Author: Santiago Bermudez

CSC 138

Section 4

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.

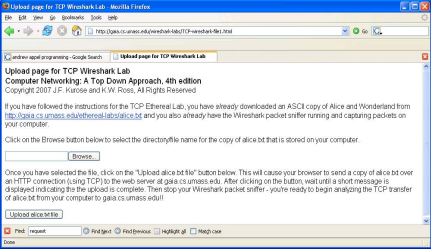
Do the following:

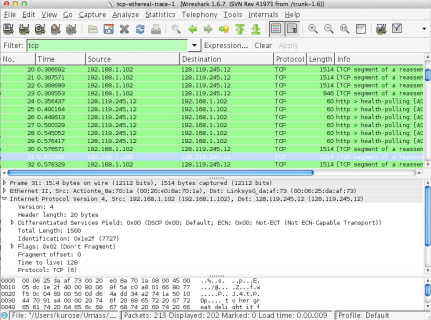
• Start up your web browser. Go the http://gaia.cs.umass.edu/wiresharklabs/alice.txt and retrieve an ASCII copy of *Alice in Wonderland.* Store this file somewhere on your computer. • Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html. • You should see a screen that looks like:

• 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.





If you are unable to run Wireshark on a live network connection, you can download a packet trace file that was captured while following the steps above on one of the author’s computers2. You may well find it valuable to download this trace even if you’ve captured your own trace and use it, as well as your own trace, when 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 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.

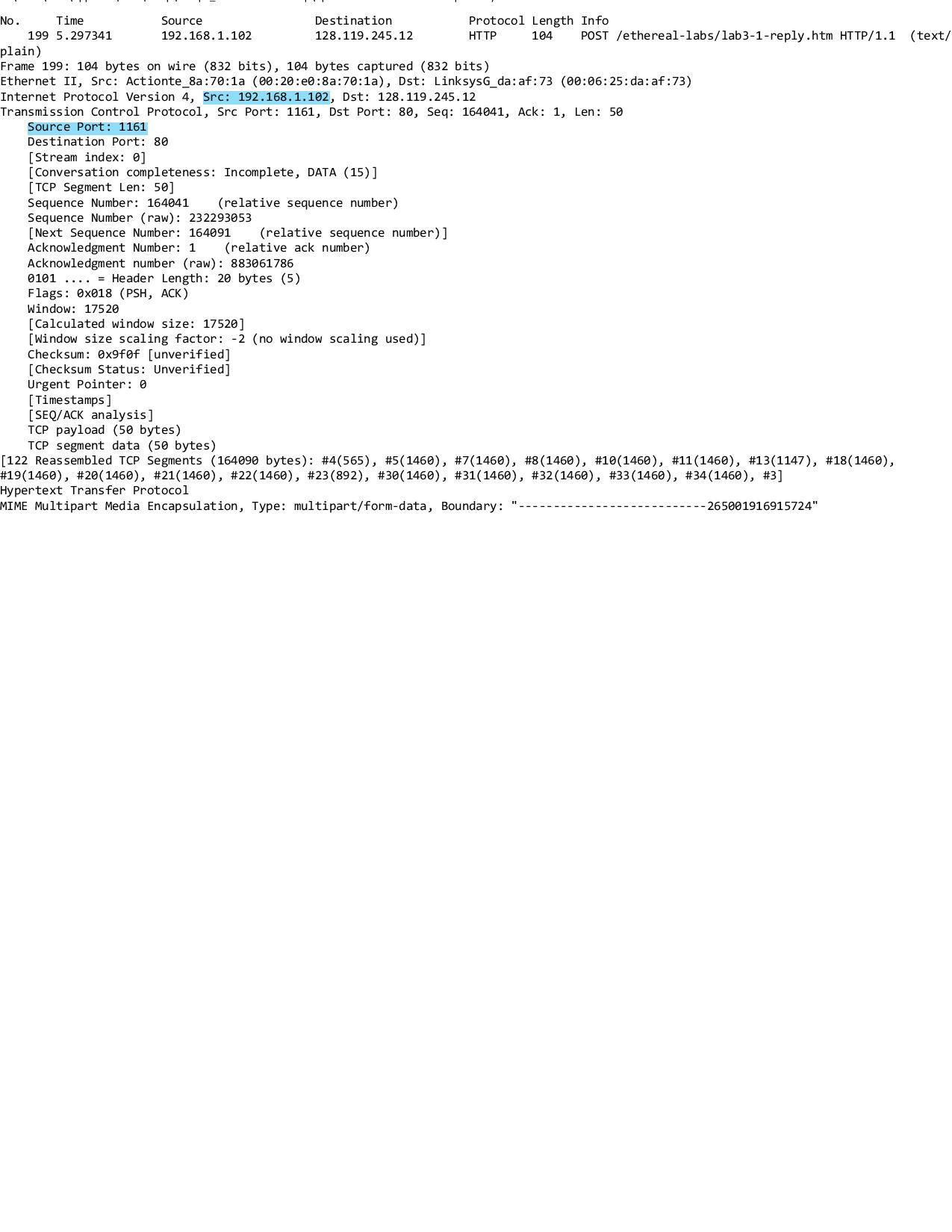
Answer the following questions, by opening the Wireshark captured packet file *tcpethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (that is download the trace and open that trace in Wireshark; see footnote 2). Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout3 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 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: 192.168.1.102

