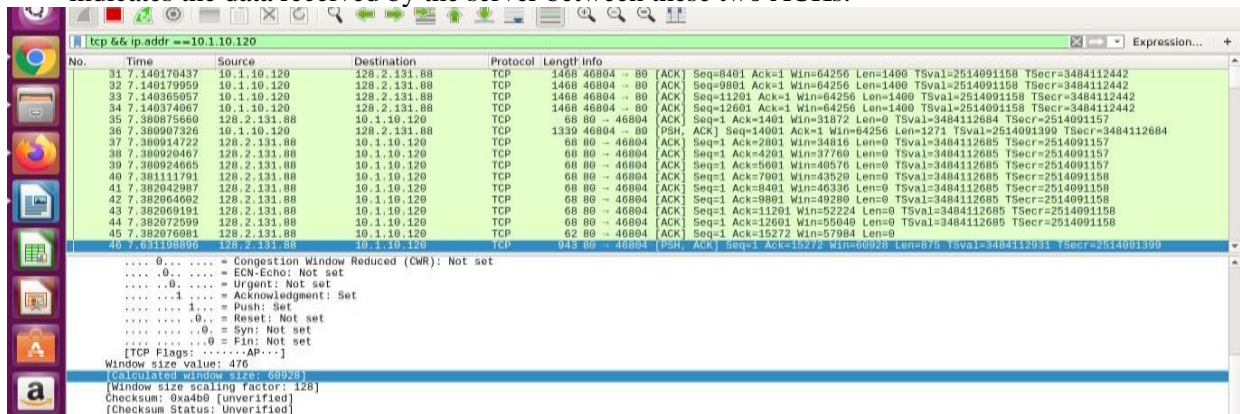
- 19) Are there any retransmitted segments? What did you check for (in the trace) to answer this question?



There are no retransmitted segments in the trace file since in the time sequence graph (stevens), all sequence numbers are monotonically increasing.

- 20) How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is delayed ACKing segments? Explain how or why not.

The difference between the acknowledged sequence numbers of two consecutive ACKs indicates the data received by the server between these two ACKs.



The receiver is ACKing every other segment. For example, segment of No. 85 acknowledged data with 1430 bytes.

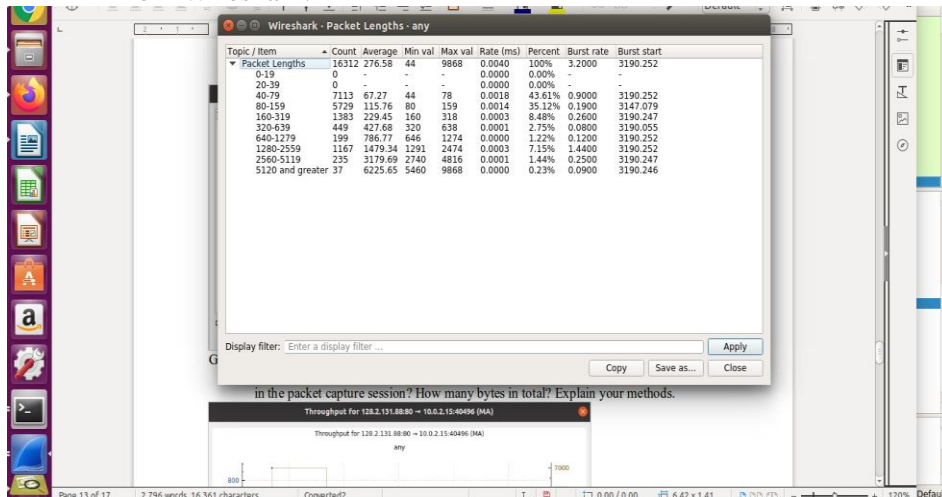
- 21) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

The cantebury.txt on the hard drive is 152,138 bytes and the download time is 1.578736000 (First TCP segment) - 0.271257000 (last ACK) = 1.307479 second. Therefore, the throughput for the TCP connection is computed as 152,138/1.307479=116359.803867 bytes/second.

