**Lab 2 TCP**

**Objectives**

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, please first review sections 3.5 and 3.7 in the textbook.

**Procedures**

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

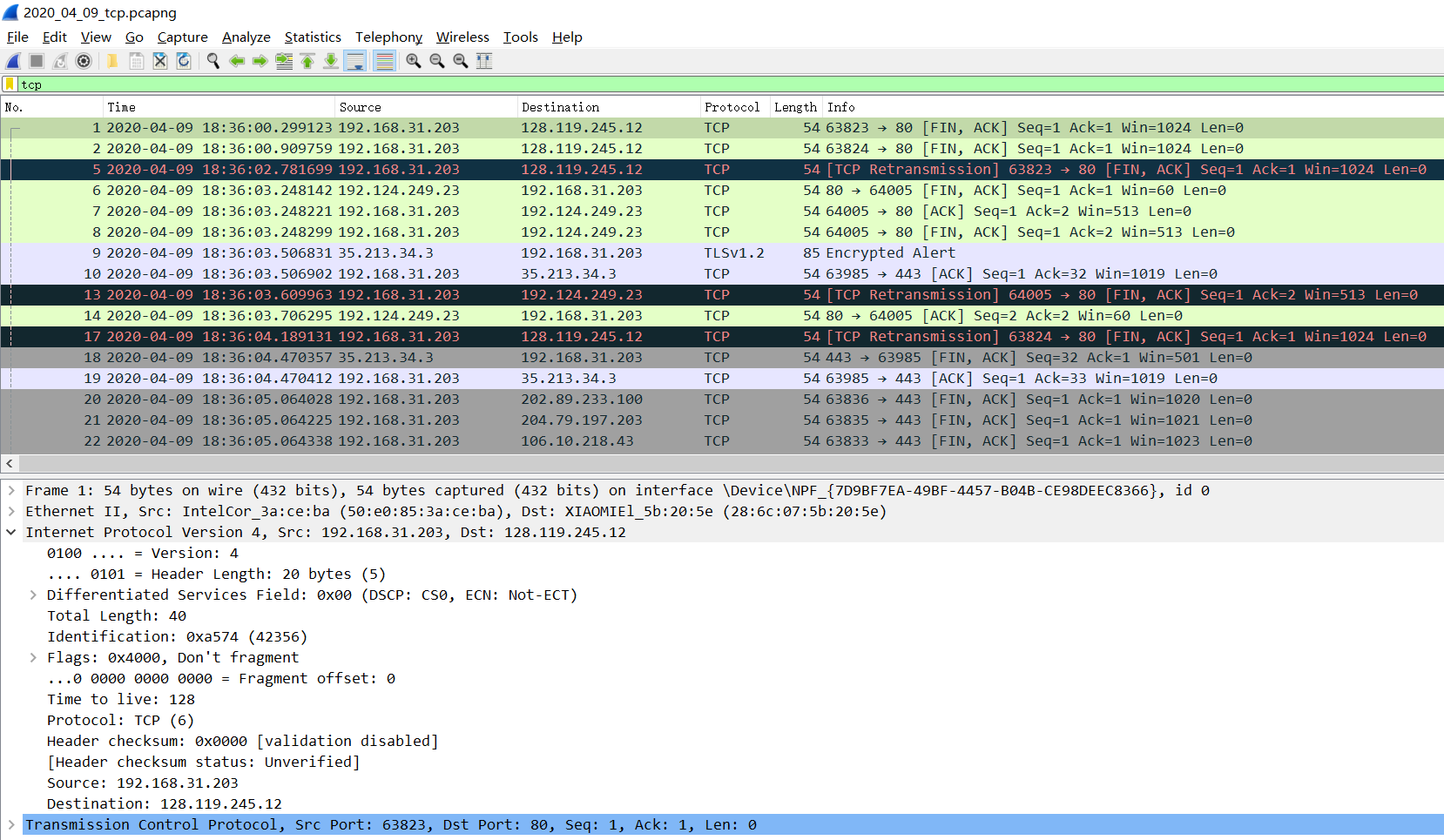
Before beginning our exploration of TCP, we’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text 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:

* Download the attached txt file alice.txt in the lab repository from the website www.github.com/network-distributed, which is 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>.
* You should see a screen that looks like:



* Use the *Browse* button in this 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.



Save this capture trace into a pcapng file. You may use it whenever you explore the questions 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 one 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, by opening the Wireshark captured packet pcapng file you saved in the first step. Whenever possible, when answering a question you should include in your lab report a printout of the packet(s) within the trace that you used to answer the question asked. Annotate to explain your answer. To print a packet, use *File->Print->Print to pdf file*, 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 the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window” (refer to Figure 2 in the “Getting Started with Wireshark” Lab if you’re uncertain about the Wireshark windows.

**Client Computer (source)**

**IP address: 10.131.232.96**

**TCP port number: 54651**

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?

