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.

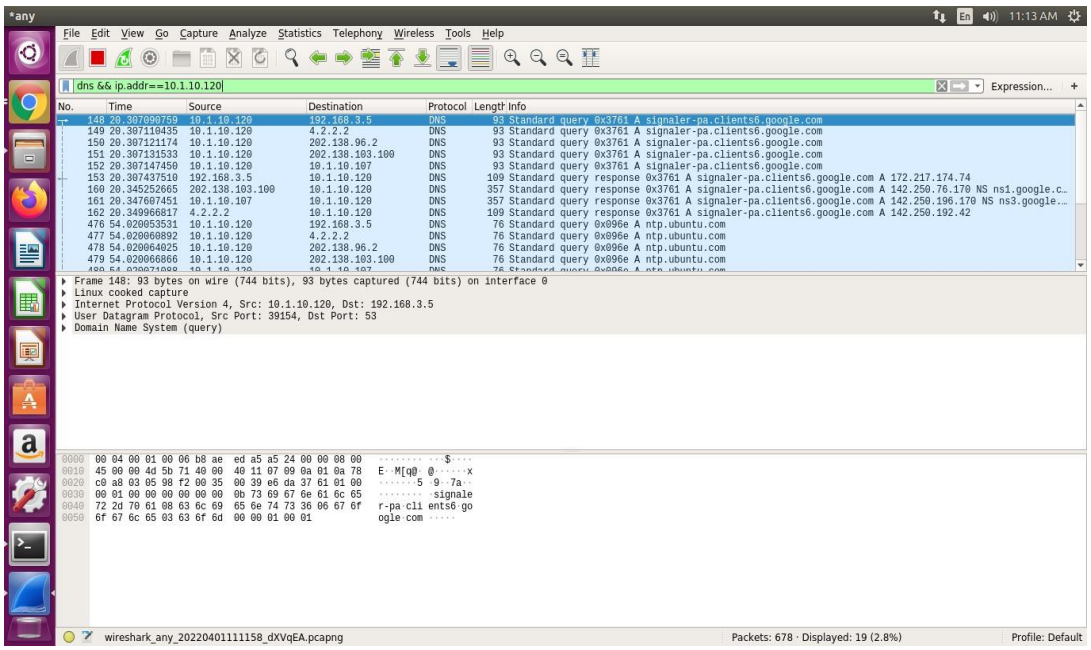
Step 1: UDP and DNS

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

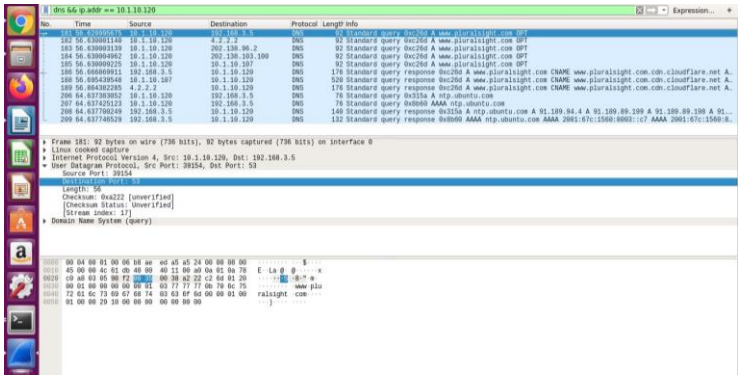


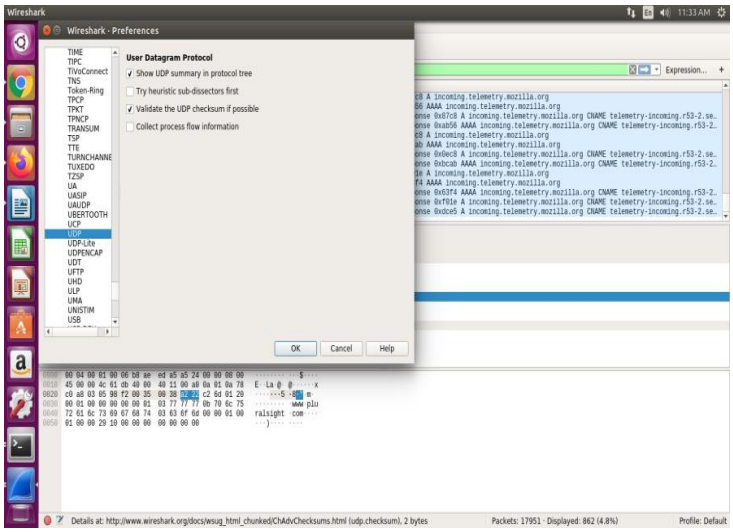
2) Use dig to generate a DNS query to lookup the domain name "**www.pluralsight.com**". Then, stop the capture.

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.

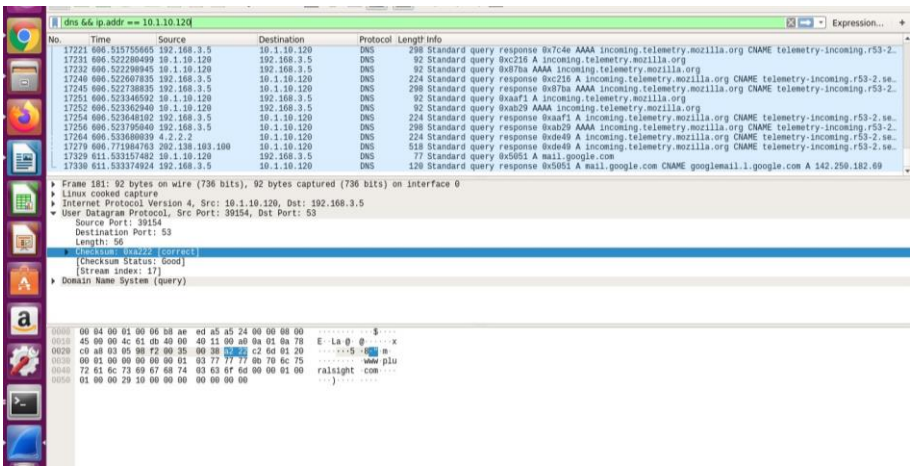
4) Take a look at the query packet on Wireshark. You'll see a bunch of bytes (70-75 bytes) listed as the actual packet contents in the bottom Wireshark window. The bytes at offsets up to number 33-34 are generated by the lower-level protocols. If you click on the "User Datagram Protocol" line in the 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.