TCP port number: 1161



2. 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: 128.119.245.12

TCP port number: 80

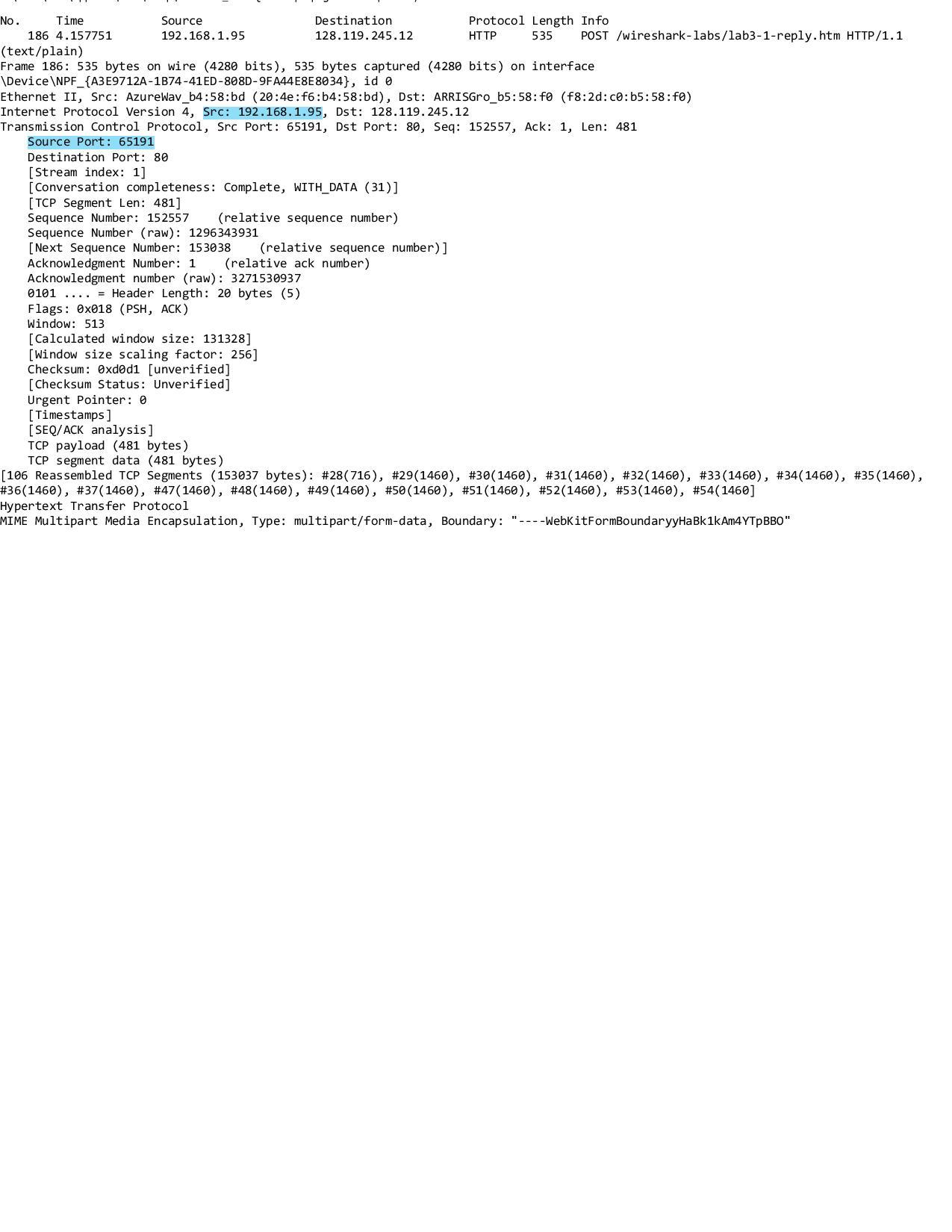


If you have been able to create your own trace, answer the following question:

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

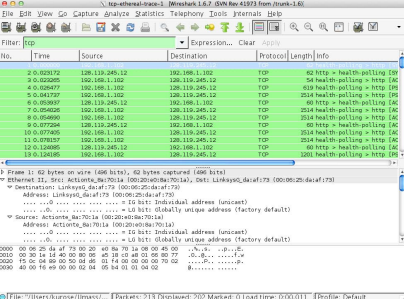
IP address: 192.168.1.95

TCP port number: 65191



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:

3 What do we mean by “annotate”? If you hand in a paper copy, please highlight where in the printout you’ve found the answer and add some text (preferably with a colored pen) noting what you found in what you ‘ve highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.



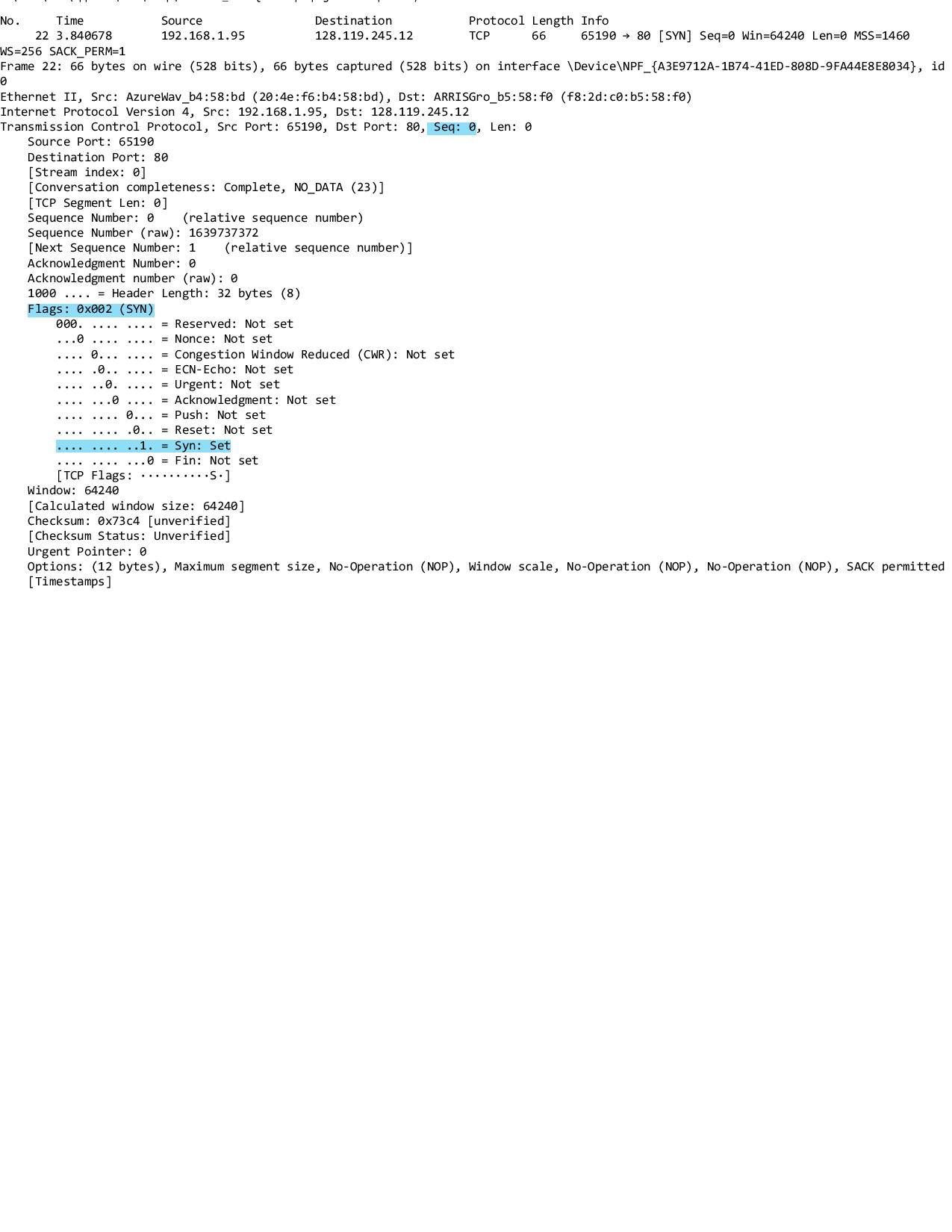
This is what we’re looking for -a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured (and/or the packet trace *tcp ethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wiresharktraces.zip; see earlier footnote) to study TCP behavior in the rest of this lab.