**Destination computer: gaia.cs.umass.edu**

**IP address: 128.119.245.12**

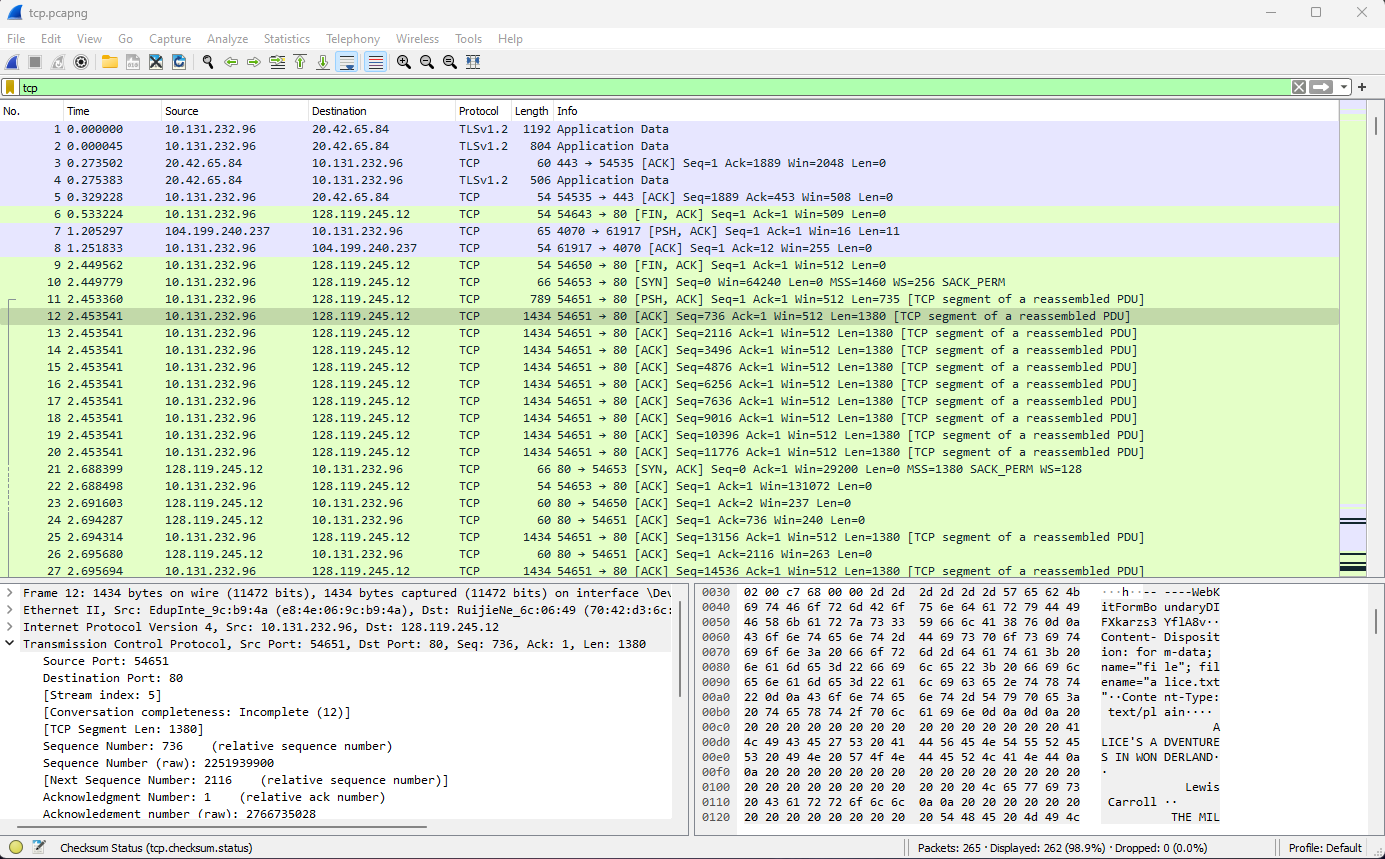
**TCP port number: 80**

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

**Client computer (source)**

**IP address: 10.131.232.96**

**TCP port number: 54651**



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:

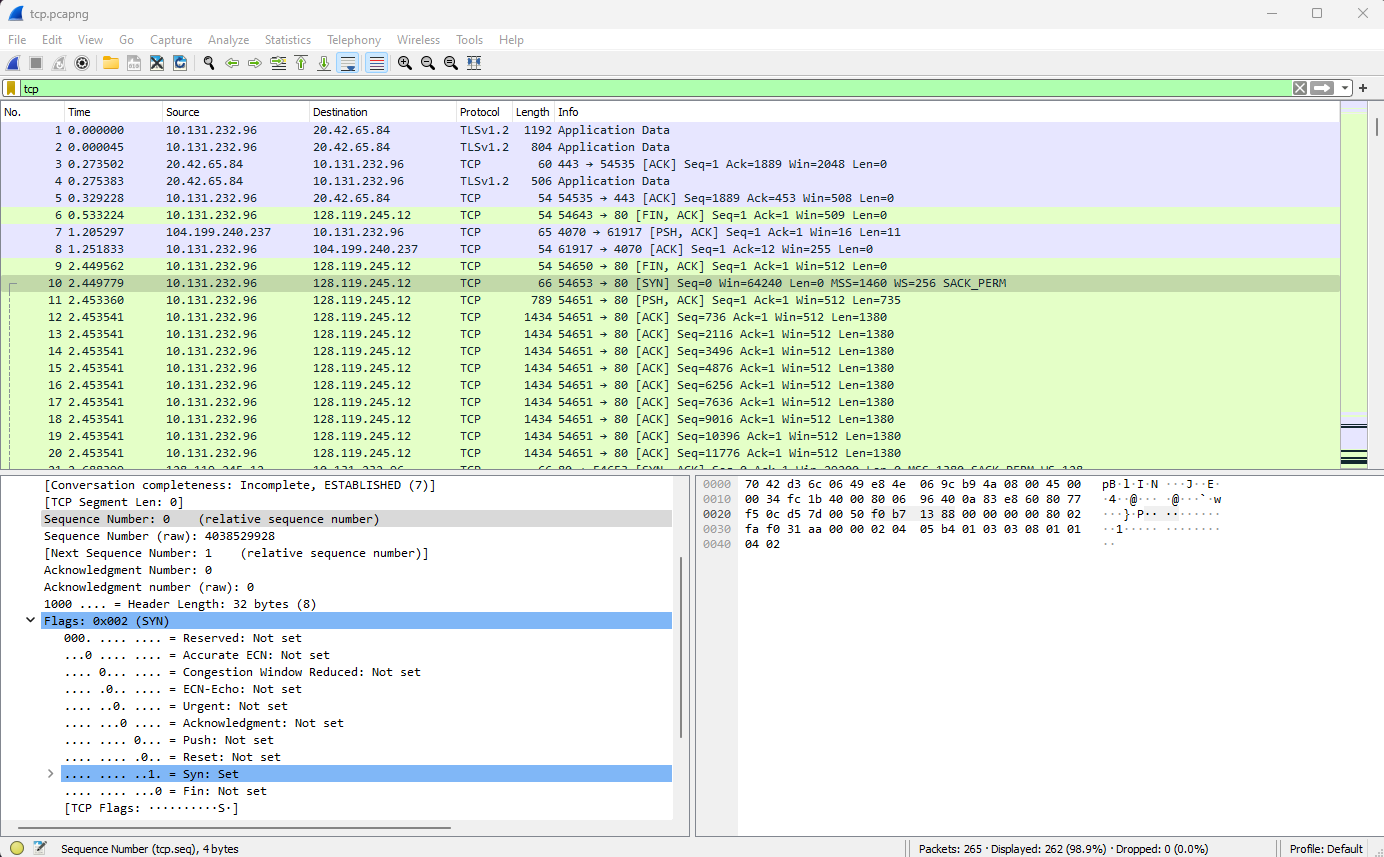
1. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?