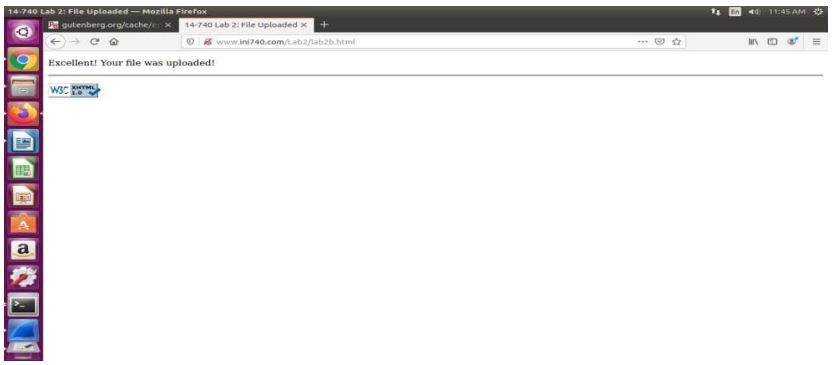
If the checksum is reported as incorrect, either it really *is* incorrect, or it's a packet sent by the machine on which the capture was done and the network adapter is doing checksum offloading (so that the copy of the packet handed to the capture program hasn't had the checksum set), or there's a bug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

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

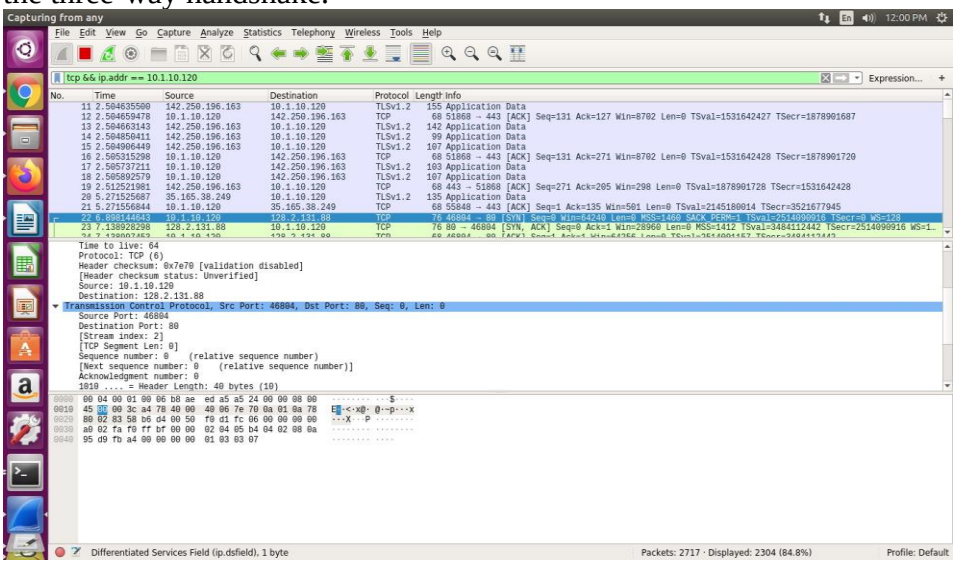
Step 2: TCP

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.



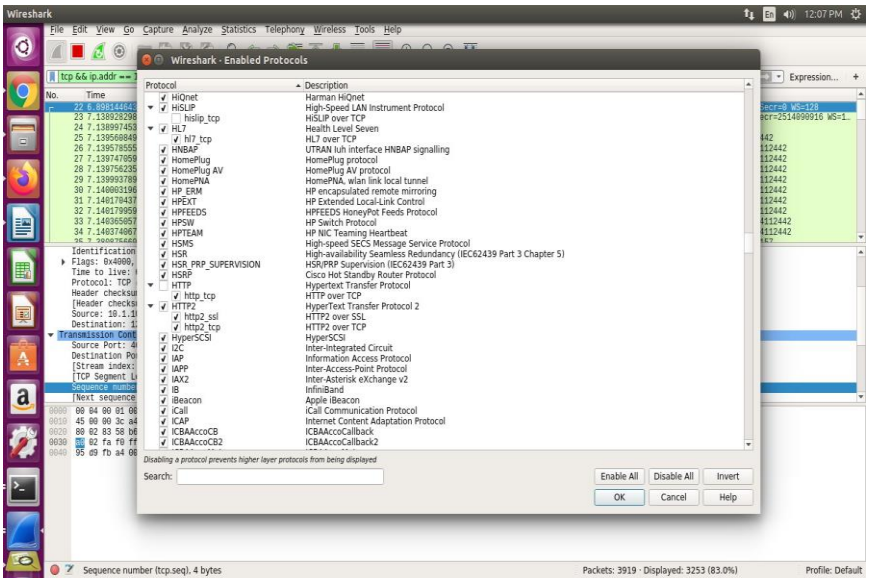
11) Let's look at what you captured. First, filter the results to look for TCP packets and to only look at those going to and from your computer with the filter **"tcp && ip.addr == <your IP address>"**. If you have other services running on your computer, you might want to further filter so you only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

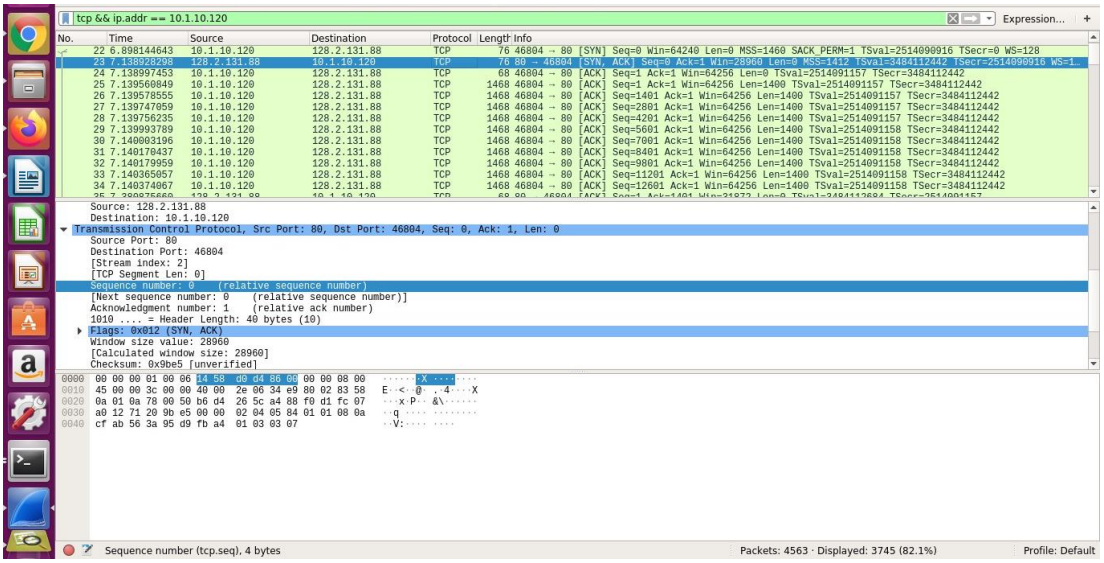


12) What is the IP address and TCP port number used by your computer (client) to transfer the file? What is the IP address of the server? On what port number is it sending and receiving TCP segments for this transfer of the file?

