**Laboratory 2 – TCP**

**Student name:** Phan Phuong Duy. **Student ID**: ITITIU16010

In this lab, we’ll investigate the behavior of the celebrated TCP protocol in detail. We’ll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carrol’s *Alice’s Adventures in Wonderland*) from your computer to a remote server. We’ll study TCP’s use of sequence and acknowledgement numbers for providing reliable data transfer; we’ll see TCP’s congestion control algorithm – slow start and congestion avoidance – in action; and we’ll look at TCP’s receiver-advertised flow control mechanism. We’ll also briefly consider TCP connection setup and we’ll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

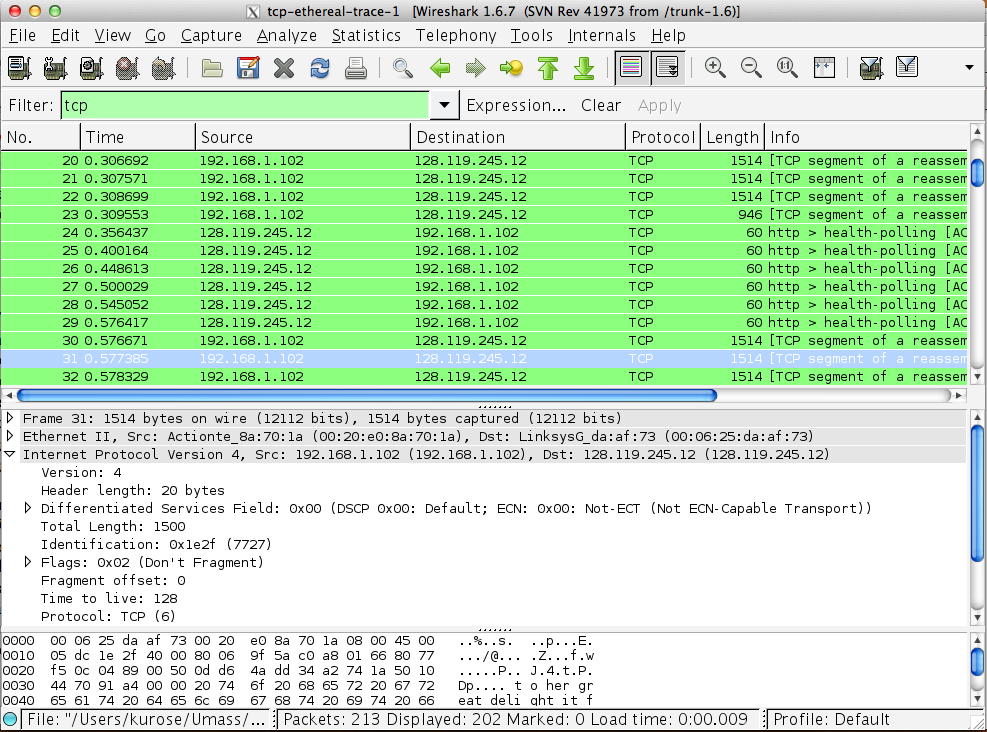
Before beginning this lab, you’ll probably want to review sections 3.5 and 3.7 in the text.

1. Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). We’re using the POST method rather than the GET method as we’d like to transfer a large amount of data *from* your computer to another computer. Of course, we’ll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do the following:

* Start up your web browser. Go the <http://gaia.cs.umass.edu/wireshark-labs/alice.txt> and retrieve an ASCII copy of *Alice in Wonderland.* Store this file somewhere on your computer.
* Next go to <http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html>.
* Use the *Browse* button in the form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually).Don’t yet press the “*Upload alice.txt file*” button.
* Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we’ll not need to select any options here).
* Returning to your browser, press the “*Upload alice.txt file*” button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
* Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.



2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let’s take a high level view of the trace.

* First, filter the packets displayed in the Wireshark window by entering “tcp” (lowercase, no quotes, and don’t forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

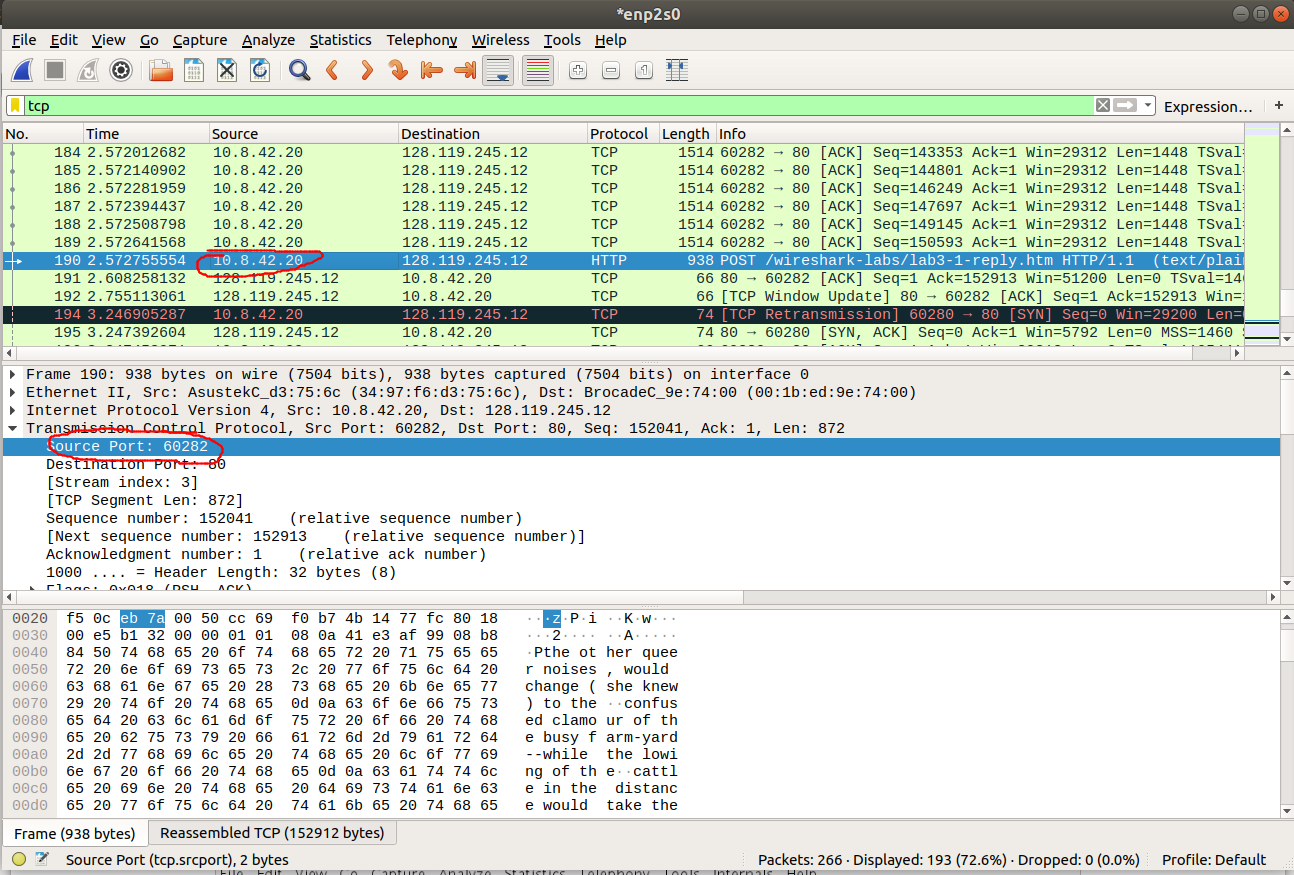
What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. Depending on the version of Wireshark you are using, you might see a series of “HTTP Continuation” messages being sent from your computer to gaia.cs.umass.edu. Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message – this is Wireshark’s way of indicating that there are multiple TCP segments being used to carry a single HTTP message. In more recent versions of Wireshark, you’ll see “[TCP segment of a reassembled PDU]” in the Info column of the Wireshark display to indicate that this TCP segment contained data that belonged to an upper layer protocol message (in our case here, HTTP). You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions. Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Highlight and annotate the printout to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question.

1. **What is the IP address and TCP port number used by your client computer (source) that is transferring the file to gaia.cs.umass.edu?**