**The Sequence number of the TCP SYN segment is used to initiate the TCP**

**connection between the client computer and gaia.cs.umass.edu. The value is 0 in this**

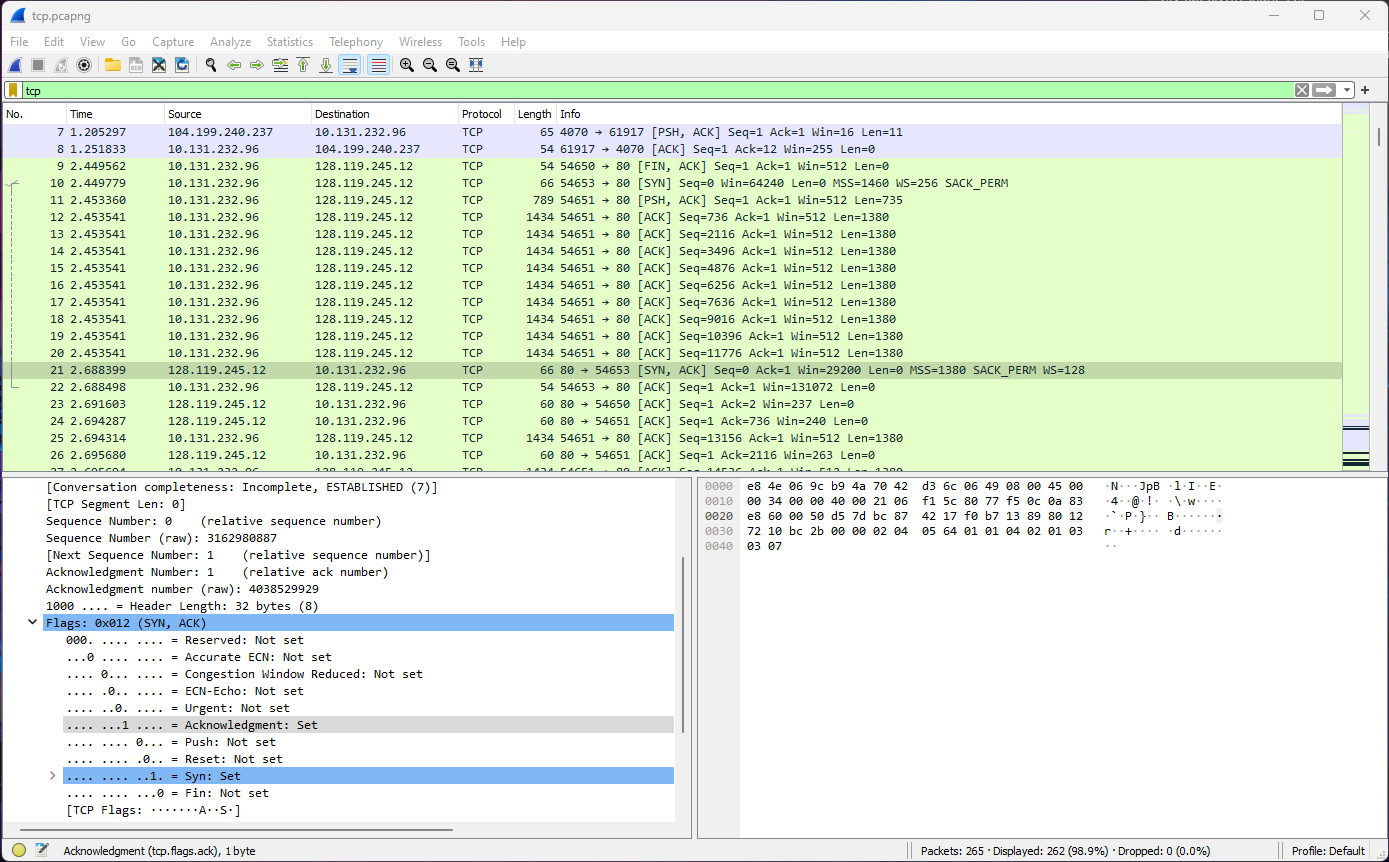
**trace.**

**The SYN flag is set to 1 and it indicates that this segment is a SYN segment.**



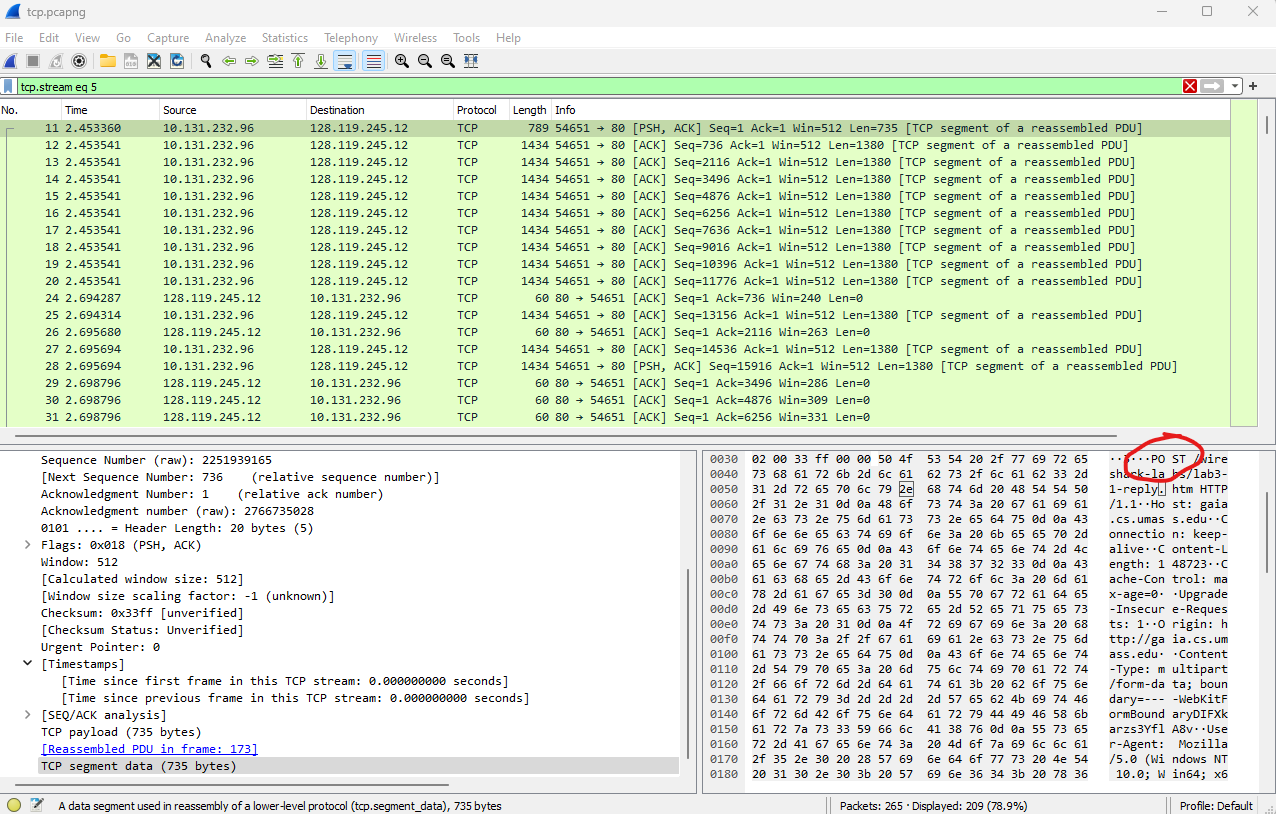
1. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK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

**In this trace, the SYNACK segment sent from gaia.cs.umass.edu to the client computer has a Sequence number of 0 in reply to the SYN. The value of the ACKnowledgement field in the SYNACK segment is determined by adding 1 to the initial sequence number of the SYN segment sent by the client computer, which is also 1. The SYN and Acknowledgement flags in the segment are both set to 1, indicating that it is a SYNACK segment.**



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.

**Packet No. 11 with stream index of 5 is the TCP segment containing the HTTP POST command. The sequence number of this segment has the value of 1 (relative sequence number) or 2251939165 (raw).**



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 in textbook) 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 in the textbook for all subsequent segments.