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 lp

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 2.060964	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)

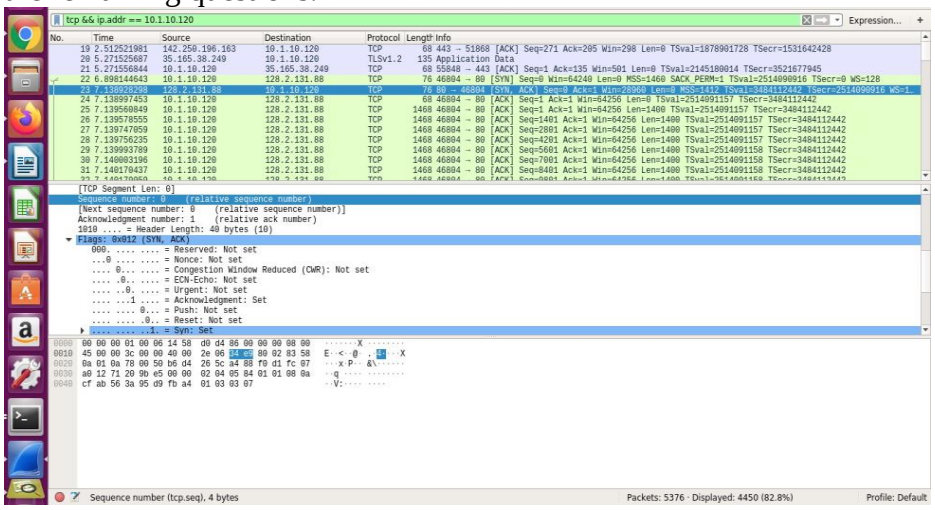
0... = Push: Not set

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

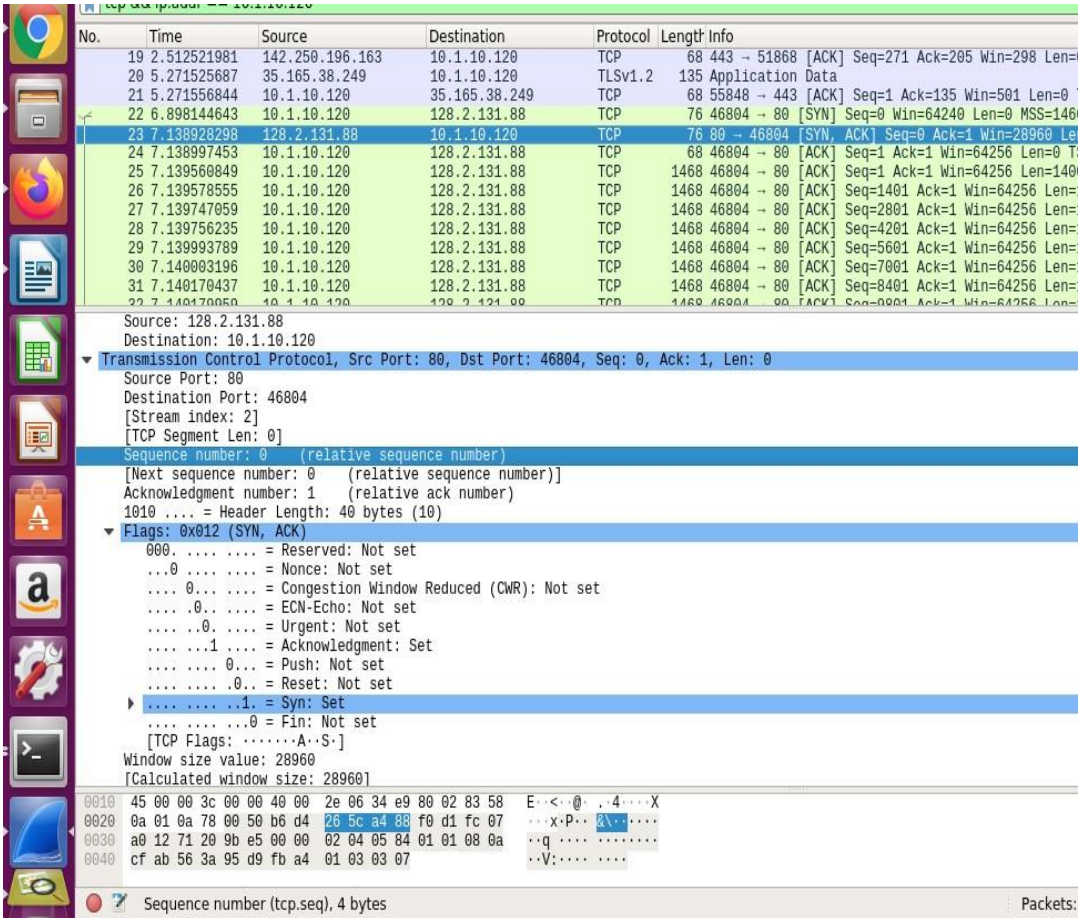
Answer the following questions for the TCP segments:

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.