My computer IP address: 10.8.42.20

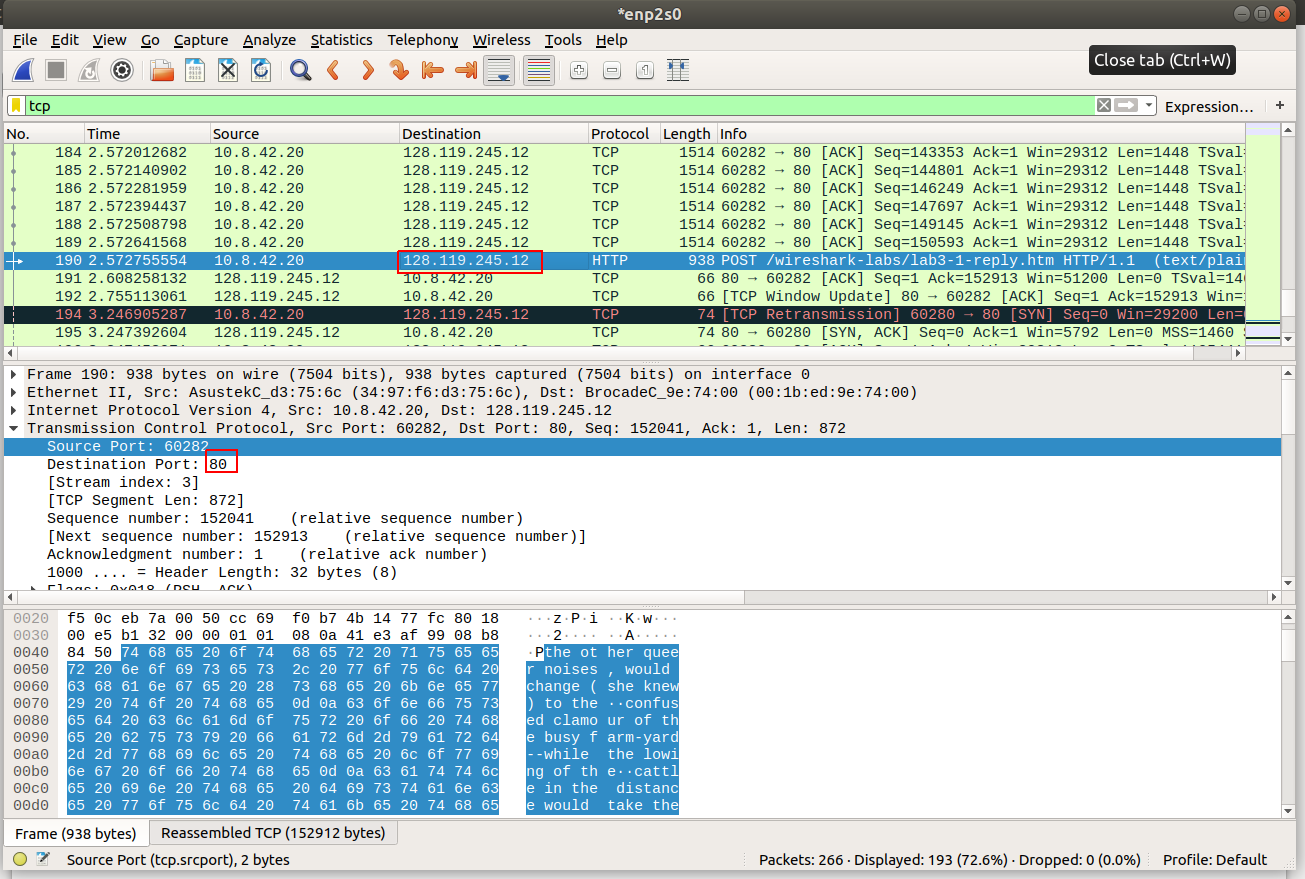
TCP port number: 60282



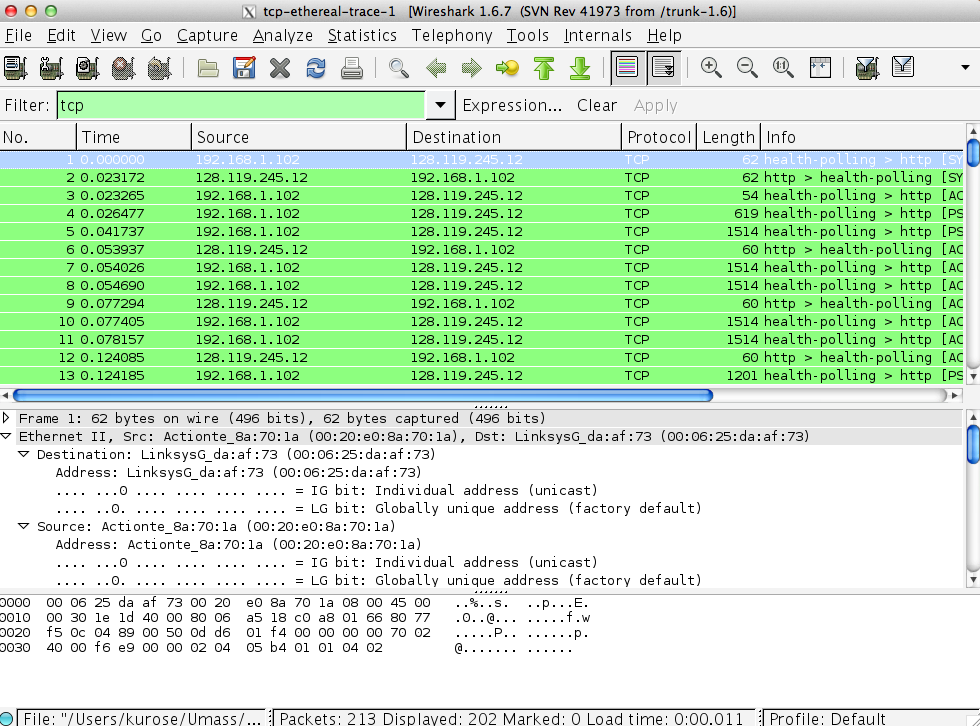
1. **What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?**

IP address of gaia.cs.umass.edu: 128.119.245.12

TCP seding and receving port number: 80



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:



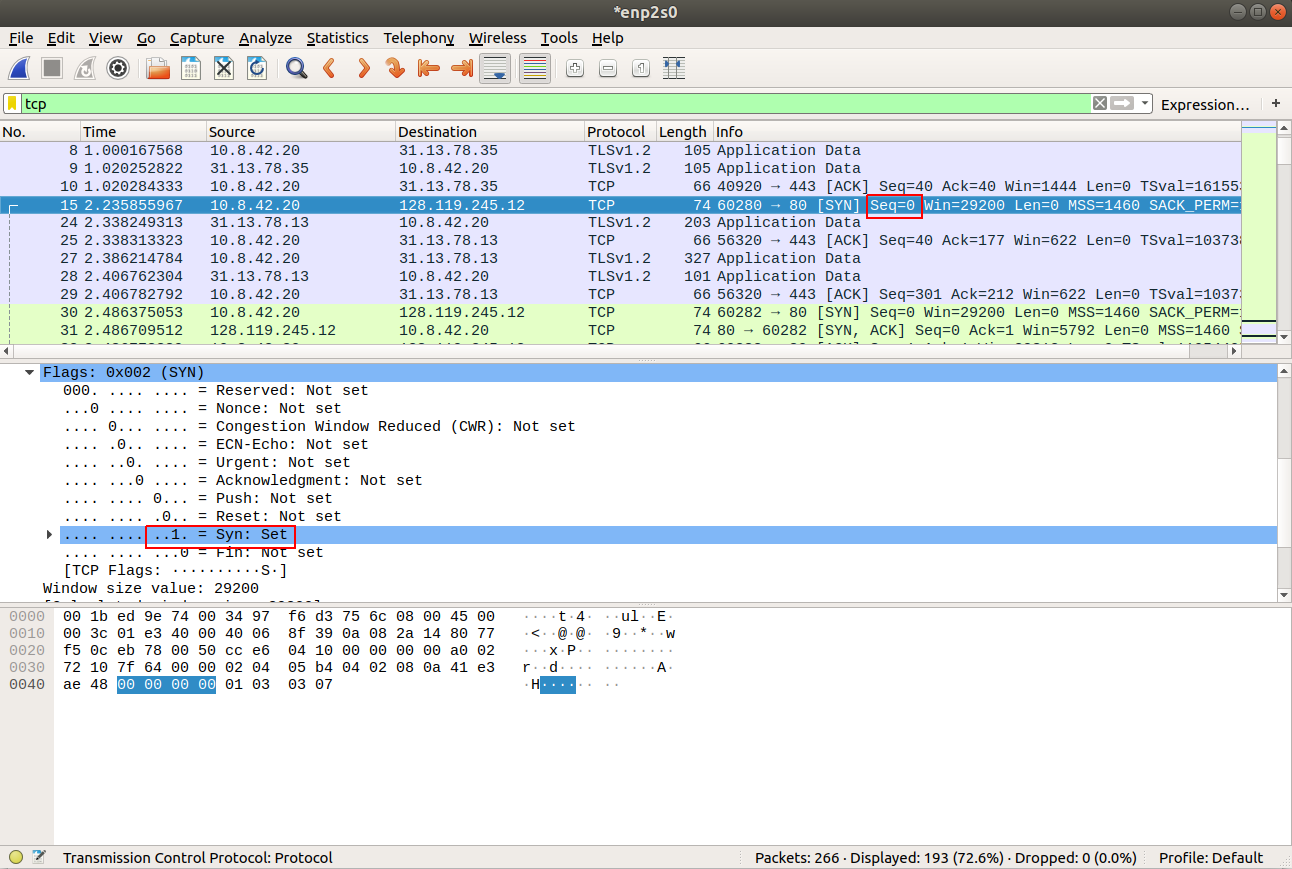
This is what we’re looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured to study TCP behavior in the rest of this lab.

3. TCP Basics

Answer the following questions for the TCP segments:

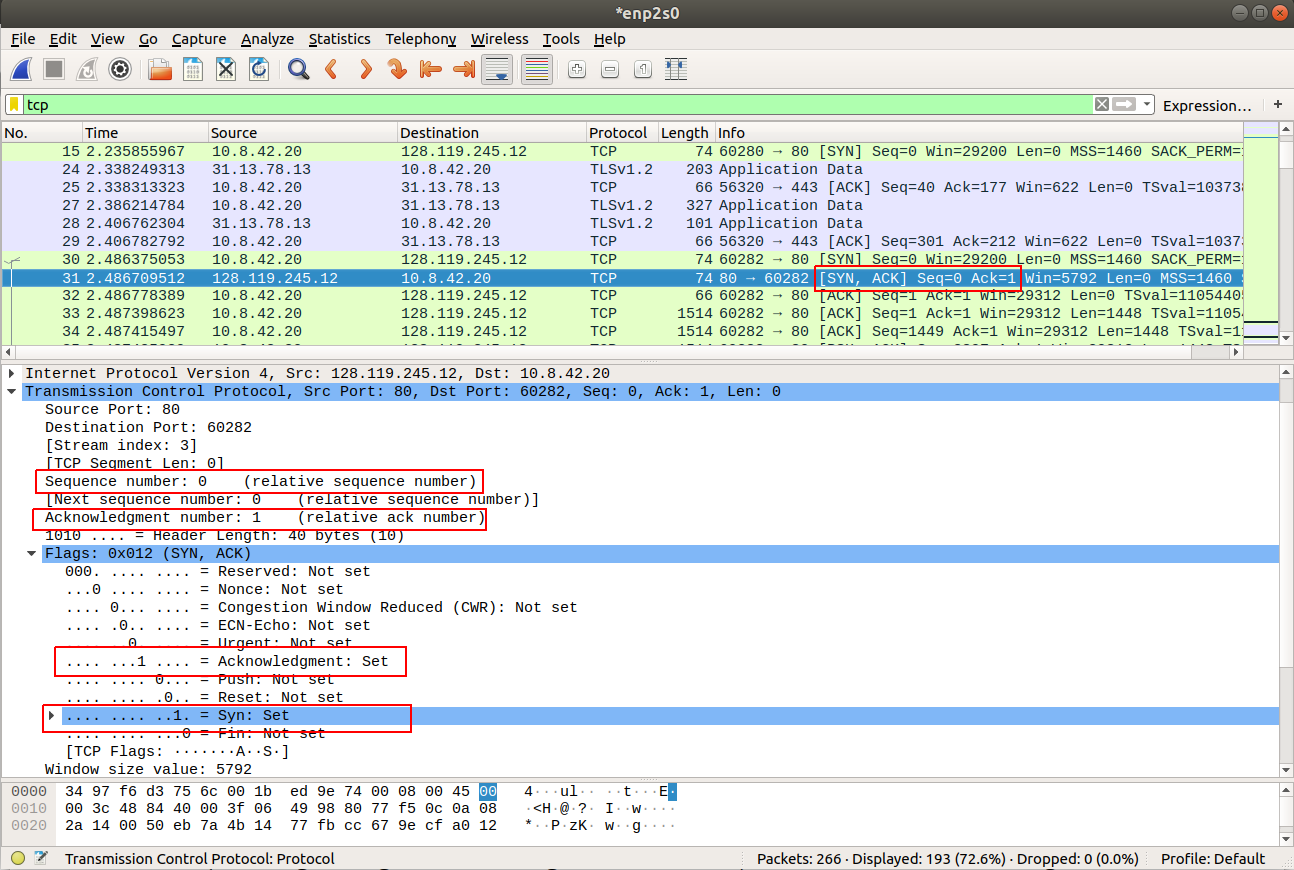
1. **What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?**

Sequence number of TCP SYN segment is 0. According to the image below, in the Flags section, the SYN flag is set to 1 to indicates the segment as a SYN segment.