*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.*

**The HTTP POST segment is considered as the first segment. Segments 1-6 are packets 11, 12, 13, 14, 15, and 16 in this trace respectively.**

Segment 1 sequence number: **1**

Segment 2 sequence number: **736**

Segment 3 sequence number: **2116**

Segment 4 sequence number: **3496**

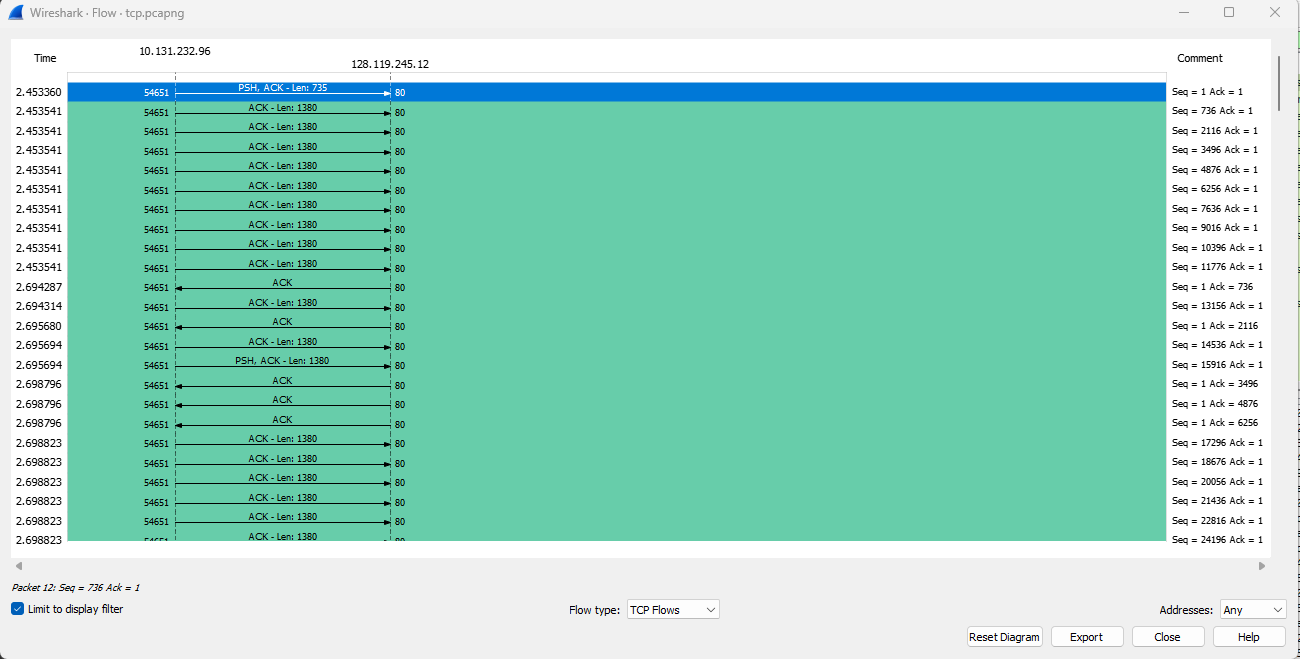
Segment 5 sequence number: **4876**

Segment 6 sequence number: **6256**

**The sending time and the received time of the ACKs recorded in the following table:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Send time | ACK received time | RTT (s) |
| Segment 1 | 2.449779 | 2.453360 | 0.003581 |
| Segment 2 | 2.453360 | 2.453541 | 0.000181 |
| Segment 3 | 2.453360 | 2.453541 | 0.000181 |
| Segment 4 | 2.453360 | 2.453541 | 0.000181 |
| Segment 5 | 2.453360 | 2.453541 | 0.000181 |
| Segment 6 | 2.453360 | 2.453541 | 0.000181 |