Step 2c: Statistics

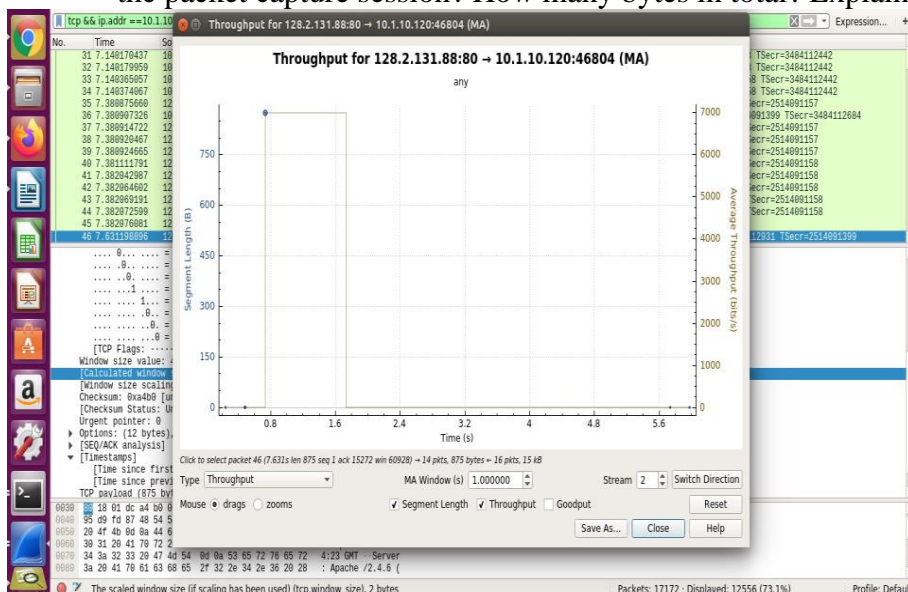
Wireshark has some fairly robust reporting abilities, most of which are accessed via the Statistics menu. Spend a few minutes messing around with the options on that menu, trying to figure out what each report is telling you. Then, answer the following questions about the Canterbury Tales capture:

- 22) What is the most common TCP packet length range? What is the second most common TCP packet length range? Why is the ratio of TCP packets of length < 4 bytes equal to zero? Describe what actions you took to get answers to these questions from Wireshark.



Go to statistics->packet lengths ... to get the desired solution

- 23) What average throughput did you use in Mbps? How many packets were captured in the packet capture session? How many bytes in total? Explain your methods.



Average throughput: 7000 bits/sec

Protocol	Frame	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Linux cooked-mode capture		100.0	12760	100.0	380.0	421027	7455	0	0
Internet Protocol Version 4		100.0	12760	6.3	25320	473	6387	0	1.817
Transmission Control Protocol		100.0	12760	88.6	3561661	6.612	6387	0	1.817
VSS-Monitoring ethernet trailer		0.0	1	0.0	2	0	2	0	0
Secure Socket Layer		0.0	6378	76.6	2162284	5.870	6360	126335	5.804
Data		0.1	12	0.4	16146	29	12	16146	29

Total 7654 packets were captured. 4020775 bytes in total.

24) 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?