3. TCP Basics

Answer the following questions for the TCP segments:

4. 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 segment used to initiate the TCP connection is 0. If you look in the Flags section, you can see that the SYN flag is set to 1 which tells us that this is a SYN segment.



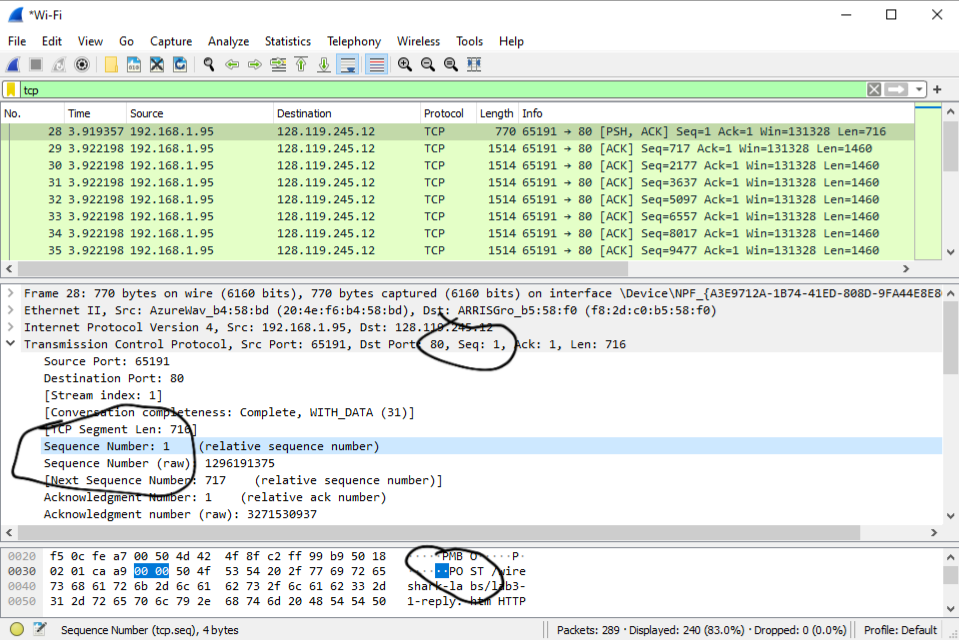
5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK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

The sequence number of the SYNACK segment sent by gaia.cs.umass.edu is 0. The value of the Acknowledgement field in the SYNACK segment is 1. Gaia.cs.umass.edu determined that value by adding 1 to the initial sequence number of the SYN segment from the client computer. In the segment, you determine that it is a SYNACK segment if both the SYN flag and ACKnowledgement flag are set to 1.



6. 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 is 1.



7. 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.