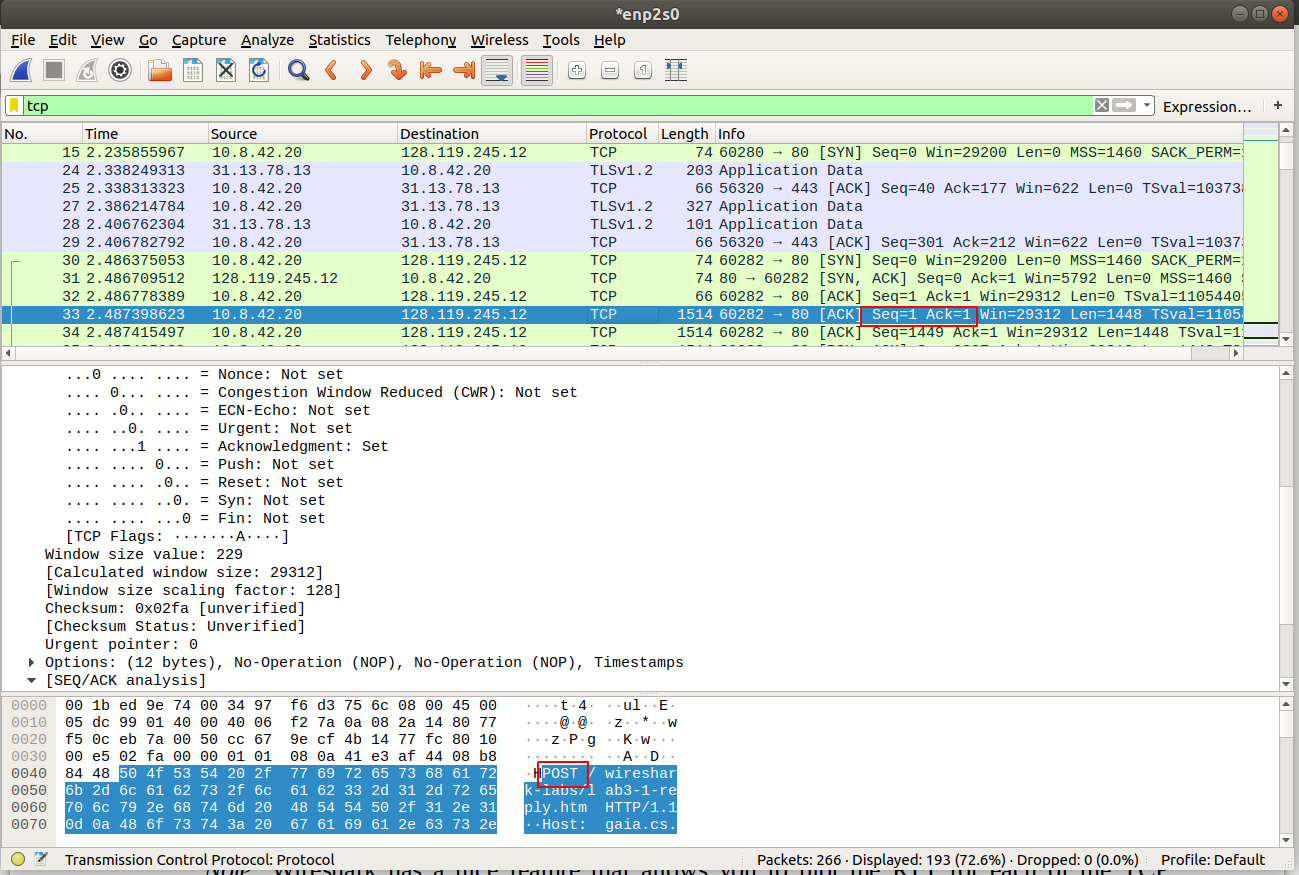
1. **What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?**

The sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN is 0. The value of the Acknowledgement field is 1. The server determines the value of Acknowledgement field by adds 1 to the initial sequence number of SYN segment. In this case, because the sequence number of SYN segment from client computer is 0, so the Acknowledgement number in SYN\_ACK segment is 1. Therefore, the segment is SYN\_ACK if both Acknowledgement and Syn flags are 1.



1. **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.**

The sequence number of the TCP segment containing the HTTP POST command is 1.



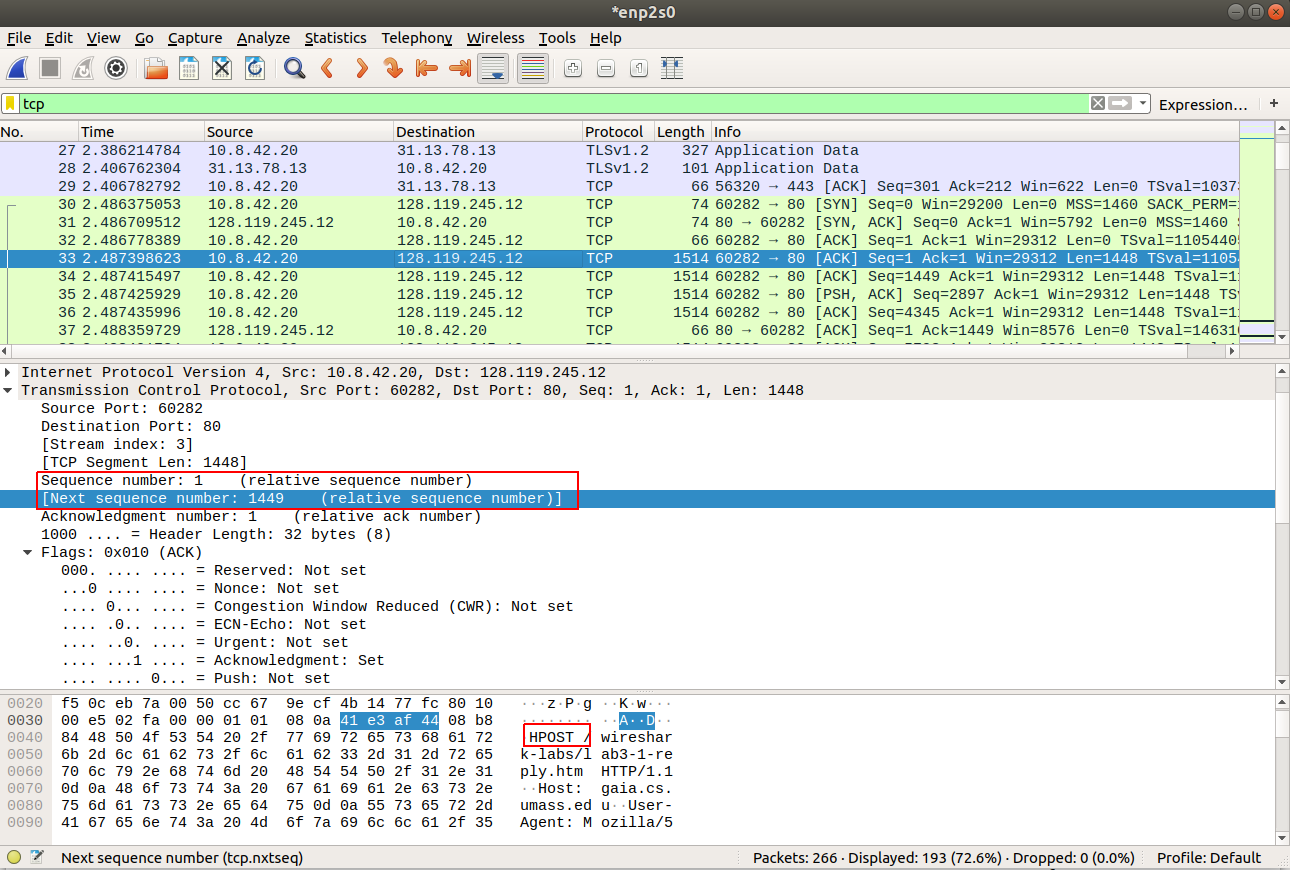
1. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3, page 239 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 239 for all subsequent segments.

First six segments in the TCP connection:

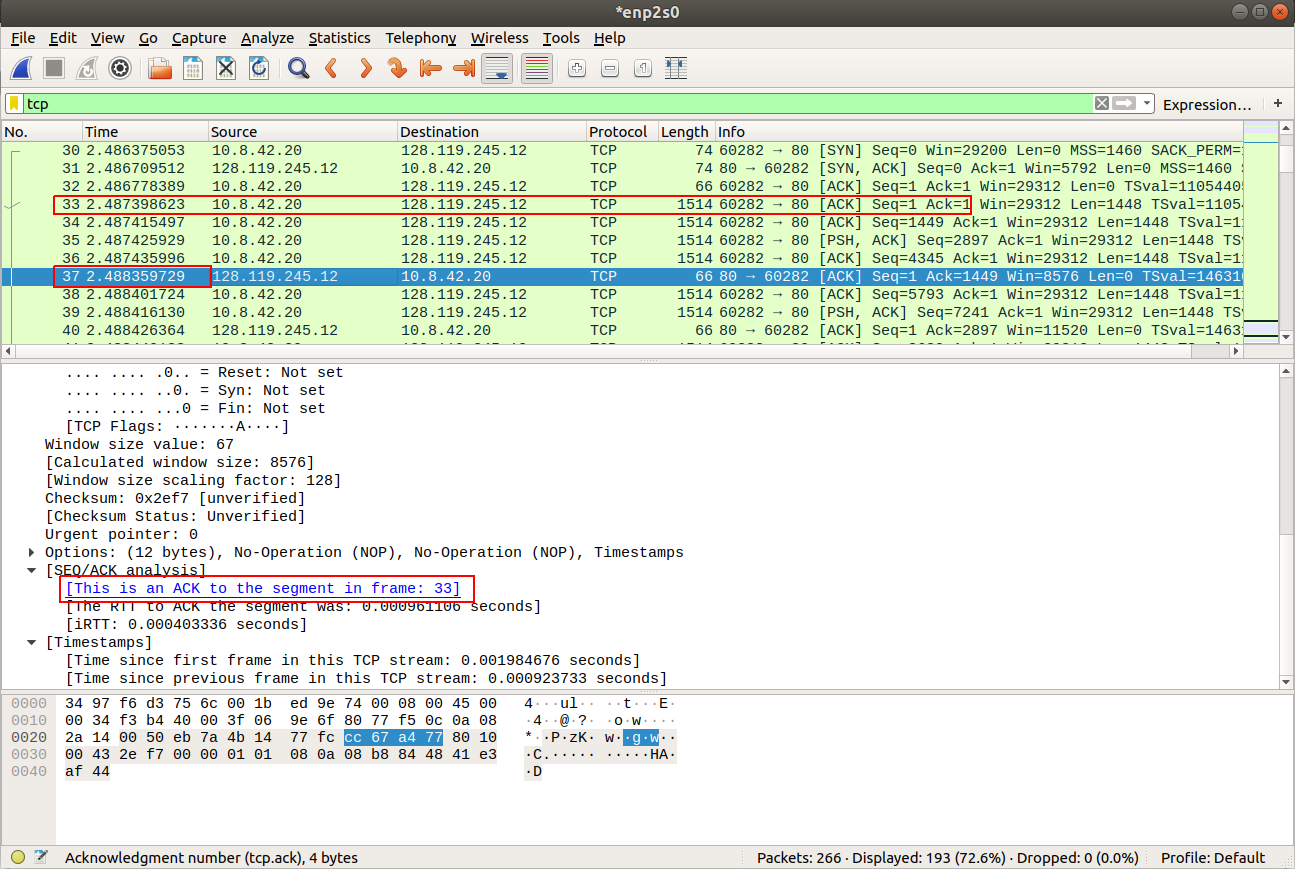
1st segment:

Sequence number: 1

Time : 2.487398623



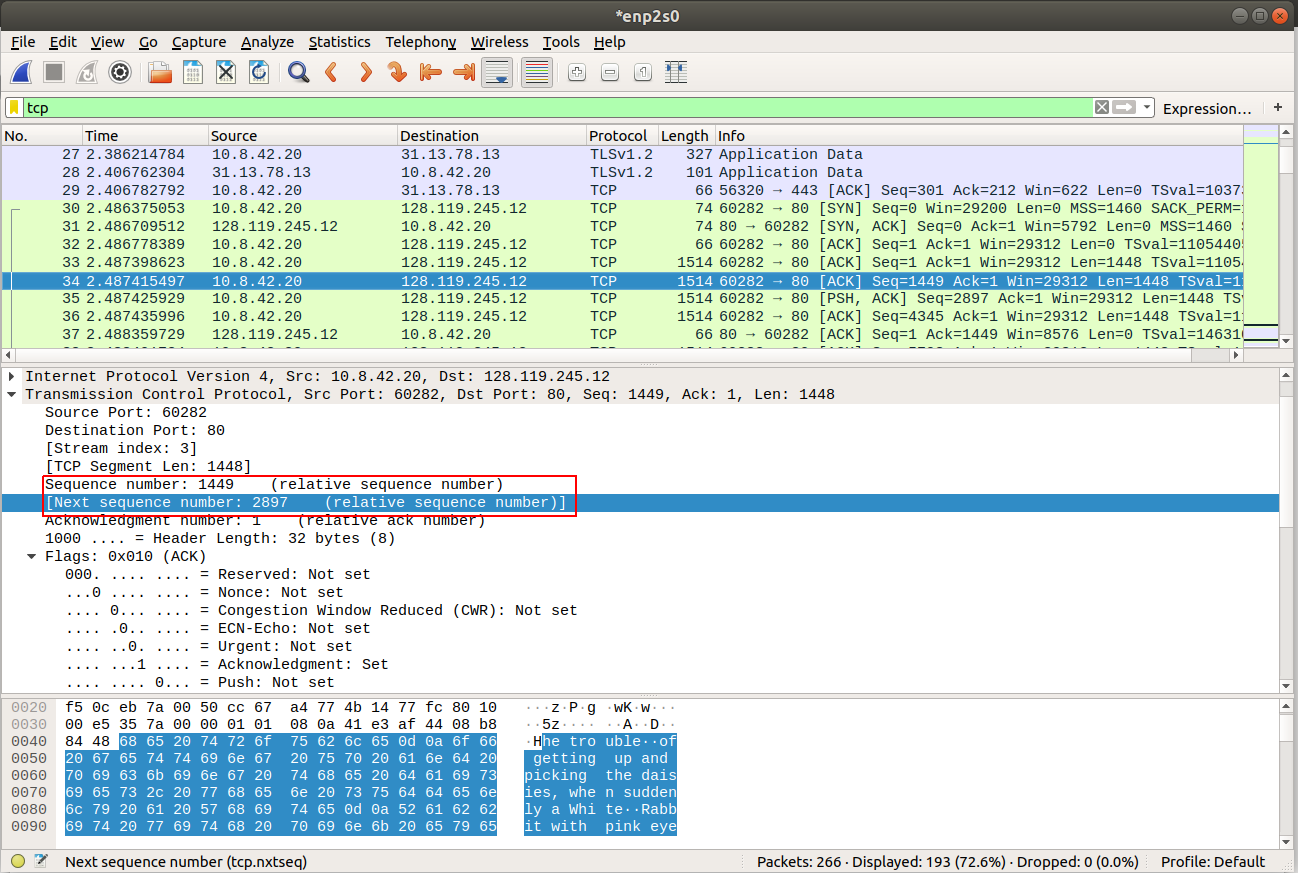
Time the ACK received: 2.488359729



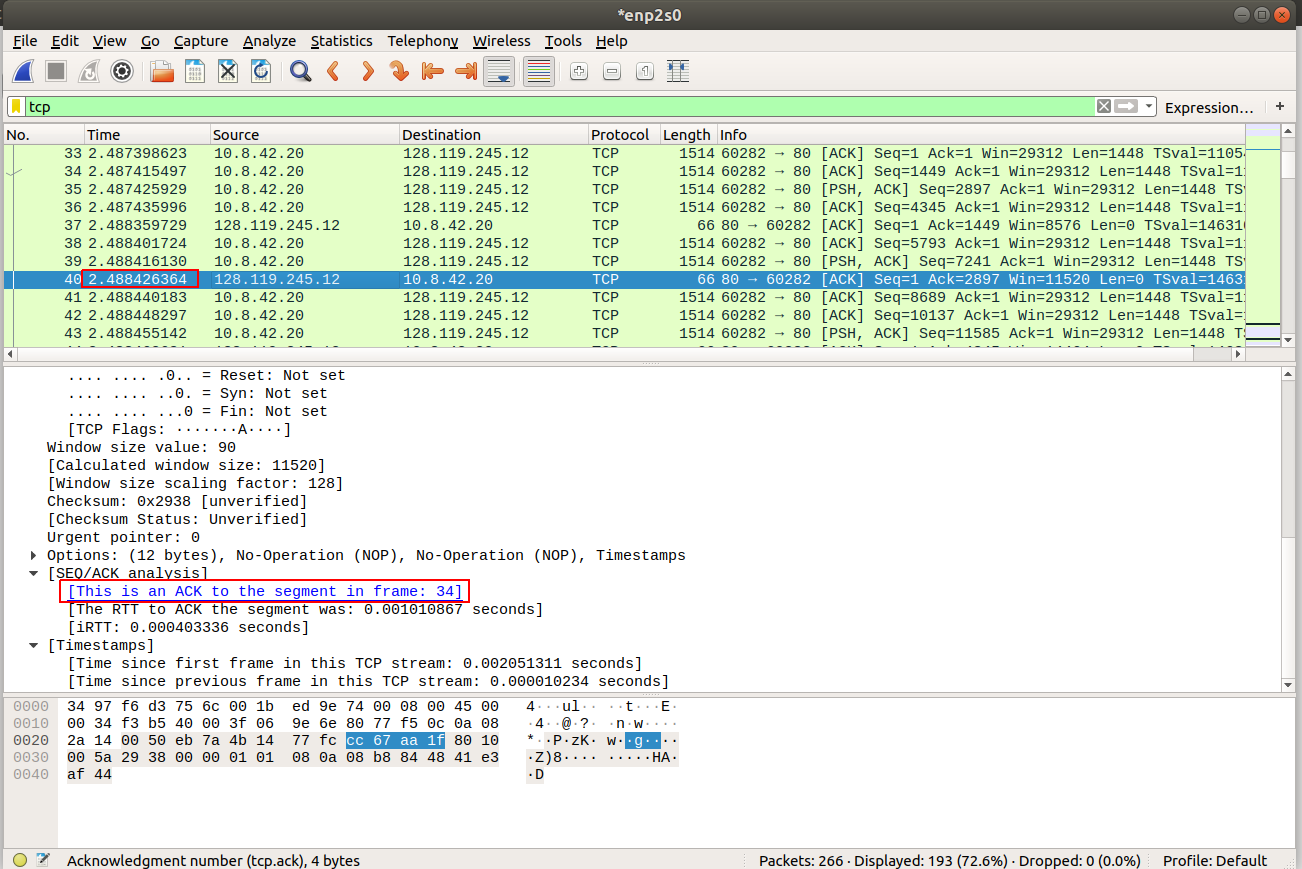
2nd segment:

Sequence number: 1449

Time : 2.487415497



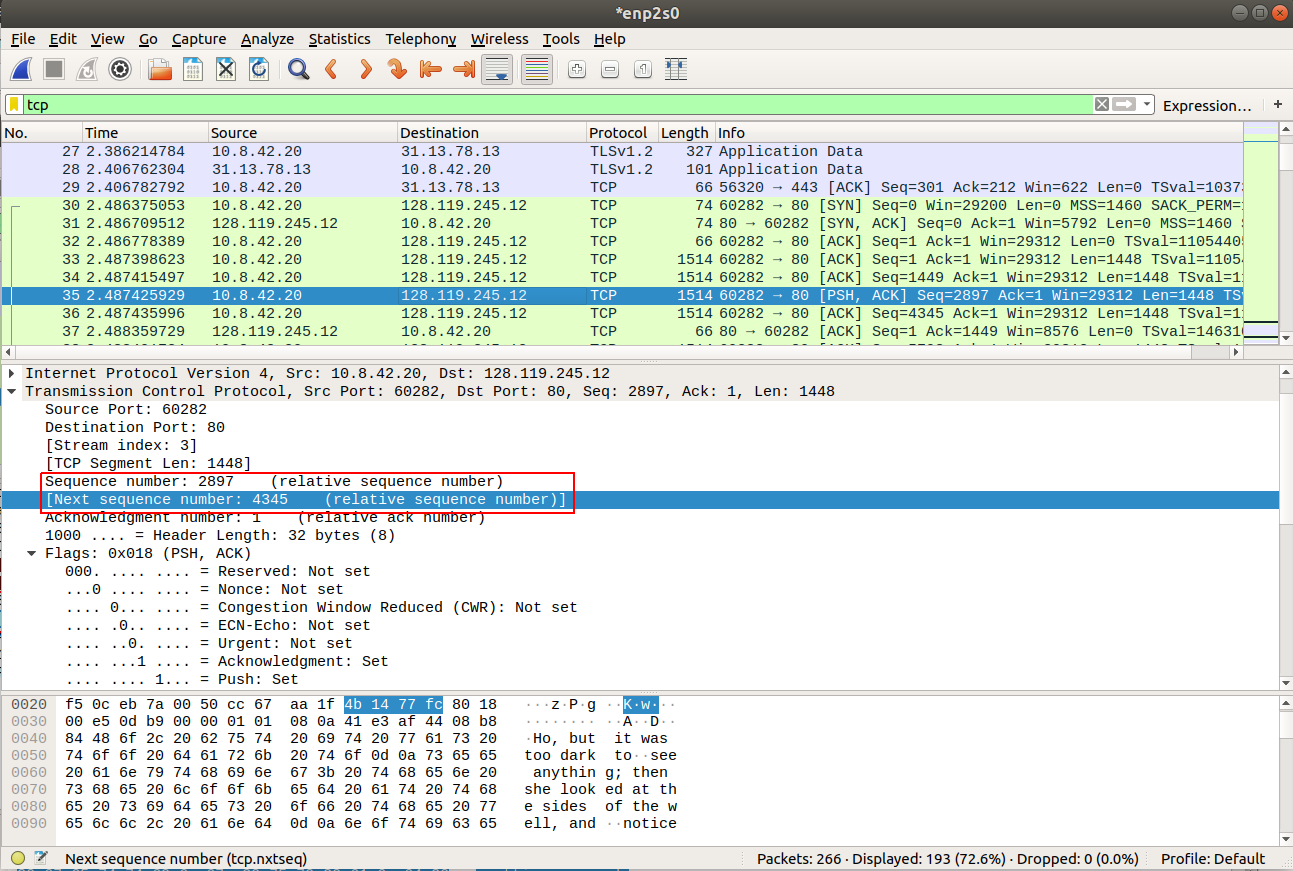
Time the ACK received: 2.488426364



3rd segment:

Sequence number: 2897

Time: 2.487425929



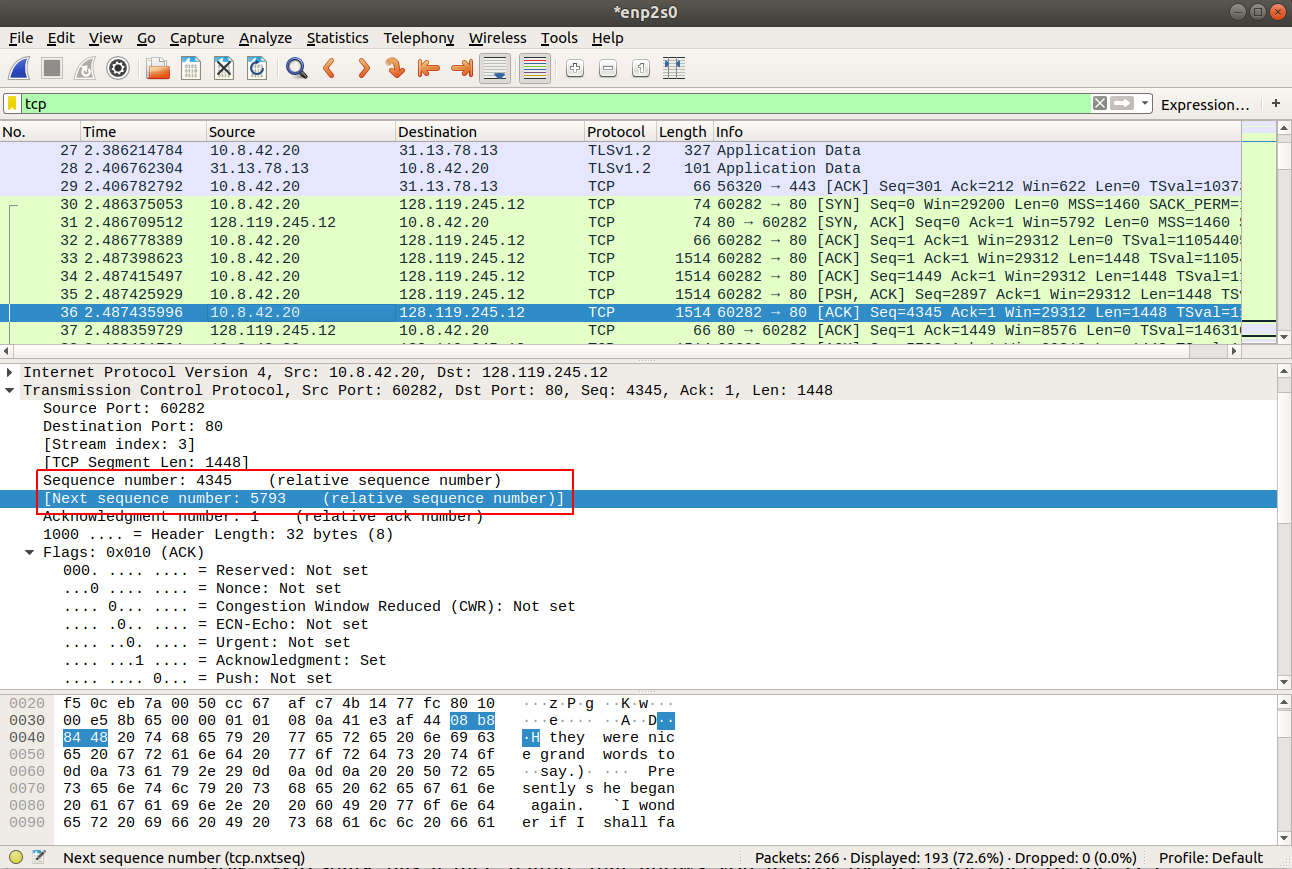
Time the ACK received: 2.488462631



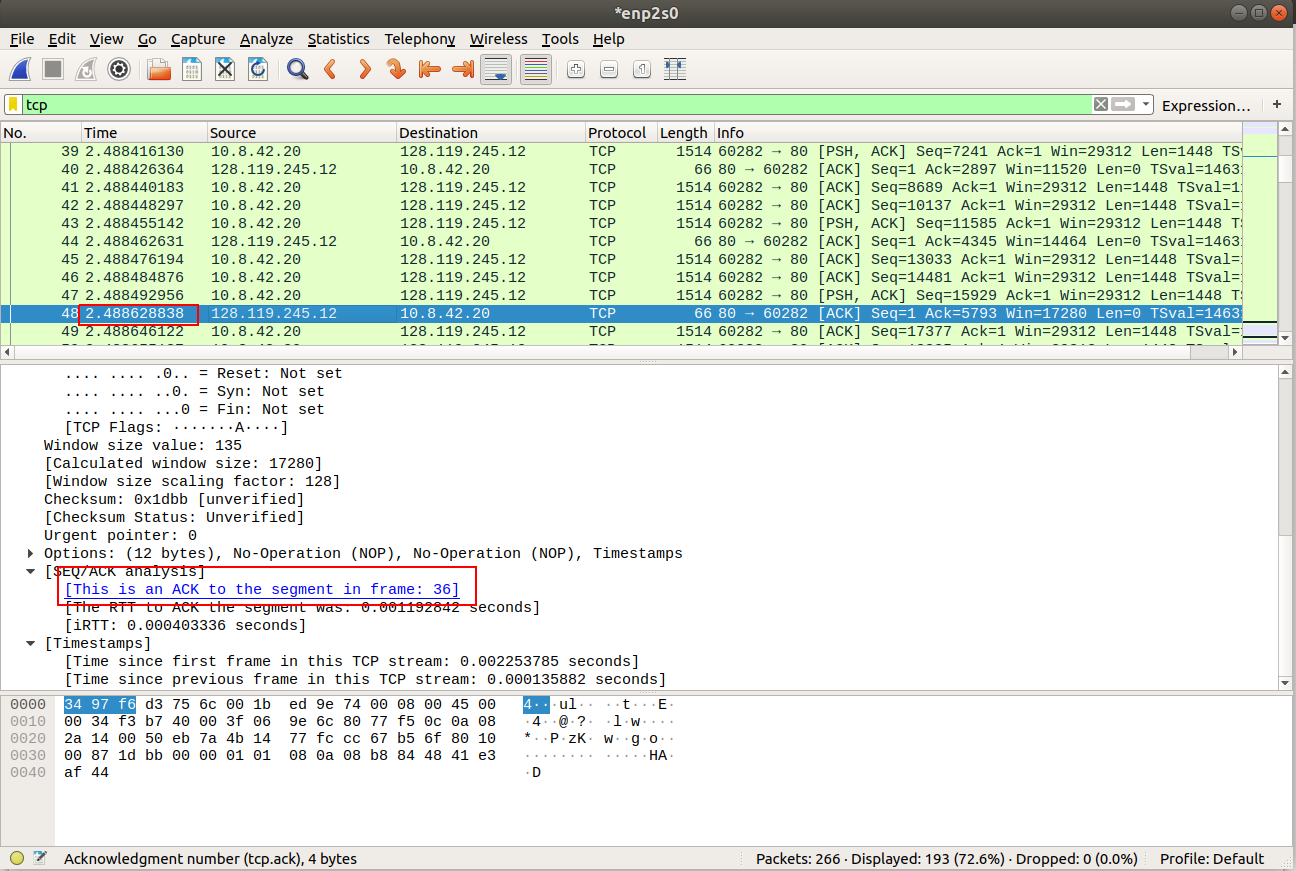
4th segment:

Sequence number: 4345

Time: 2.487435996



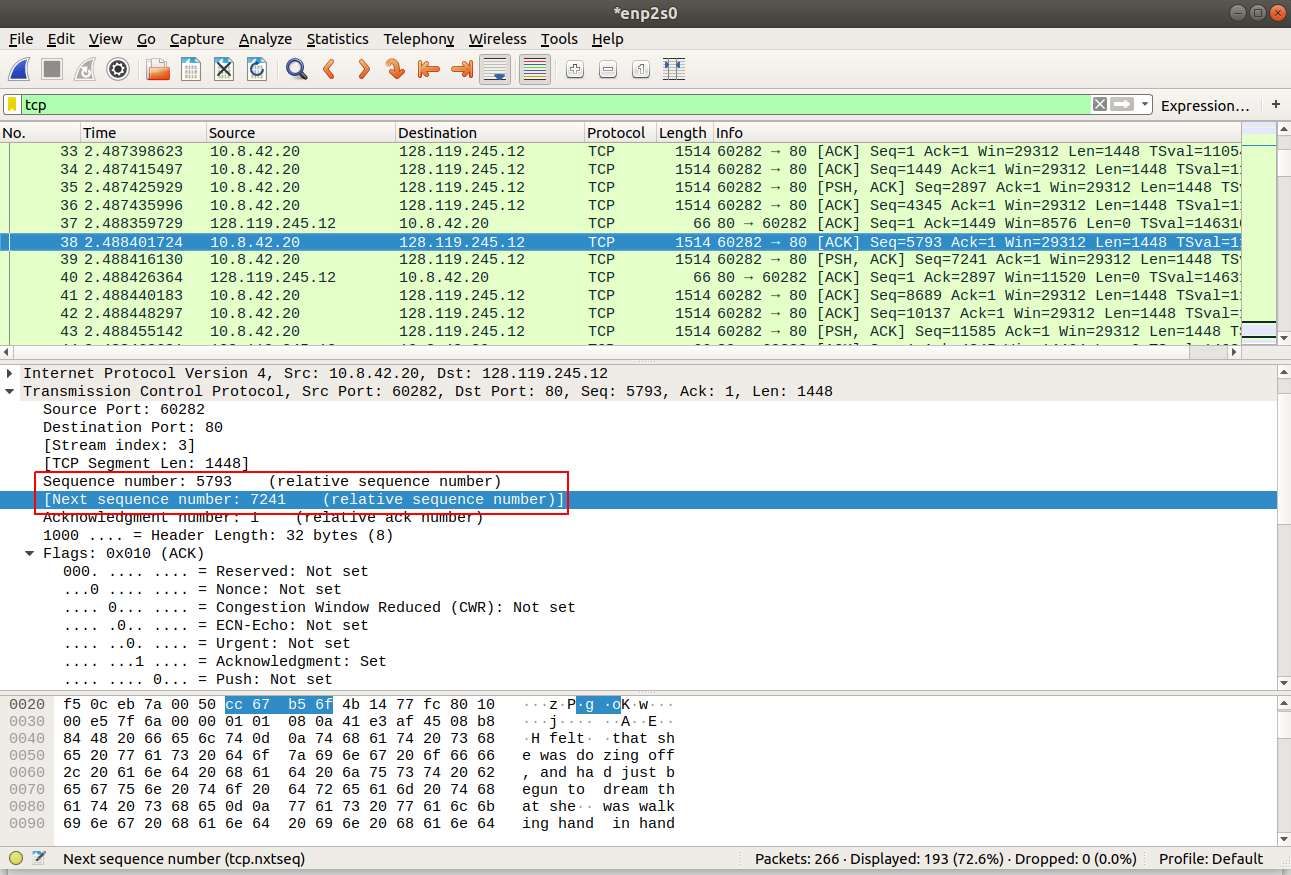
Time the ACK received: 2.488628838



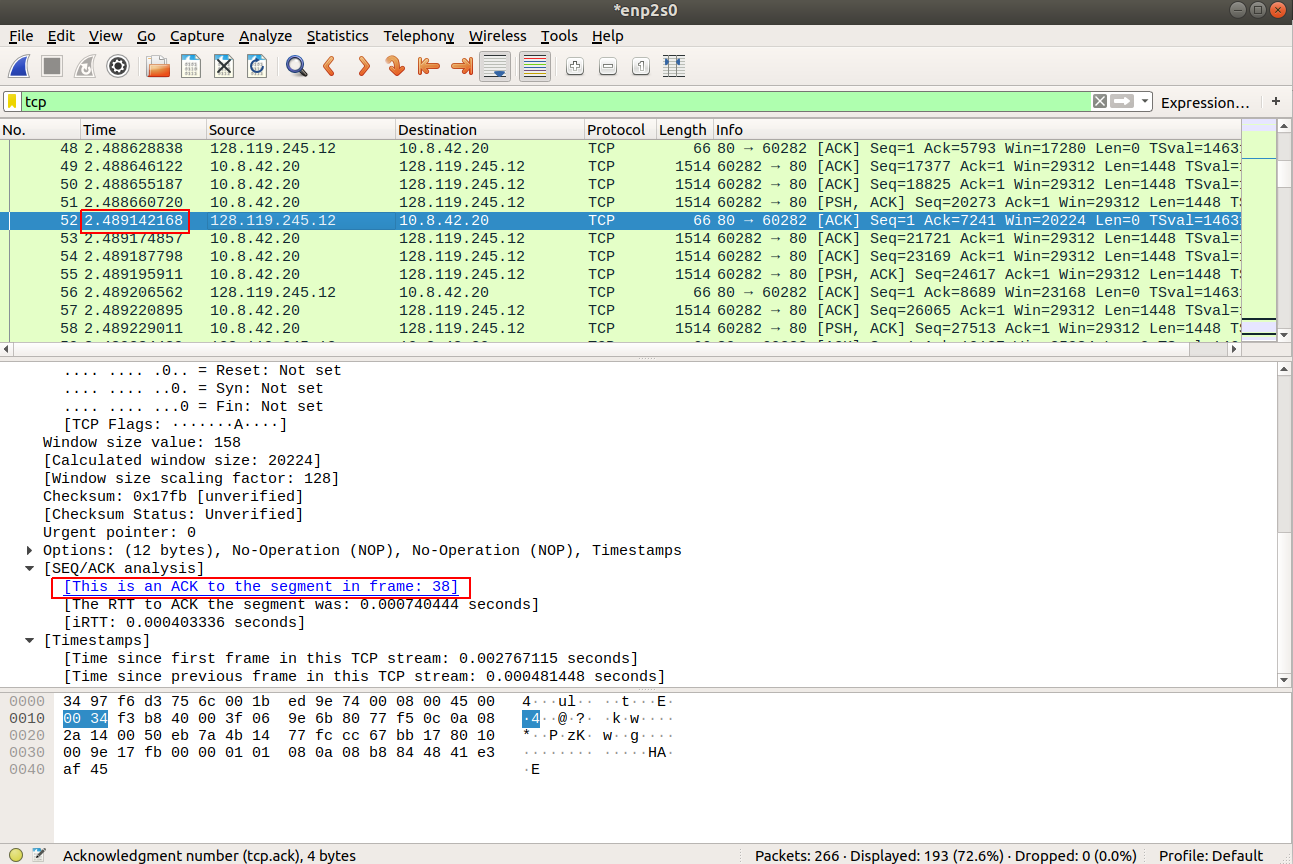
5th segment:

Sequence number: 5793

Time: 2.488401724



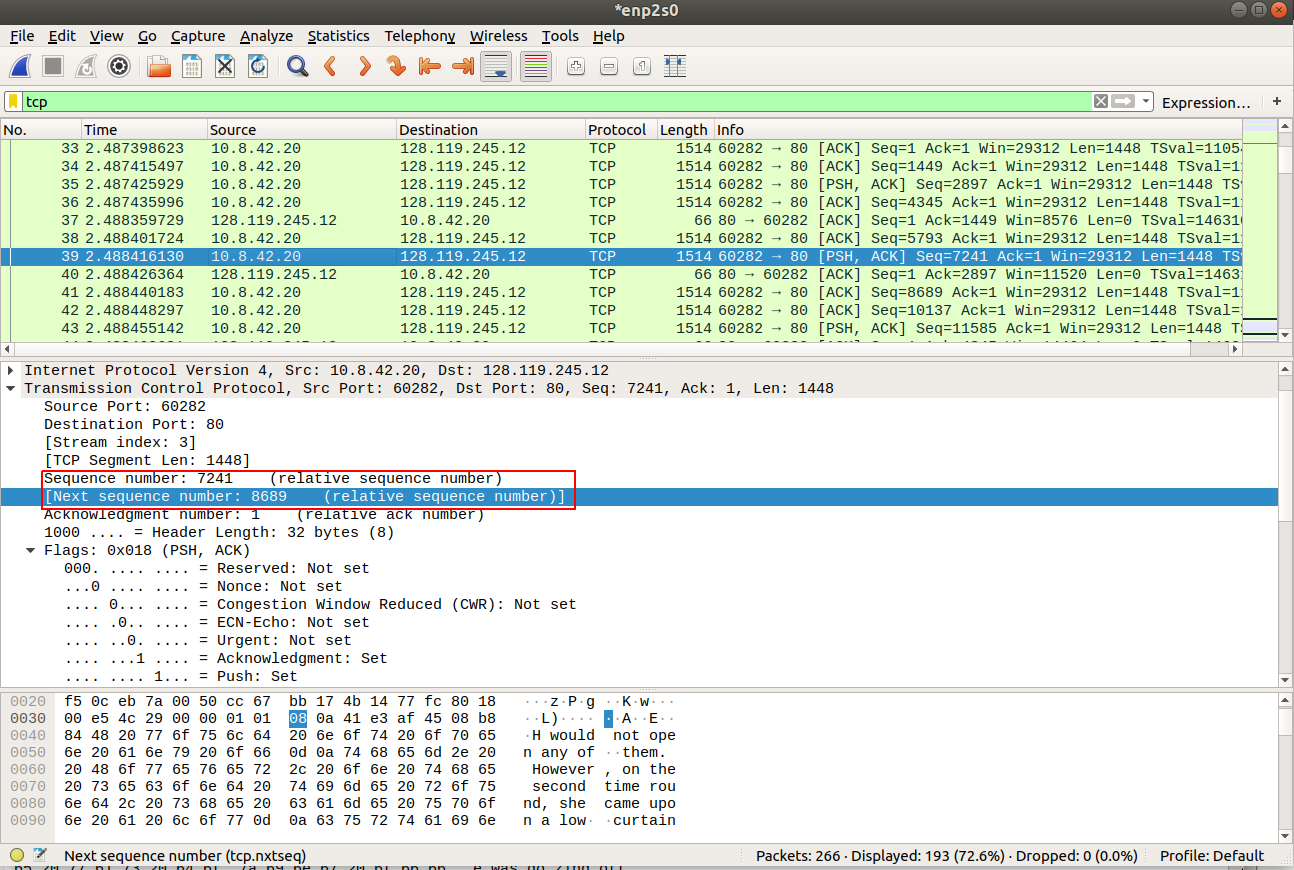
Time ACK received: 2.489142168



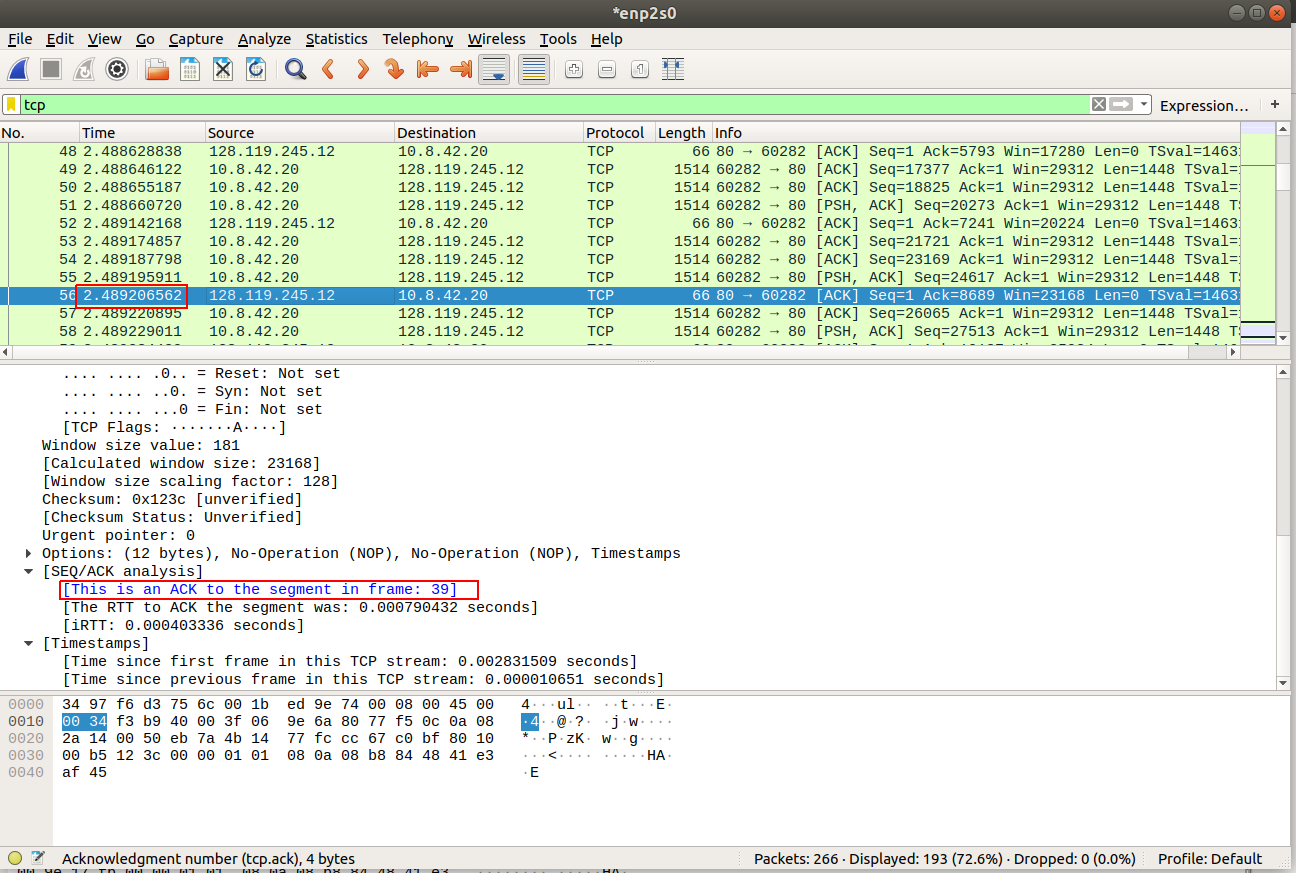
6th segment:

Sequence number: 7241

Time: 2.488416139



Time ACK received*:*2.489206562

**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Segment | Frame (packet number) | Sequence number | Sent time | ACK received time | RTT (seconds) |
| 1 | 33 | 1 | 2.487398623 | 2.488359729 | 0.000961106 |
| 2 | 34 | 1449 | 2.487415497 | 2.488426364 | 0.001010867 |
| 3 | 35 | 2897 | 2.487425929 | 2.488462631 | 0.001036702 |
| 4 | 36 | 4345 | 2.487435996 | 2.488628838 | 0.001192842 |
| 5 | 38 | 5793 | 2.488401724 | 2.489142168 | 0.000740444 |
| 6 | 39 | 7241 | 2.488416139 | 2.489206562 | 0.000790432 |

*Formula: EstimatedRTT = 0.875 • EstimatedRTT + 0.125 • SampleRTT*

EstimatedRTT after the receipt of the ACK of segment 1:

EstimateRTT = RTT for segment 1 = 0.000961106

EstimatedRTT after the receipt of the ACK of segment 2:

EstimateRTT = 0.875 \* 0.000961106 + 0.125 \* 0.001010867 = 0.000967326125

EstimatedRTT after the receipt of the ACK of segment 3:

EstimateRTT = 0.875 \* 0.000967326125 + 0.125 \* 0.001036702 = 0.000975998

EstimatedRTT after the receipt of the ACK of segment 4:

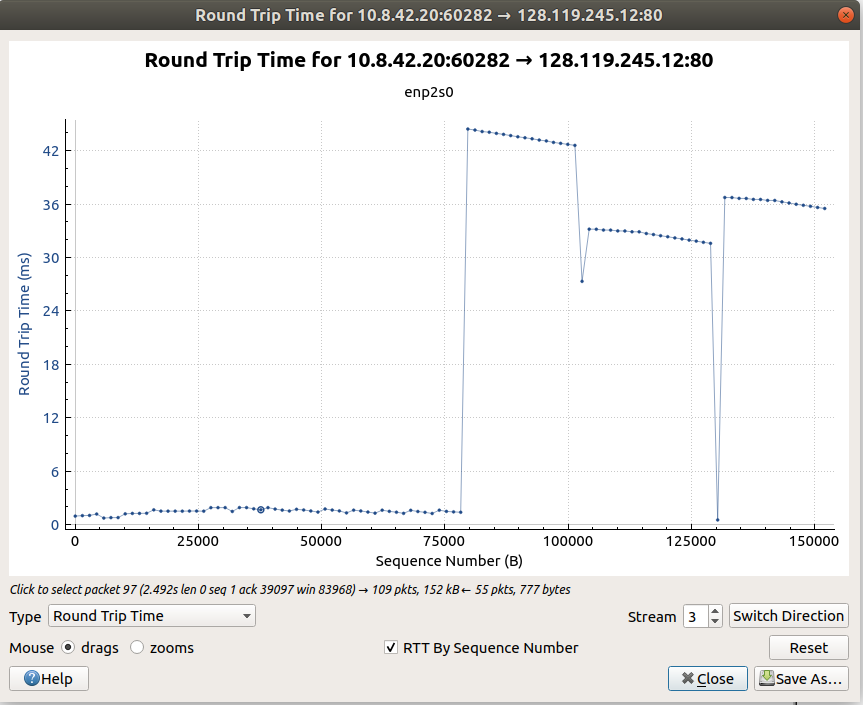
EstimateRTT = 0.875\* 0.000975998109375 + 0.125 \* 0.001192842 = 0.00100310

EstimatedRTT after the receipt of the ACK of segment 5:

EstimateRTT = 0.875 \* 0.00100310 + 0.125 \* 0.000740444 = 0.000970271

EstimatedRTT after the receipt of the ACK of segment 6:

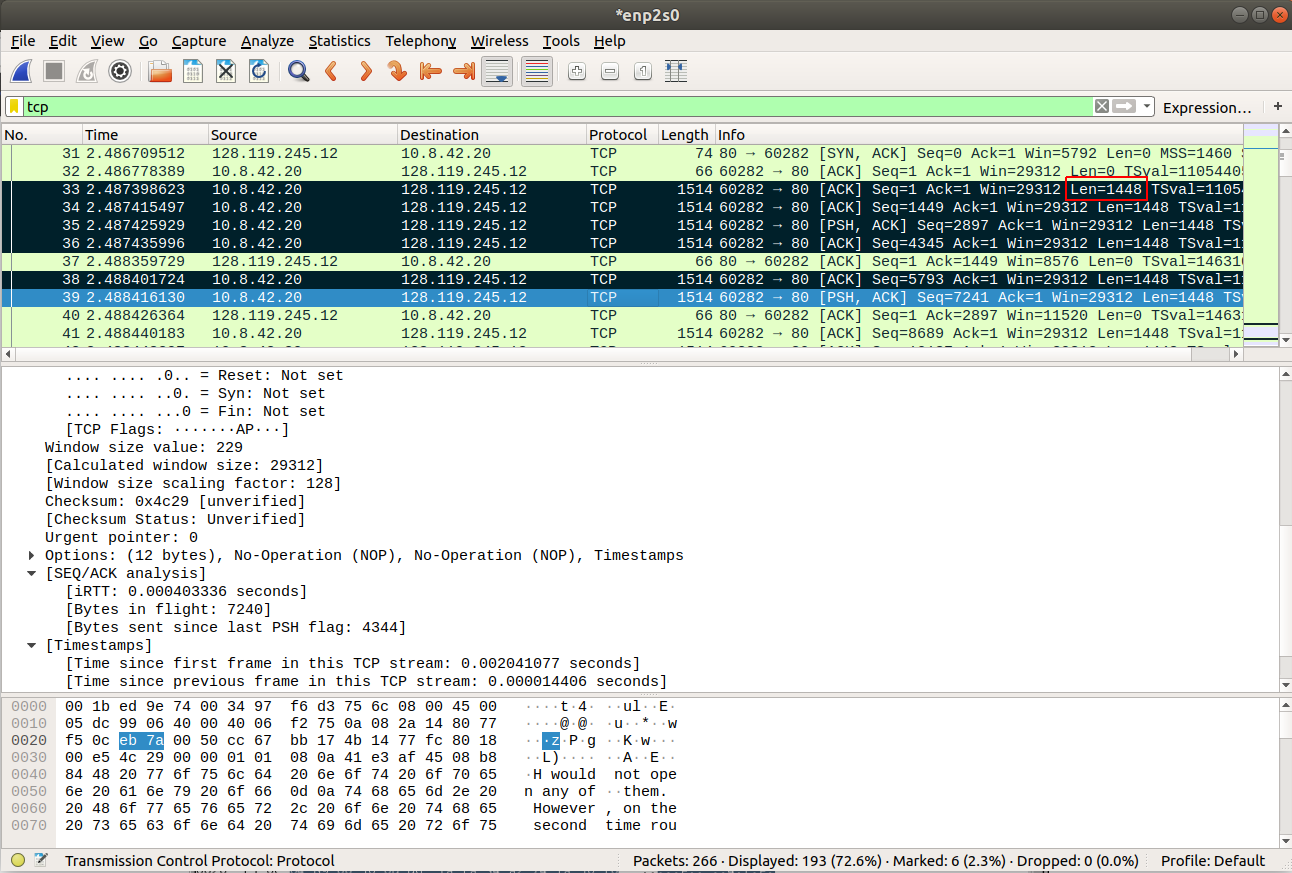
EstimateRTT = 0.875 \* 0.000970271 + 0.125 \* 0.000790432 = 0.000947791

**

*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 gaia.cs.umass.edu server. Then select: *Statistics->TCP Stream Graph->Round Trip Time Graph.*

1. What is the length of each of the first six TCP segments?

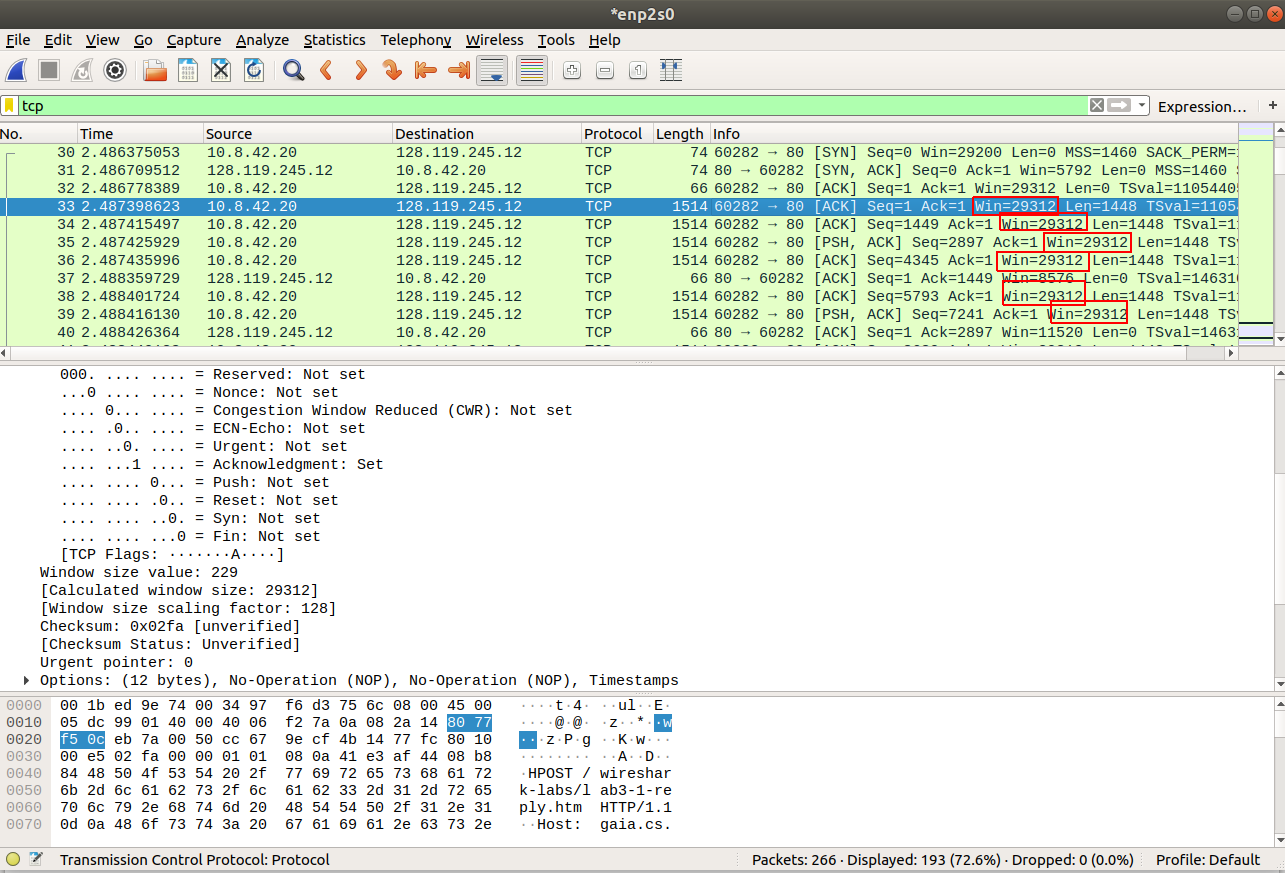
Length of each of packet of the first six packets is 1448 bytes



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

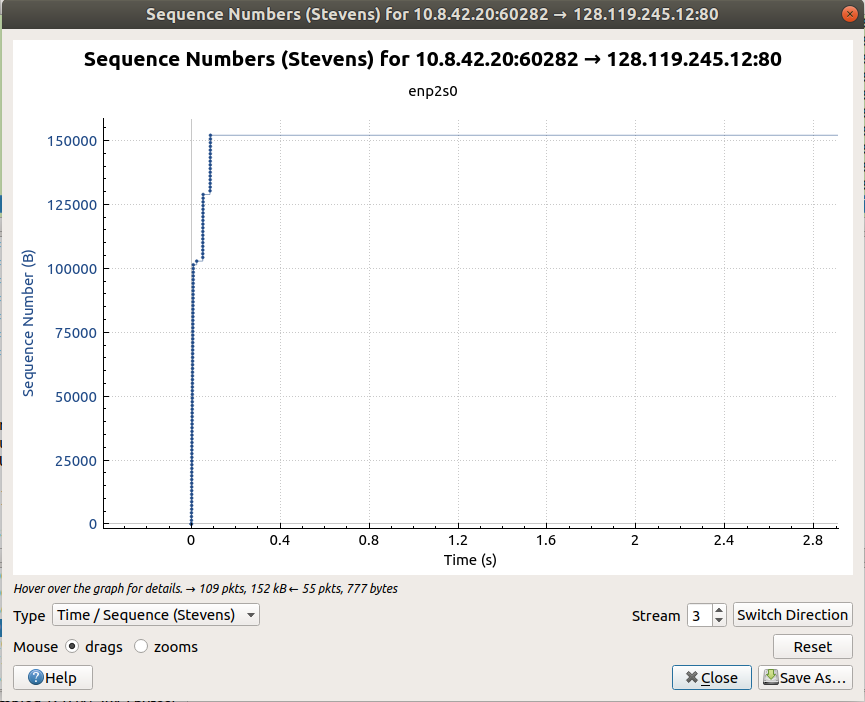
The sender is never throttled due to lacking of receiver buffer space because the buffer window size is increased until the maxium of the receiver buffer size.