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.1.10.120	55848	151.101.153.16	80	443	2,910,295	1,455	98 k	1,455	196 k	2.272244	4363.1243	181	53
10.1.10.120	51868	142.250.196.163	443	194	18 k	95	10 k	99	10 k	2.464861	1320.7130	53	53
10.1.10.120	46804	128.2.131.88	80	30	18 k	16	16 k	14	1.829	6.898145	5.9738	21 k	21 k
10.1.10.120	38748	142.250.182.74	443	571	69 k	225	30 k	346	38 k	9.936097	1368.9321	181	181
10.1.10.120	38746	142.250.182.74	443	392	38 k	196	20 k	196	17 k	9.939519	1369.1868	122	122
10.1.10.120	36134	142.250.196.14	443	199	13 k	102	13 k	97	6.726	15.392016	4317.2695	13	13
10.1.10.120	56220	142.250.196.14	443	105	22 k	51	13 k	54	9.085	20.149690	371.3048	299	299
10.1.10.120	51664	142.250.205.234	443	8	617	4	272	4	345	21.504016	58.6423	37	37
10.1.10.120	43748	74.125.200.189	443	271	32 k	129	16 k	142	16 k	22.220414	1393.1911	93	93
10.1.10.120	51176	142.250.182.69	443	54	17 k	24	6.906	30	10 k	35.839974	322.4105	171	171
10.1.10.120	53600	142.250.183.138	443	14	1,025	7	476	7	549	39.936007	193.0583	19	19
10.1.10.120	41356	52.41.2.143	443	60	5,280	24	2,268	36	3,012	149.360568	4203.7785	4	4
10.1.10.120	56254	142.250.196.14	443	60	13 k	31	8,339	29	5,046	437.989540	253.4793	263	263
10.1.10.120	51202	142.250.182.69	443	40	8,308	20	4,941	20	3,367	453.712171	240.6024	164	164
10.1.10.120	51204	142.250.182.69	443	40	8,480	20	4,752	20	3,728	717.769072	240.5328	158	158
10.1.10.120	37174	172.217.163.202	443	25	5,954	12	2,375	13	3,579	724.339637	0.1481	128 k	128 k
10.1.10.120	43752	172.217.174.238	443	87	19 k	43	11 k	44	7,853	733.92772	262.6149	358	358
10.1.10.120	49794	142.250.77.42	443	41	7,401	20	3,655	21	3,746	896.096665	240.5389	121	121
10.1.10.120	47900	172.217.174.238	443	308	68 k	155	41 k	153	26 k	1038.009588	1153.5800	287	287
10.1.10.120	60256	34.120.5.221	443	27	3,804	16	2,147	11	1,657	1047.074120	170.0714	100	100
10.1.10.120	60418	142.250.183.37	443	42	4,465	20	4,906	22	3,559	1051.721837	240.6477	79	79
10.1.10.120	60276	142.250.182.195	443	38	5,000	18	2,378	20	2,622	1288.208572	240.4727	197	197
10.1.10.120	60092	172.217.27.202	443	1,279	159 k	512	74 k	767	85 k	1379.118234	2979.1161	163	163
10.1.10.120	55562	142.250.183.170	443	38	7,621	18	3,012	20	4,609	1387.805677	240.2155	100	100
10.1.10.120	38450	142.250.183.42	443	37	5,288	18	2,496	20	2,792	1401.256302	240.4249	83	83
10.1.10.120	38452	142.250.183.42	443	40	7,193	20	3,662	20	3,531	1401.649700	240.5181	121	121
10.1.10.120	57430	142.251.12.189	443	581	72 k	278	36 k	303	36 k	1415.378048	2937.4171	99	99
10.1.10.120	60102	172.217.27.202	443	834	84 k	411	45 k	423	39 k	1423.961013	2913.6226	124	124
10.1.10.120	57492	142.250.199.163	443	503	48 k	274	24 k	229	23 k	1443.203933	2882.5305	68	68
10.1.10.120	37122	142.250.182.14	443	37	5,267	19	2,571	18	2,696	1639.457326	240.6646	85	85

To host 151.101.153.16, my local host converse the most as it sent total of 2682k bytes. 367 packets were sent from my host and 454 packets were sent from the remote host.

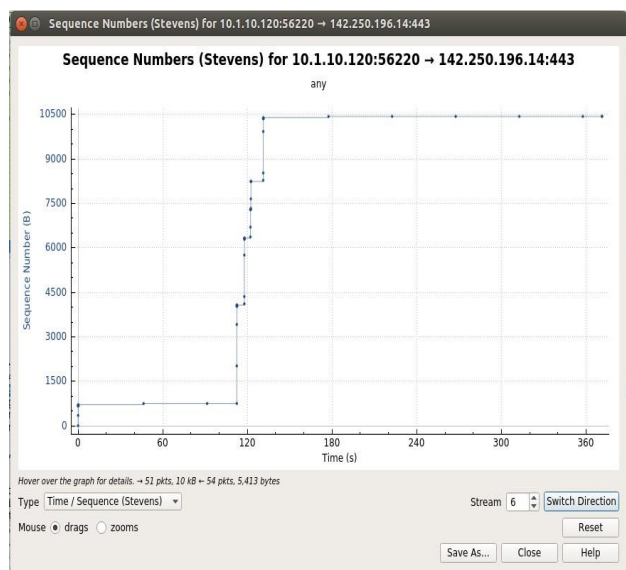
Step 3: Congestion Control

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? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text. Make sure to include a copy of the plot in your report.