*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.*

Segment 1 sequence number: 1

Segment 2 sequence number: 566

Segment 3 sequence number: 2026

Segment 4 sequence number: 3486

Segment 5 sequence number: 4946

Segment 6 sequence number: 6406

Segment 1 sent: 0.026477

Segment 2 sent: 0.041737

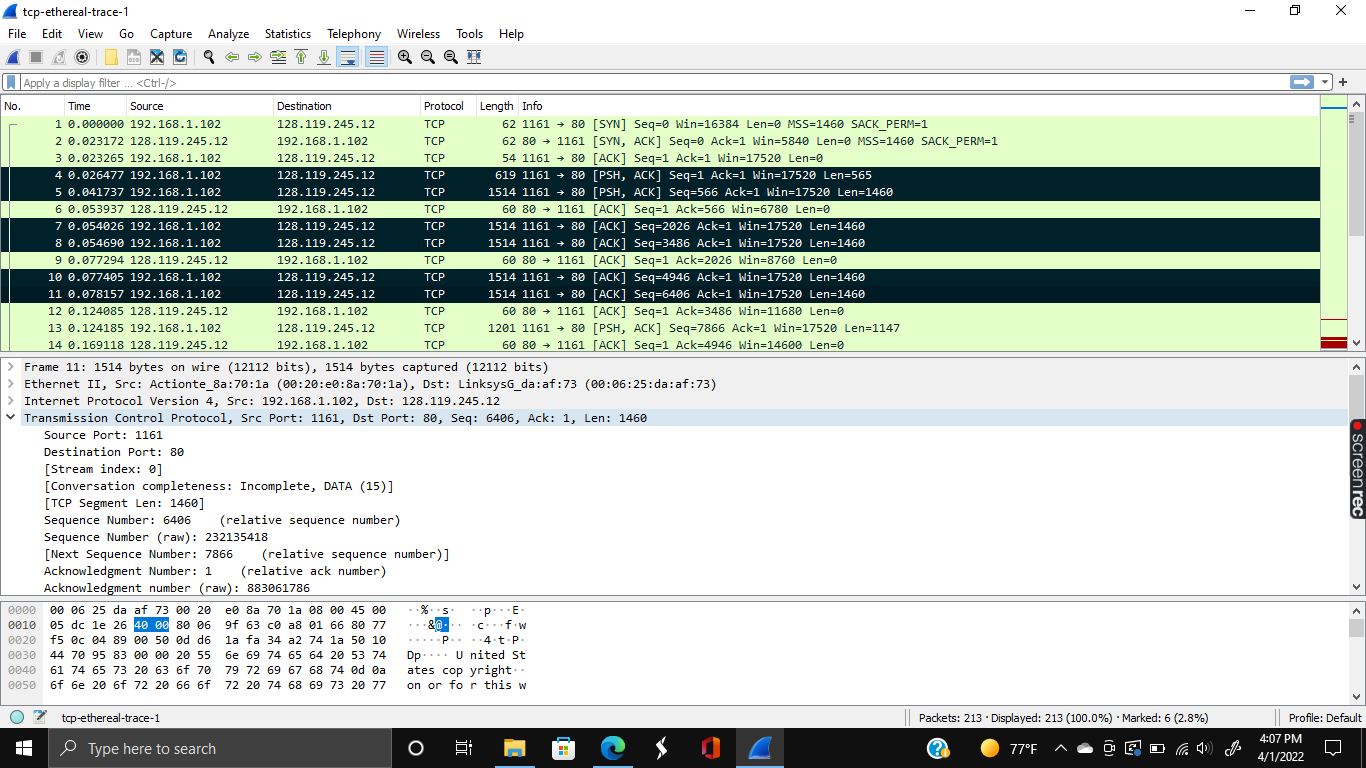
Segment 3 sent: 0.054026

Segment 4 sent: 0.054690

Segment 5 sent: 0.077405

Segment 6 sent: 0.078157

\*Please note that the packets being referred to are highlighted in black (\*or dark blue) in the screenshot below!