**RTT = ACK time - Send time**



**EstimatedRTT = 0.875 \* EstimatedRTT + 0.125 \* SampleRTT**

**Segment 1: 0.003581 seconds**

**Segment 2: 0.875 \* 0.003581 + 0.125 \* 0.000181 = 0.003156 seconds**

**Segment 3: 0.875 \* 0.003156 + 0.125 \* 0.000181 = 0.002784 seconds**

**Segment 4: 0.875 \* 0.002784 + 0.125 \* 0.000181 = 0.002458 seconds**

**Segment 5: 0.875 \* 0.002458 + 0.125 \* 0.000181 = 0.002173 seconds**

**Segment 6: 0.875 \* 0.002173 + 0.125 \* 0.000181 = 0.001924 seconds**

1. What is the length of each of the first six TCP segments?[[1]](#footnote-0)

**Segment 1: 735**

**Segment 2: 1380**

**Segment 3: 1380**

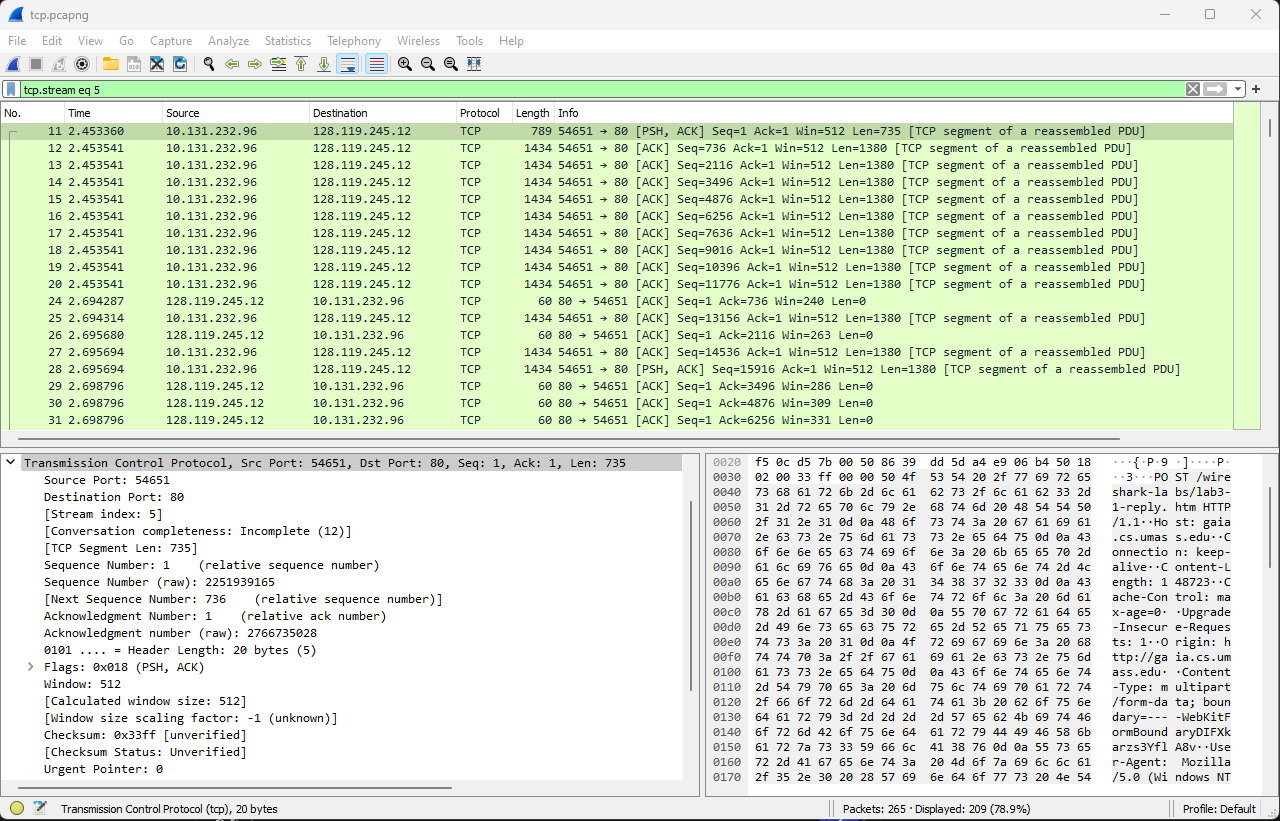
**Segment 4: 1380**

**Segment 5: 1380**

**Segment 6: 1380**

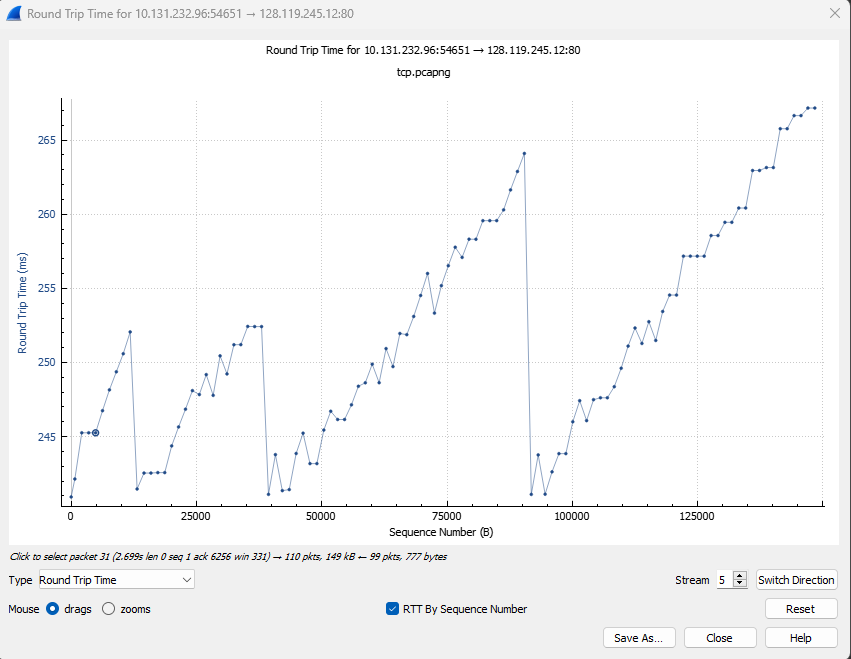
1. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