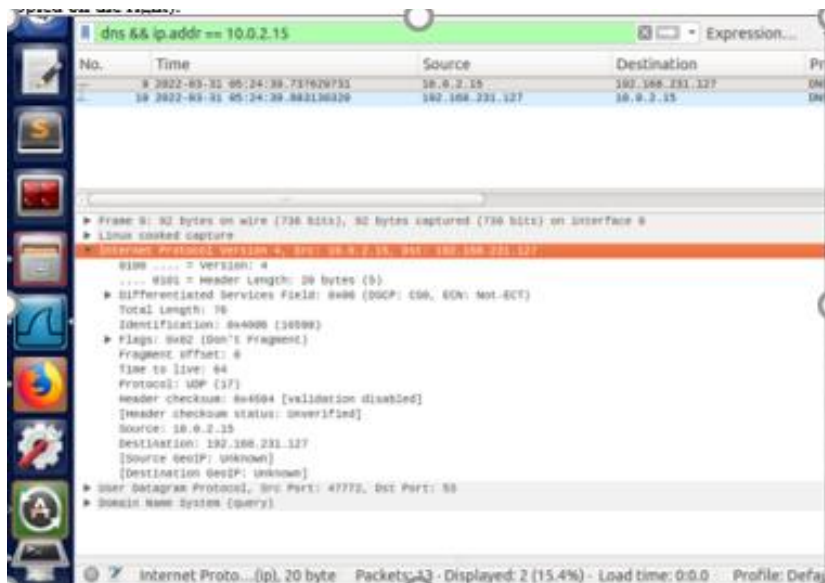
Slowstart phase begins at 0s and ends at 0.212s and congestion avoidance takes place at 0th segment.

Step 4: The Network Layer

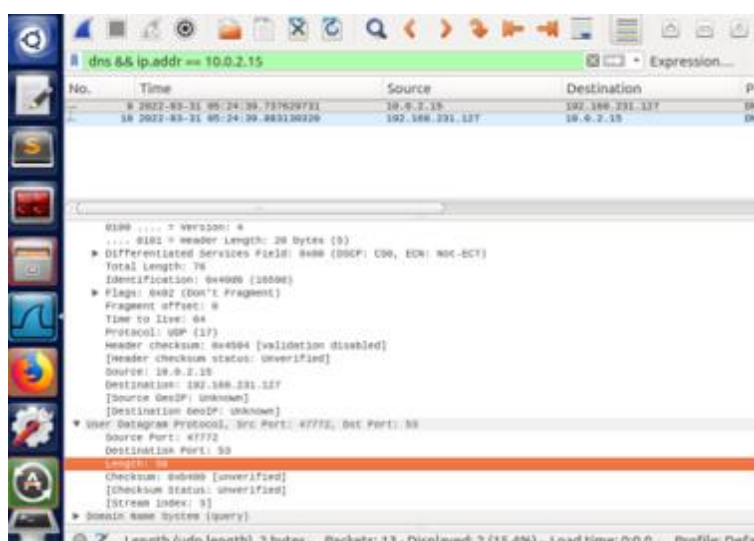
Let's take this opportunity to check out a bit of IP traffic. We don't have to capture any additional traffic, as everything we've seen today is carried over IP packets.

- 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**).

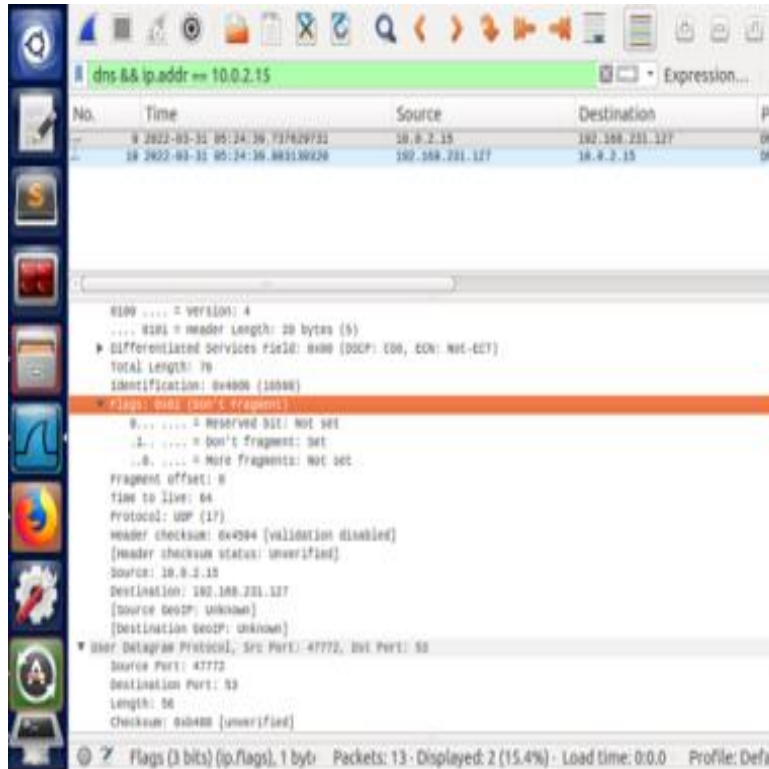
Match up the header fields with the format we discussed in class (don't just look through Wireshark's display -- instead, match the raw bytes with the pictures we saw in lecture, which I've copied on the right).