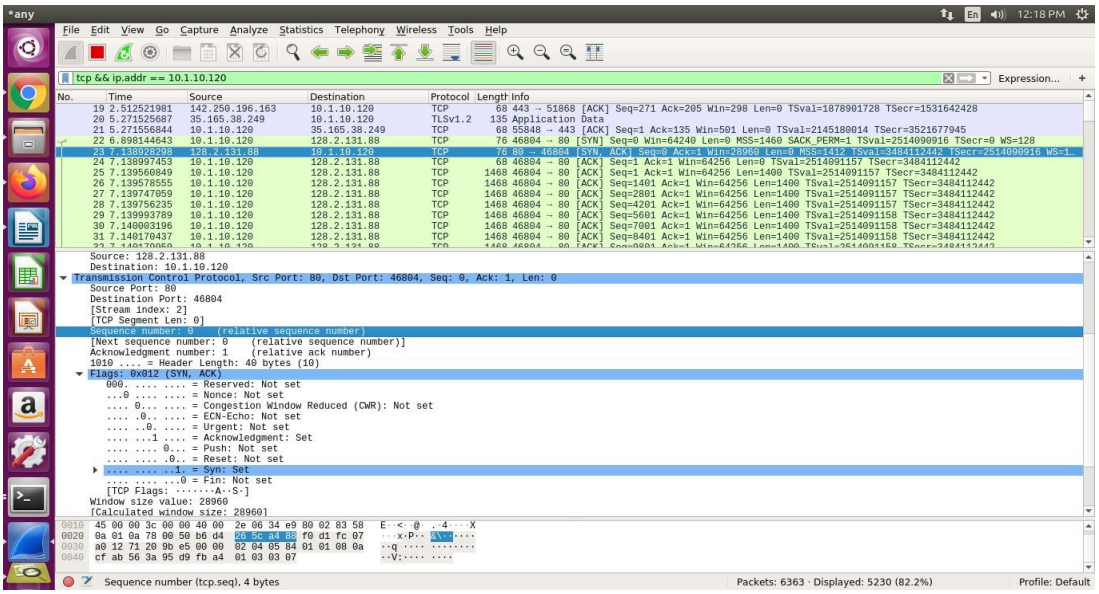


As we can see, SYN is set.
The sequence number of the TCP SYN segment is 0 since it is used to imitate the TCP connection between the client computer and gaia.cs.umass.edu. According to above figure, in the Flags section, the Syn flag is set to 1 which indicates that this segment is a SYN segment

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: 0

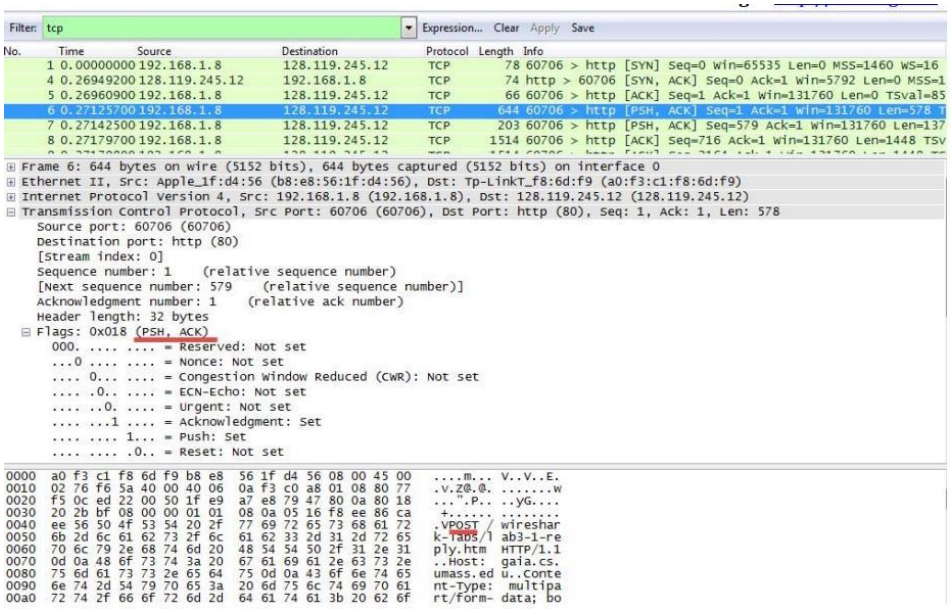


The value of the acknowledgement field in the SYNACK segment is 1.

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 form 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

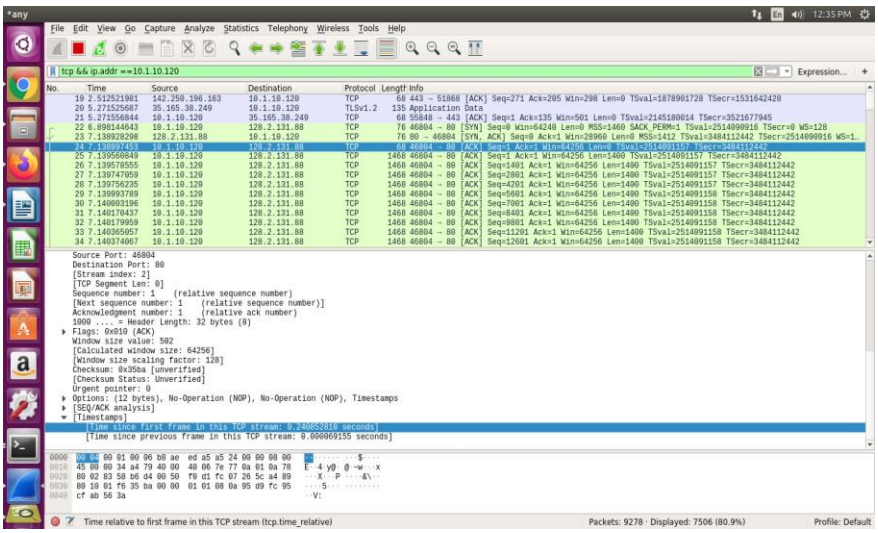
A segment will be identified as a SYNACK segment if both SYN flag and Acknowledgement in the segment are set to 1.

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.

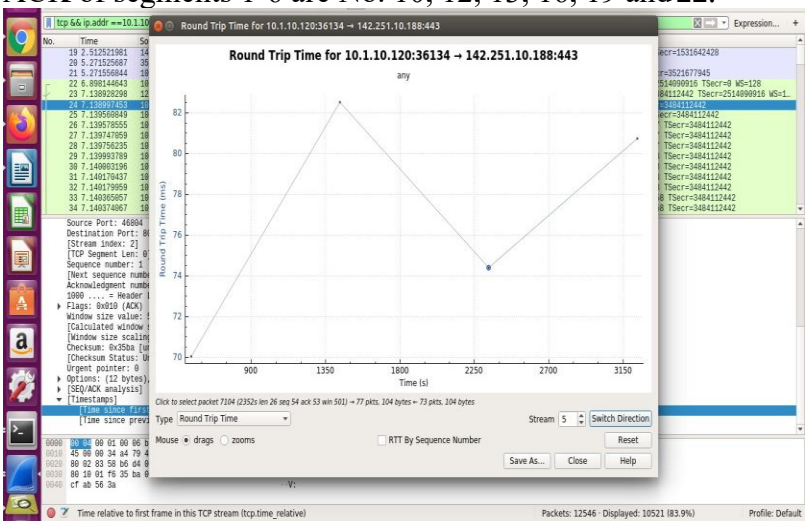


According to above figure, the segment No.6 contains the HTTP POST command, the sequence number of this segment is 1.