|  |  |
| --- | --- |
| Segment | Available buffer space advertised at the received |
| 1 | 29312 |
| 2 | 29312 |
| 3 | 29312 |
| 4 | 29312 |
| 5 | 29312 |
| 6 | 29312 |



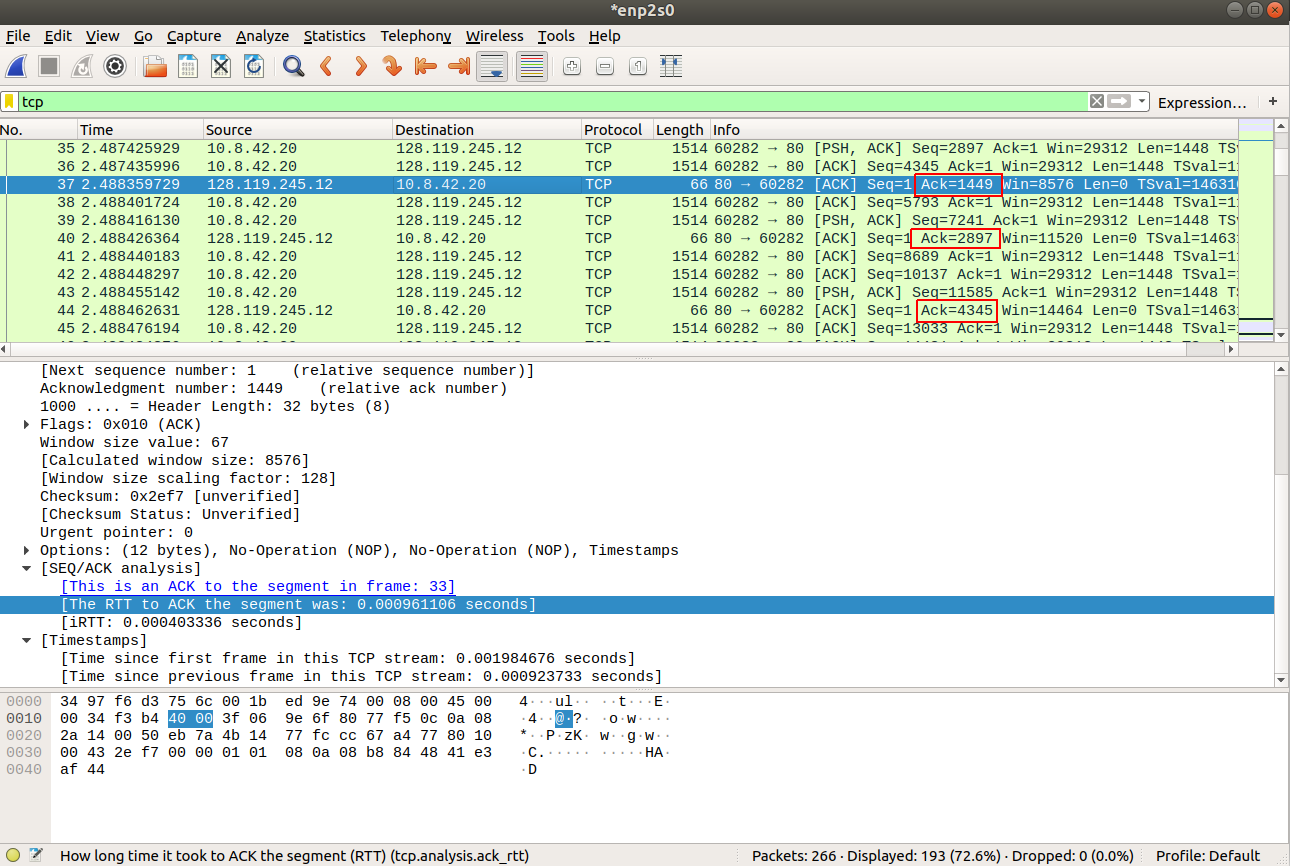
1. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

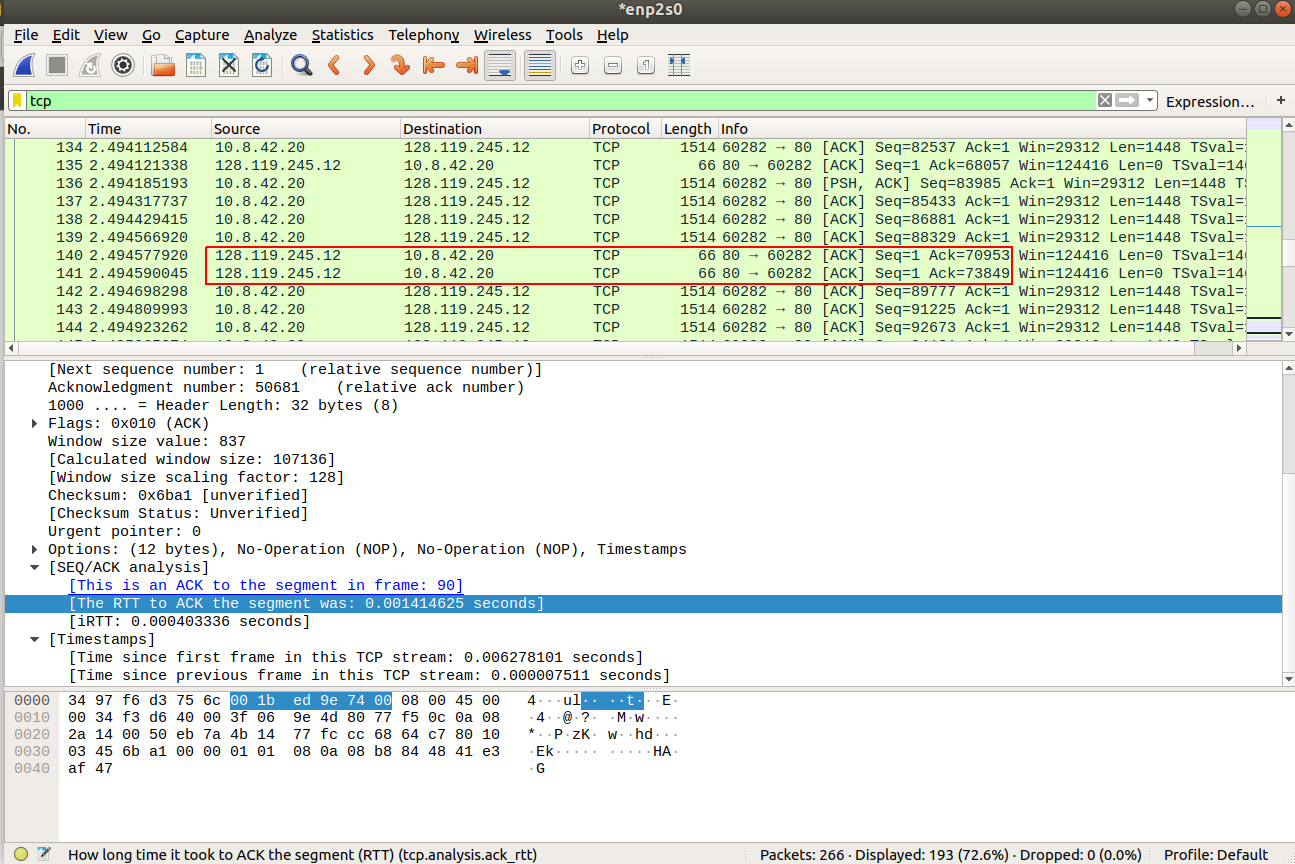
There is no segments that has to be retransmitted. In the Time/Sequence (Stevens) graph, we can see that all the sequence number are increasing steadily. If there is any restranmitted segment, the sequence number of this segment must be smaller than its neighboring ones.



1. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 247 in the text).

The receiver typically acknowledge in an ACK 1448 bytes. We can see that in the first ACK the value is 1449 bytes and in the next ACK the value is 2897 bytes. So, we minus between two consecutive times: 2897 – 1449 = 1448 bytes.

By inspecting other segments, we can see that the receiver is also ACKing every other received segment. At the packet number 141 , it acknowledged 73849 – 70953 = 2896 = 1448 \* 2



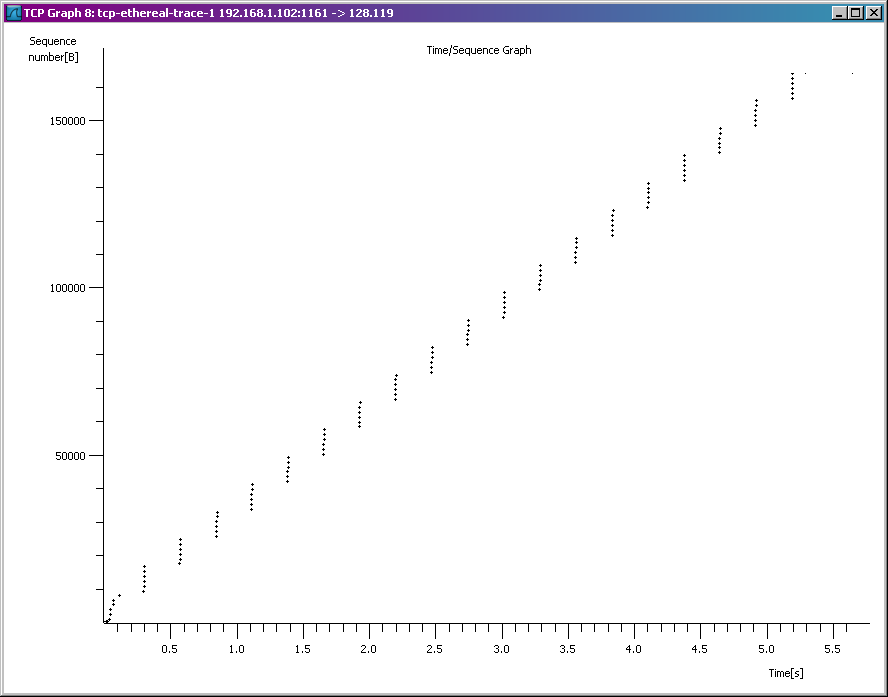
1. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

Throughput =Window size / RTT = 29312 / 0.000961106 = 30498196 bytes/s = 30498 kb/s

4. TCP congestion control in action

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 out data.

* 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 similar to the following plot, which was created from the captured packets in the packet trace [*tcp-ethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace) (see earlier footnote ):



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.

Answer the following questions for the TCP segments the packet trace [*tcp-ethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace)

1. Use the *Time-Sequence-Graph(Stevens*) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. 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.

By observing the plot, we can see that the slow-start phase only lasts for first 1-1.5 second. Afterwards, it seems that the TCP session is always in congestion avoidance state. In this case, we do not observe the expected linear increase behaviour, i.e. the TCP transmit window does not grow linearly during this phase.

1. Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu.

**THE END.**