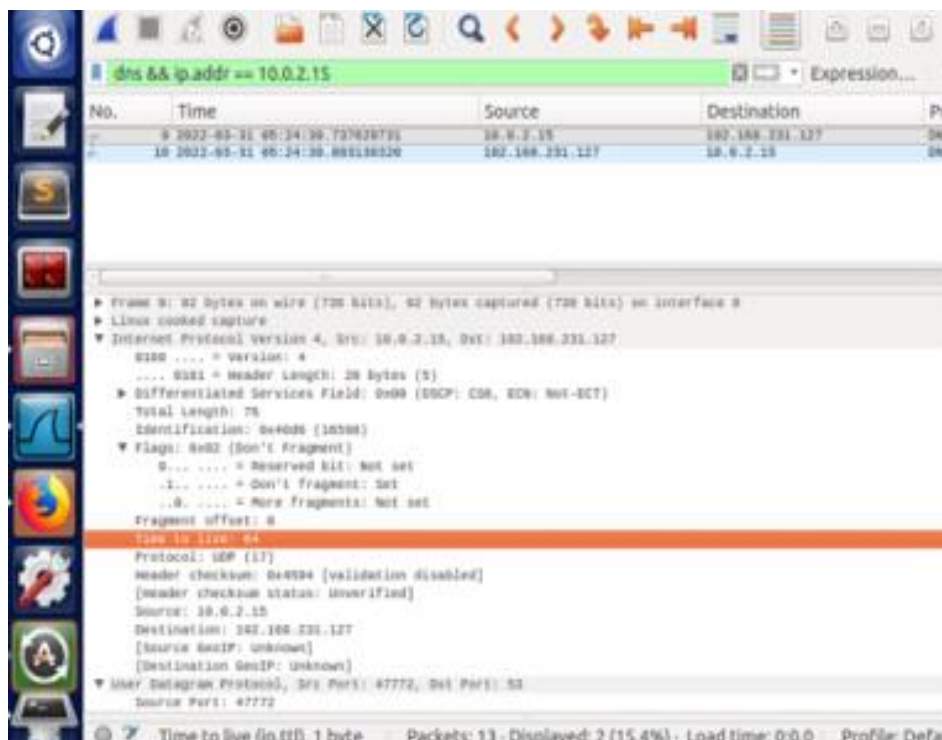
- 29) Most of the fields should match up and make perfect sense. Verify the Datagram Length, Upper-layer protocol and the IP address fields.



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



- 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.

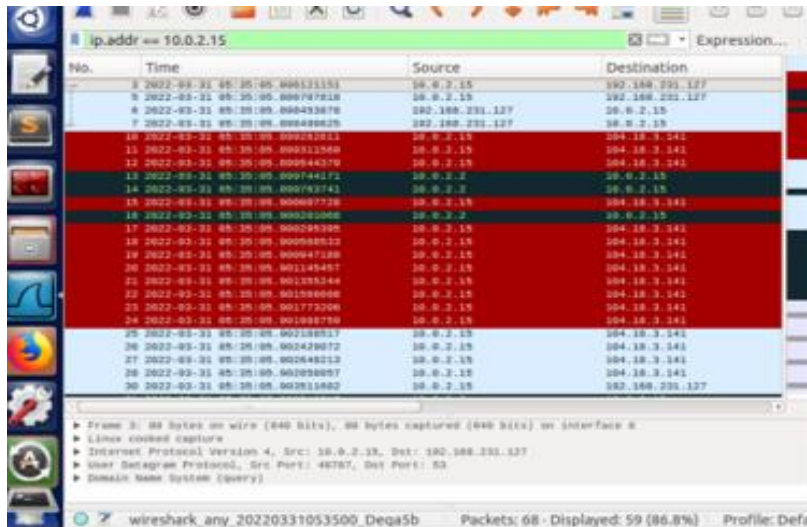


OS: Ubuntu
OS Version:

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>”)



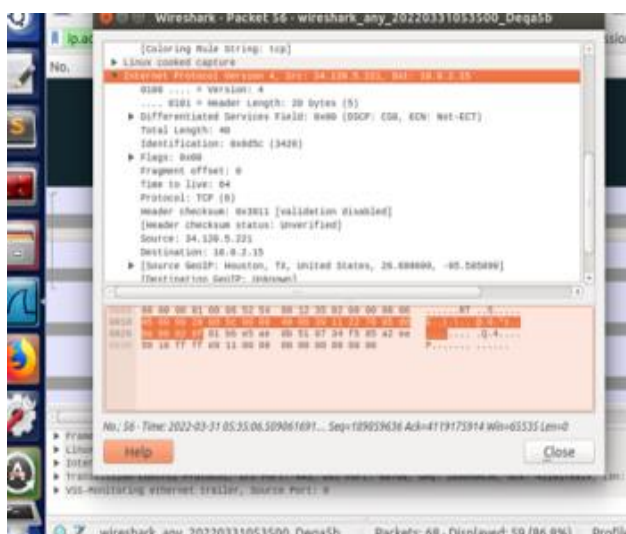
- 33) In a terminal window, execute the traceroute utility to trace from your computer to www.cmuj.jp or www.regjeringen.no or some other far-away destination (like we did in our class). If you are having trouble with the weird traceroutes, try this from a non-campus location (your home, a restaurant,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

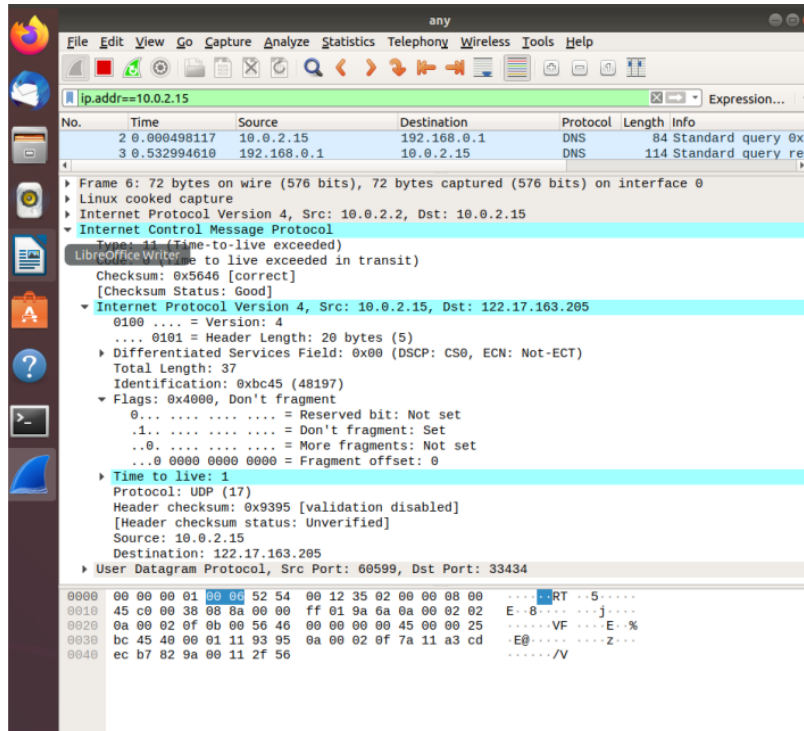
- 35) What are the transmitted segments like? Describe the important features of the segments you observe. In particular, examine the destination port field. What characteristics do you observe about this port number and why would it be chosen so?

Dest Port: 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?



- 38) Lab1 asserted that ping operates in a similar fashion to traceroute. Use Wireshark to show the degree to which this is true. What differences and similarities are there between the network traffic of ping versus traceroute?