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.



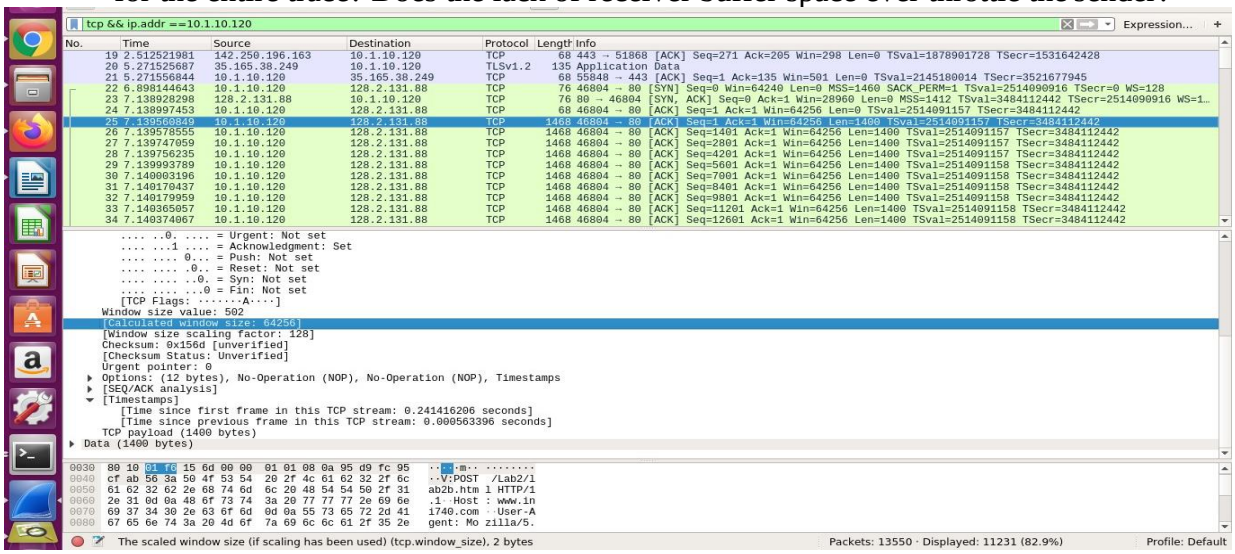
	Sent time
SEGMENT 1	0.226361740
SEGMENT 2	0.227254542
SEGMENT 3	0.227486752
SEGMENT 4	0.227693336
SEGMENT 5	0.227889134
SEGMENT 6	0.228106280

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: 504f5354202f7769726573686172726b2d6c6162732f6c6162...
  [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.

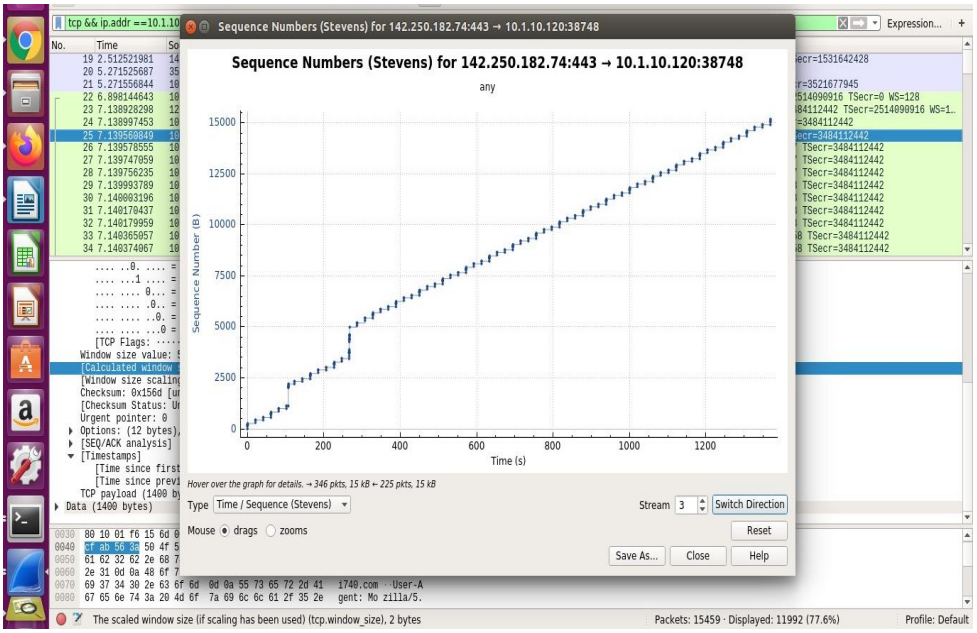
18) What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?



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

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

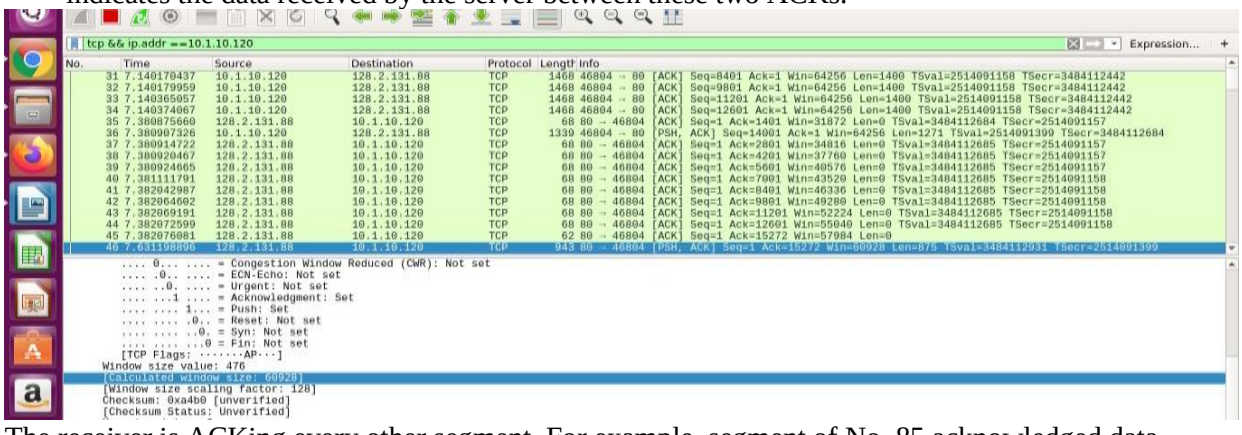
19) Are there any retransmitted segments? What did you check for (in the trace) to answer this question?