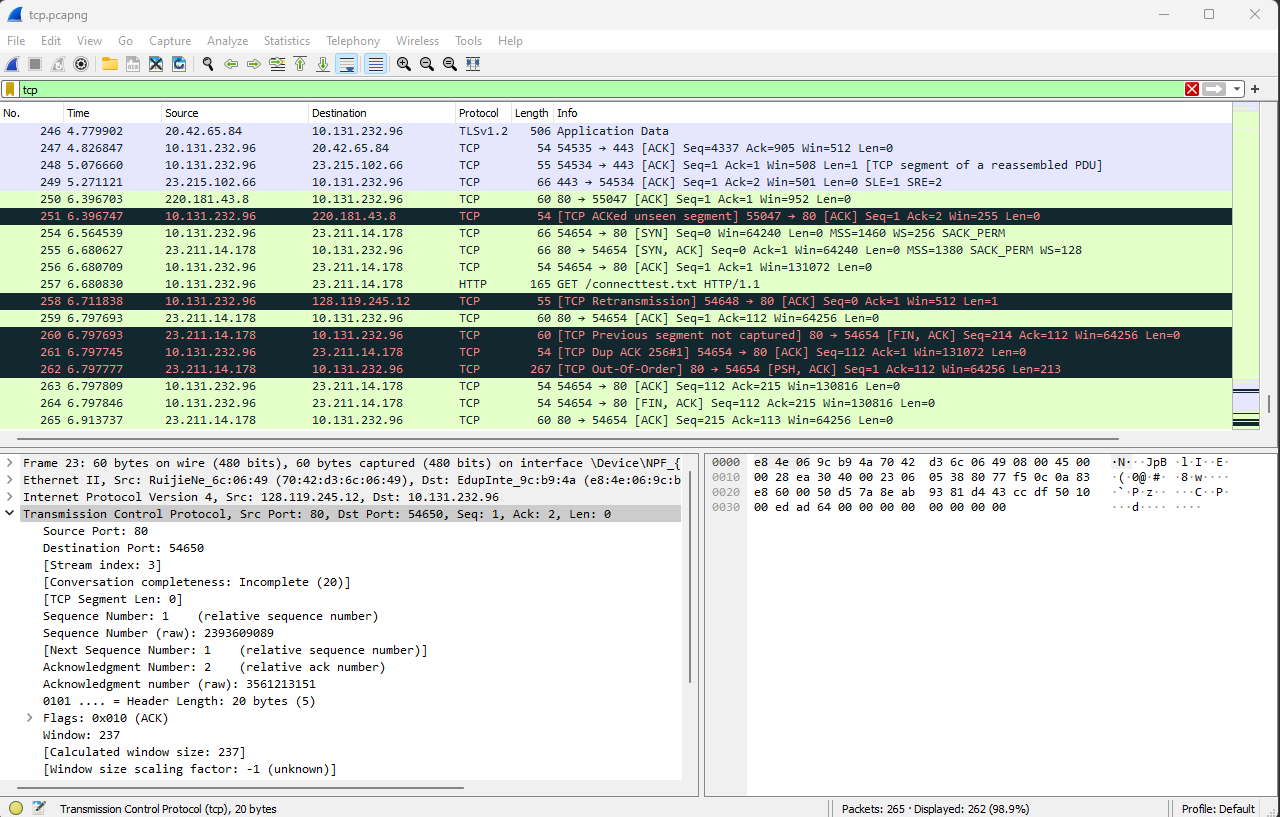
**The minimum amount of buffer space (receiver window) advertised at the gaia.cs.um.ass.edu for the entire trace is 512 bytes, which shows in the first acknowledgement from the server. No, it does not throttle due to lacking of receiver buffer space by inspecting this trace.**



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

**Yes, there are re-transmitted segments in the trace file. We can verify this by checking the sequence numbers of the TCP segments in the trace file. In the *Time-Sequence-Graph (Stevens)* of this trace, all sequence numbers from the source (10.131.232.96:54651) to the destination (128.119.245.12) are fluctuating and some re-transmitted segments are smaller than those of its neighboring segments. Also, in the trace file, some packets from the destination address display a [TCP Retransmission] label in its info column.**