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.

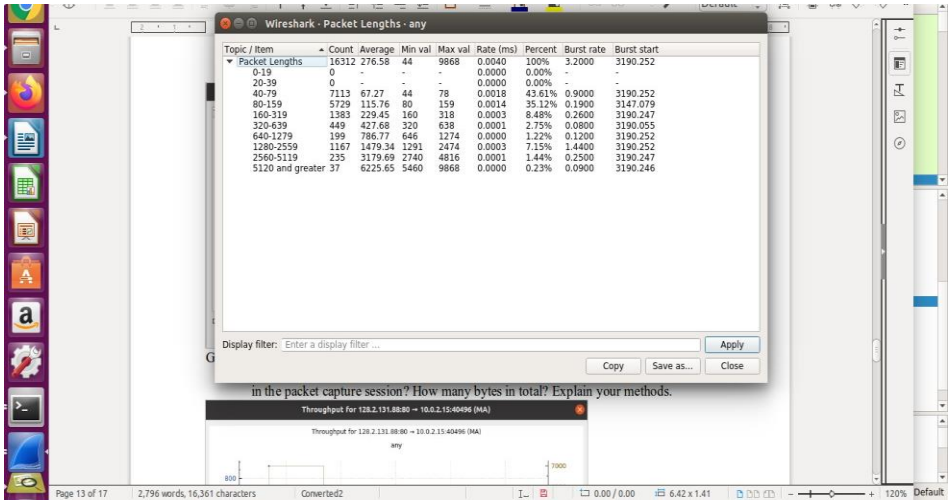
21) What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

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

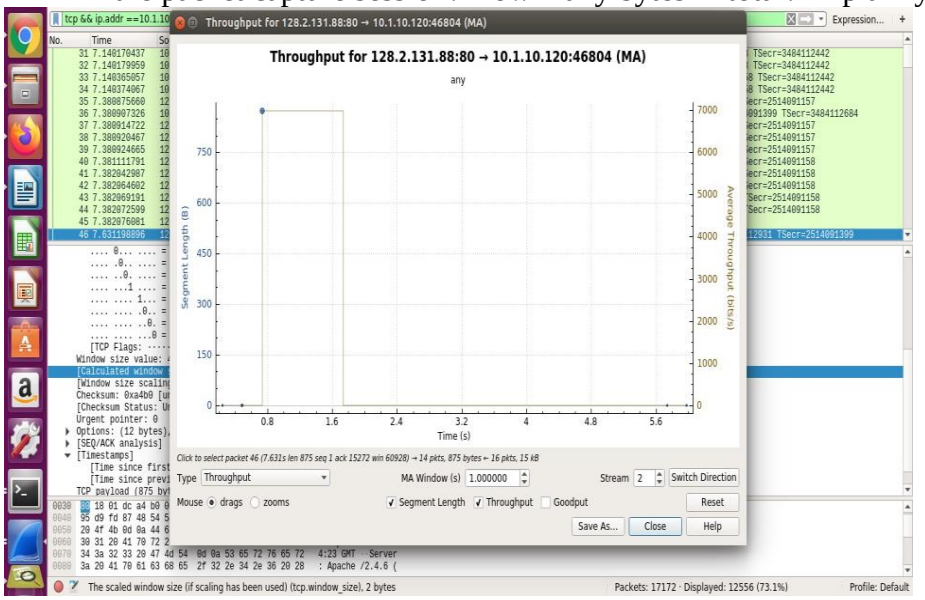
Wireshark has some fairly robust reporting abilities, most of which are accessed via the Statistics menu. Spend a few minutes messing around with the options on that menu, trying to figure out what each report is telling you. Then, answer the following questions about the Canterbury Tales capture:

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.

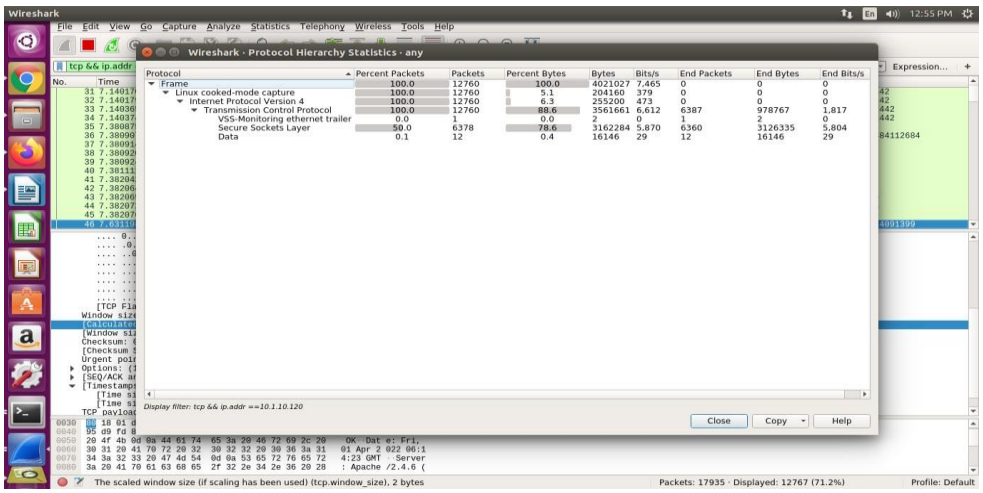


Go to statistics->packet lengths … to get the desired solution

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.

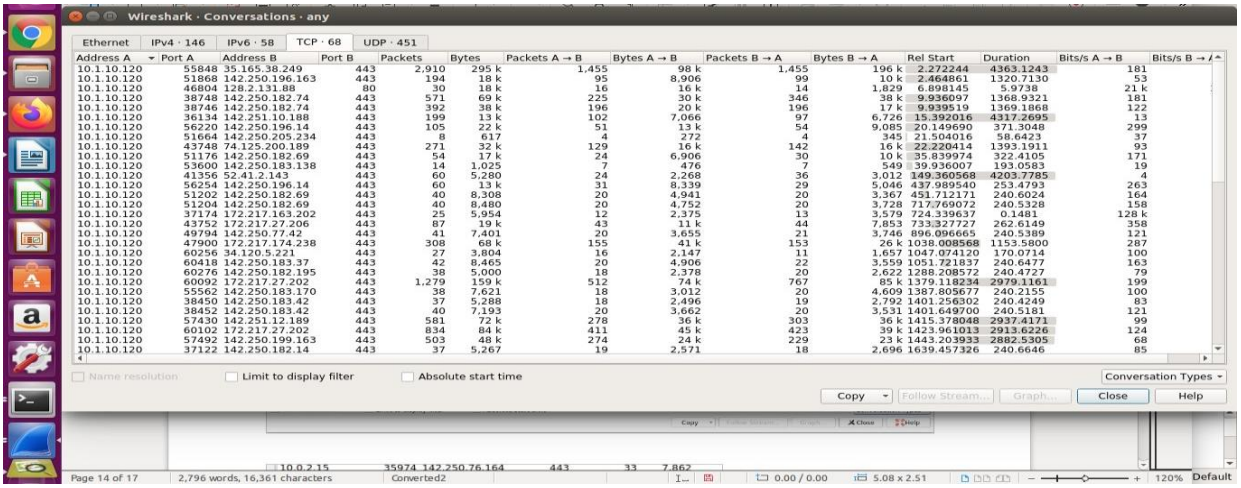


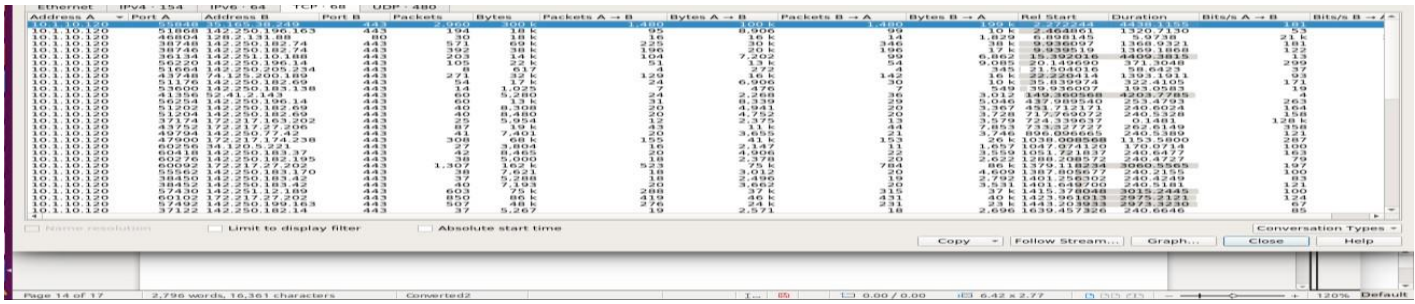
Average throughput: 7000 bits/sec



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?





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.

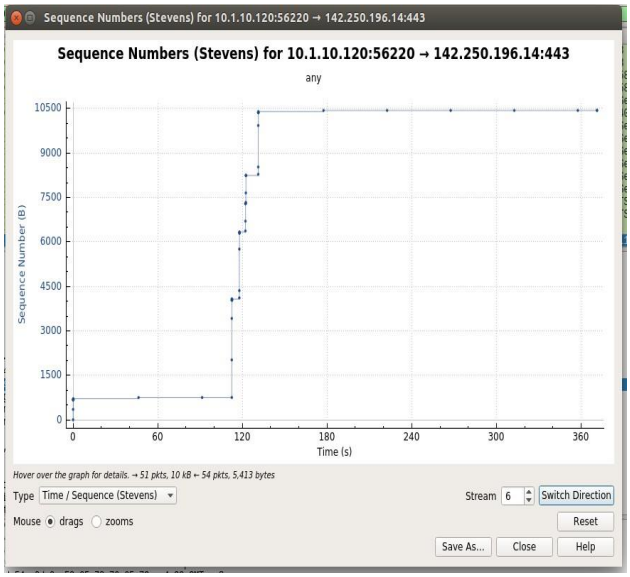
Step 3: Congestion Control

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 0^{th} segment.

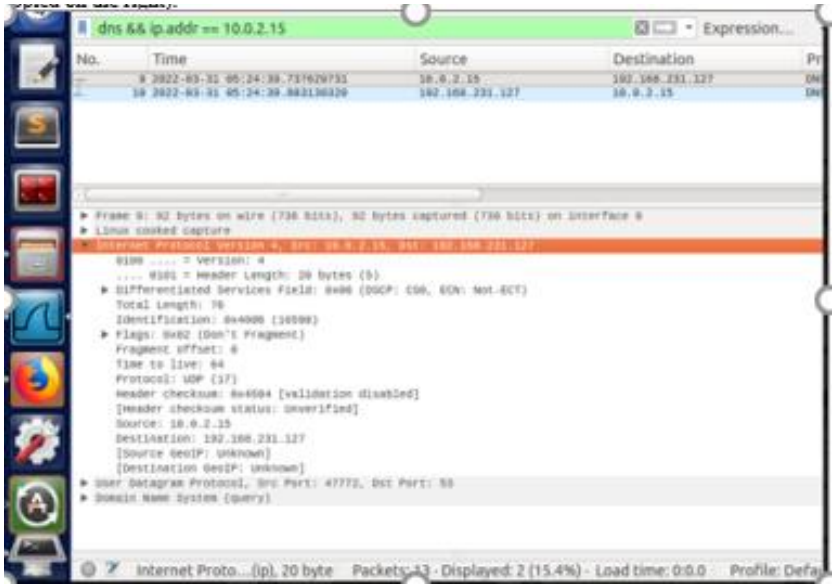
Step 4: The Network Layer

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.