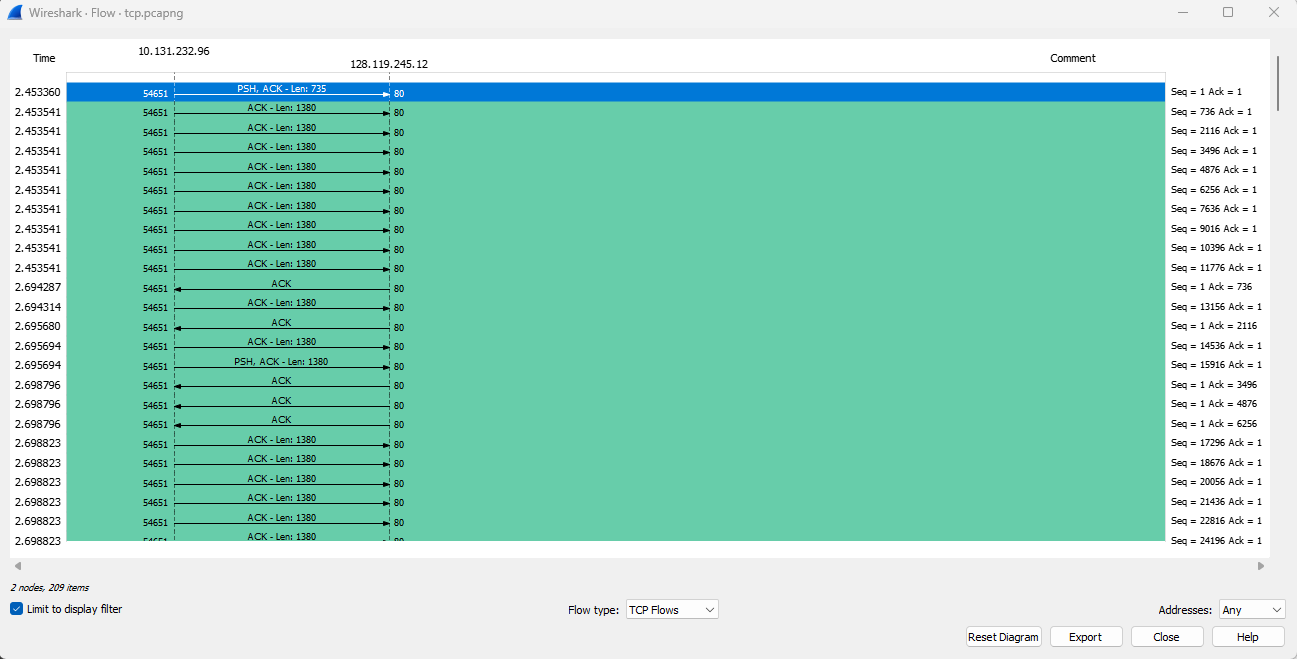




1. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 250 in the text).

|  |  |  |
| --- | --- | --- |
|  | **Acknowledged sequence number** | **Acknowledged data** |
| **ACK 1** | **736** | **736** |
| **ACK 2** | **2116** | **1380** |
| **ACK 3** | **3496** | **1380** |
| **ACK 4** | **4876** | **1380** |
| **ACK 5** | **6256** | **1380** |
| **ACK6** | **7636** | **1380** |
| **...** | **…** | **…** |

**Answer: The receiver is typically ACKing 1380 bits of data. There are cases where a receiver ACKs every other received segment. This can be seen when there are two ACKs in a row.**



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

**Calculation:**

**Total Data Transmitted: 139459 -1 = 139458 bytes**

**Total Transmission time: 3.511820 - 2.453360 = 1.05846 seconds**

**Since the computation of TCP throughput largely depends on the selection of the averaging time period, we first select the average time period as the whole connection time. Then, the average throughput for the TCP connection is computed as the ration between the total amount of data and the total transmission time. The total amount of data transmitted can be computed by the difference between the sequence number of the first TCP Segment (1 bytes for No. 11 Segment) and the acknowledged sequence number of the last ACK (139459 bytes for No. 226 segment).**

**The whole transmission time is the difference of the time instant of the first TCP segment (2.453360 seconds for No. 11 segment) and the time instant of the last ACK (3.511820 seconds for No. 226 segment). Therefore, the total transmission time is 1.05846 seconds. Hence, the throughput for the TCP connection is computed as 139458 / 1.05846 and the result is 131.755 KBytes/sec.**

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](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace)* [in](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace)  :

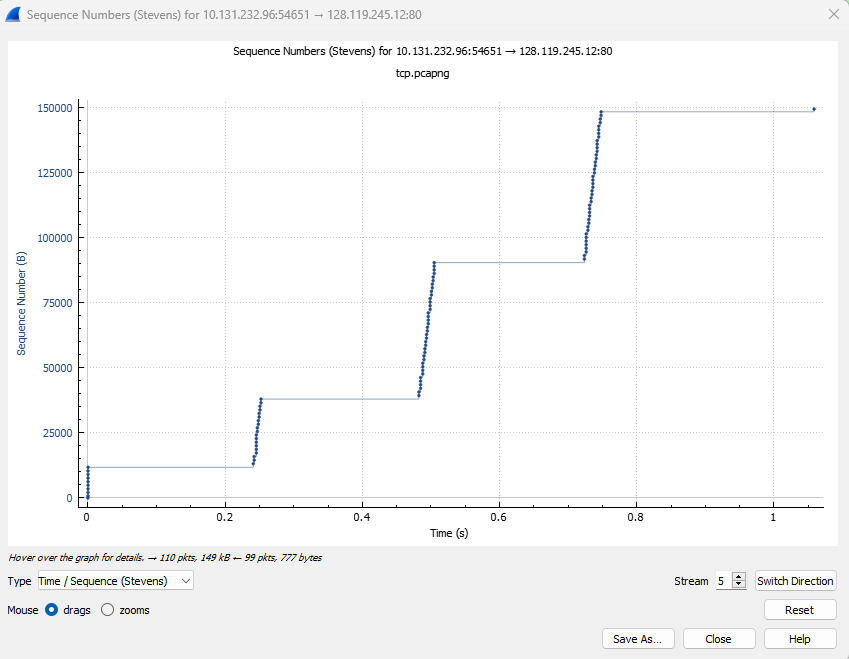


Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

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

**TCP’s slow start phase begins from packet No. 11 and ends in packet No. 20. Then, congestion avoidance takes over and resets in packet No. 25 until packet No. 52. This pattern occurs also in No. 55 - No. 109, No. 112 - 173 and then finally ends in packet No. 226 with the last ACK. TCP’s behavior is differs slightly to what we’ve studied in the text. In my graph, it displays an exponentially plotted slow start graphs but compared to the text’s graph, the plotted graph is a lot more jagged and uneven. Additionally, in my graph, it shows perfectly vertical graphs indicating congestion avoidance compared to the more gradual graphs shown in the text.**



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.

1. If your trace indicates a TCP length greater than 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload), and your computer is using an Ethernet connection, then Wireshark is reporting the wrong TCP segment length; it will likely also show only one large TCP segment rather than multiple smaller segments. Your computer is indeed probably sending multiple smaller segments, as indicated by the ACKs it receives. This inconsistency in reported segment lengths is due to the interaction between the Ethernet driver and the Wireshark software. [↑](#footnote-ref-0)