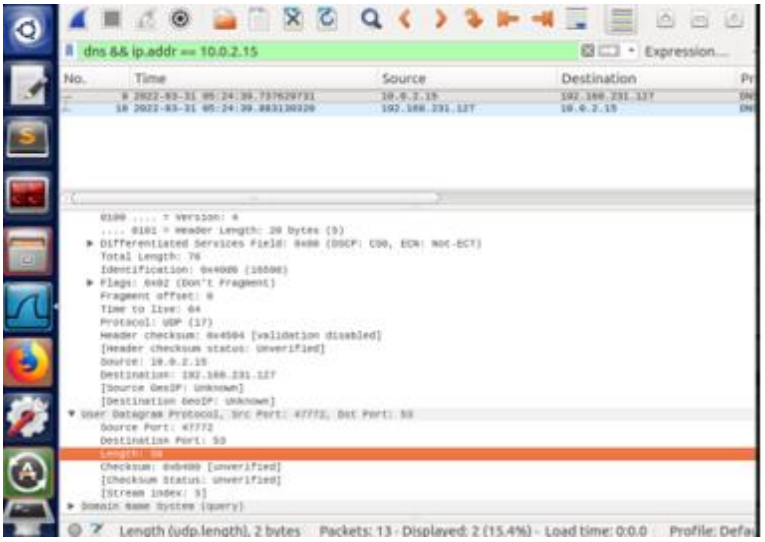
27) Load the capture file that you saved in step 1. Recall that this was a simple DNS query, carried in a UDP packet.

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).

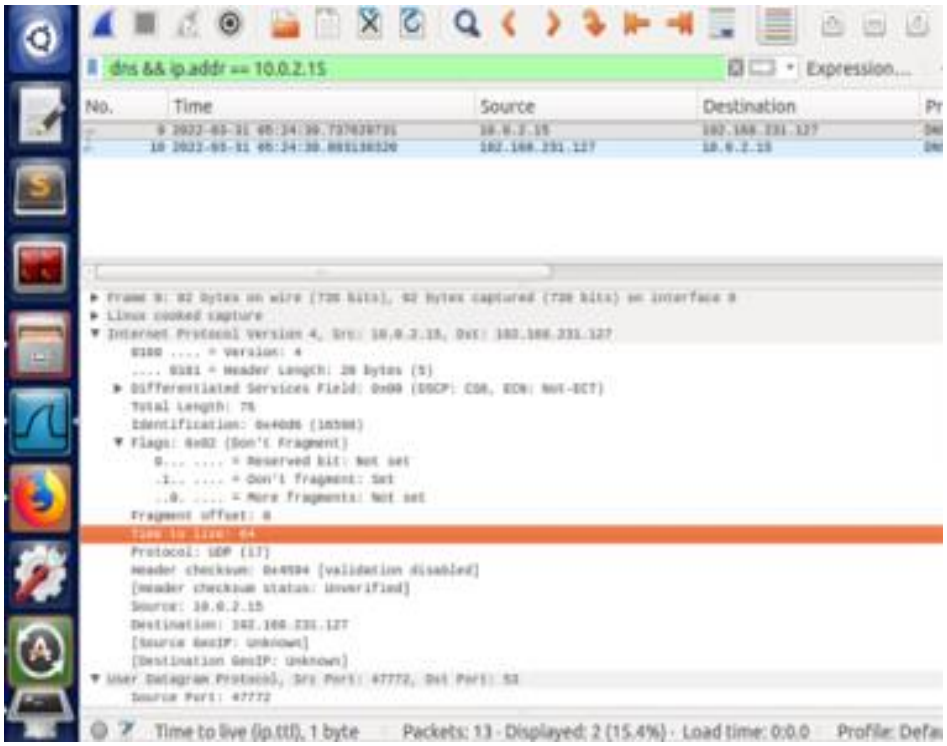


29) Most of the fields should match up and make perfect sense. Verify the Datagram Length, Upper-layer protocol and the IP address fields.



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.

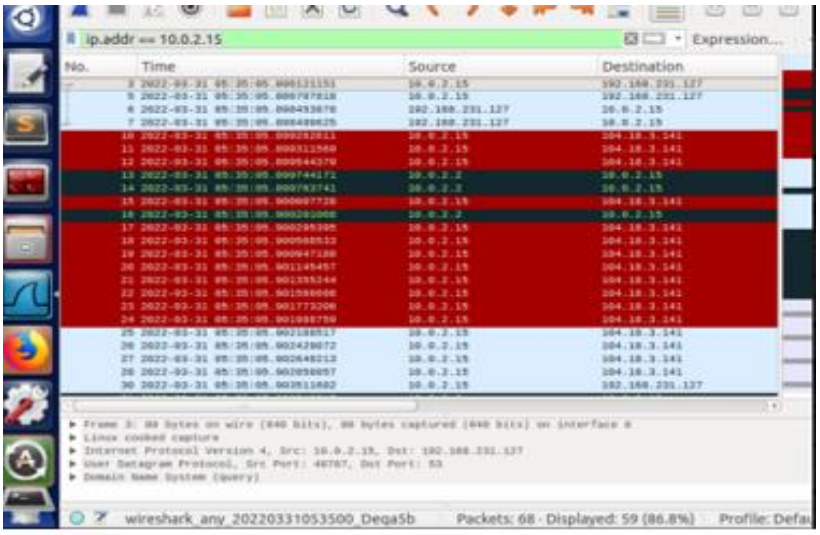


OS: Ubuntu
OS Version:

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>**")



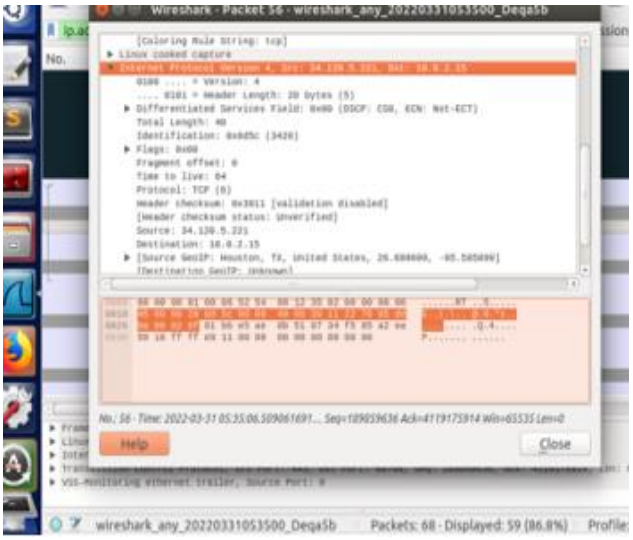
33) In a terminal window, execute the traceroute utility to trace from your computer to **www.cmuj.jp** or **www.regjeringen.no** or some other far-away destination (like we did in our class). If you are having trouble with the weird traceroutes, try this from a non-campus location (your home, a restaurant,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