Segment 1 ACK received: 0.053937

Segment 2 ACK received: 0.077294

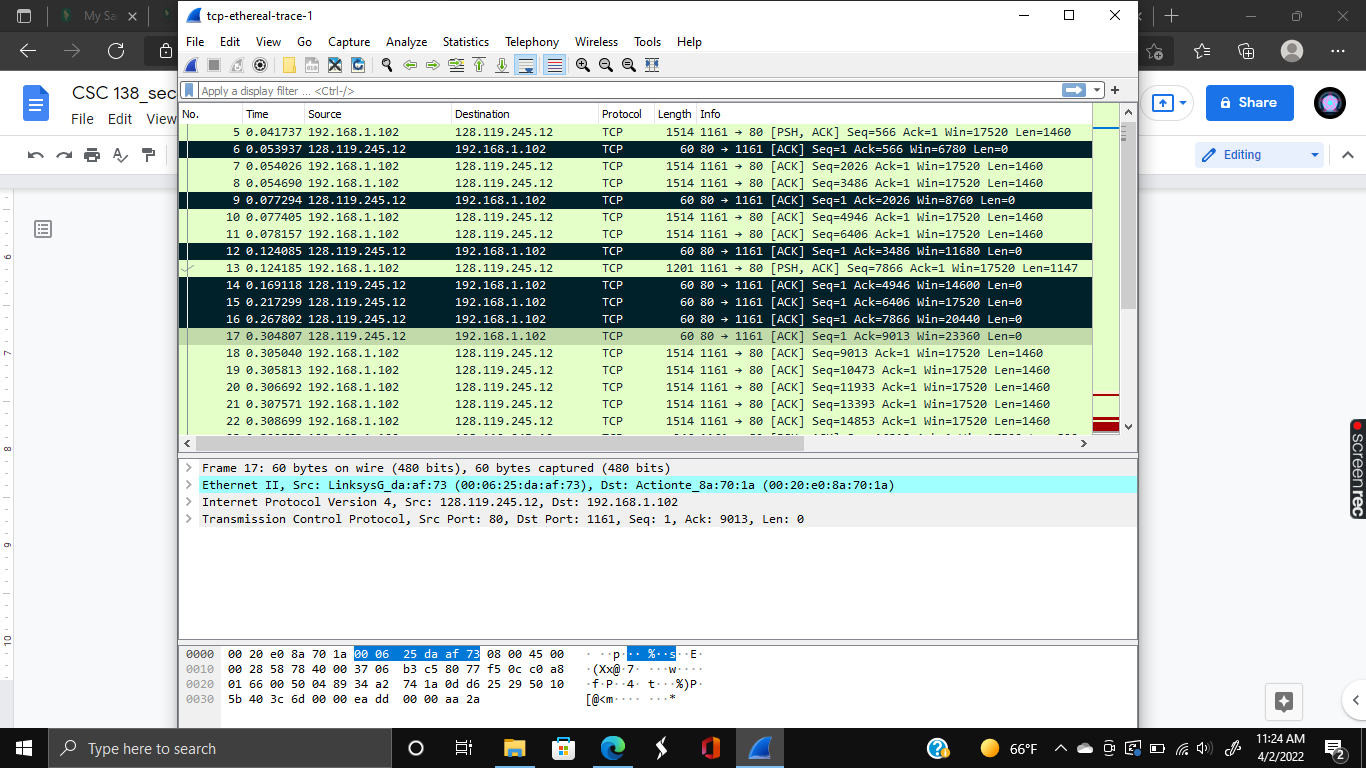
Segment 3 ACK received: 0.124085

Segment 4 ACK received: 0.169118

Segment 5 ACK received: 0.217299

Segment 6 ACK received: 0.267802

\*Just like before, the packets being referred to are highlighted in black (\*or dark blue) in the screenshot below!



Segment 1 RTT: 0.053937 - 0.026477 = 0.02746

Segment 2 RTT: 0.077294 - 0.041737 = 0.03557

Segment 3 RTT: 0.124085 - 0.054026 = 0.070059

Segment 4 RTT: 0.169118 - 0.054690 = 0.114428

Segment 5 RTT: 0.217299 - 0.077405 = 0.139894

Segment 6 RTT: 0.267802 - 0.078157 = 0.189645

Segment 1 EstimatedRTT:

EstimatedRTT = RTT for Segment 1 = 0.02746 seconds

Segment 2 EstimatedRTT:

EstimatedRTT = 0.875 \* 0.02746 + 0.125 \* 0.035557 = 0.0285

Segment 3 EstimatedRTT:

EstimatedRTT = 0.875 \* 0.0285 + 0.125 \* 0.070059 = 0.0337

Segment 4 EstimatedRTT:

EstimatedRTT = 0.875 \* 0.0337+ 0.125 \* 0.11443 = 0.0438

Segment 5 EstimatedRTT:

EstimatedRTT = 0.875 \* 0.0438 + 0.125 \* 0.13989 = 0.0558

Segment 6 EstimatedRTT:

EstimatedRTT = 0.875 \* 0.0558 + 0.125 \* 0.18964 = 0.0725

8. What is the length of each of the first six TCP segments?4

Segment 1 length: 565

Segment 2 length: 1460

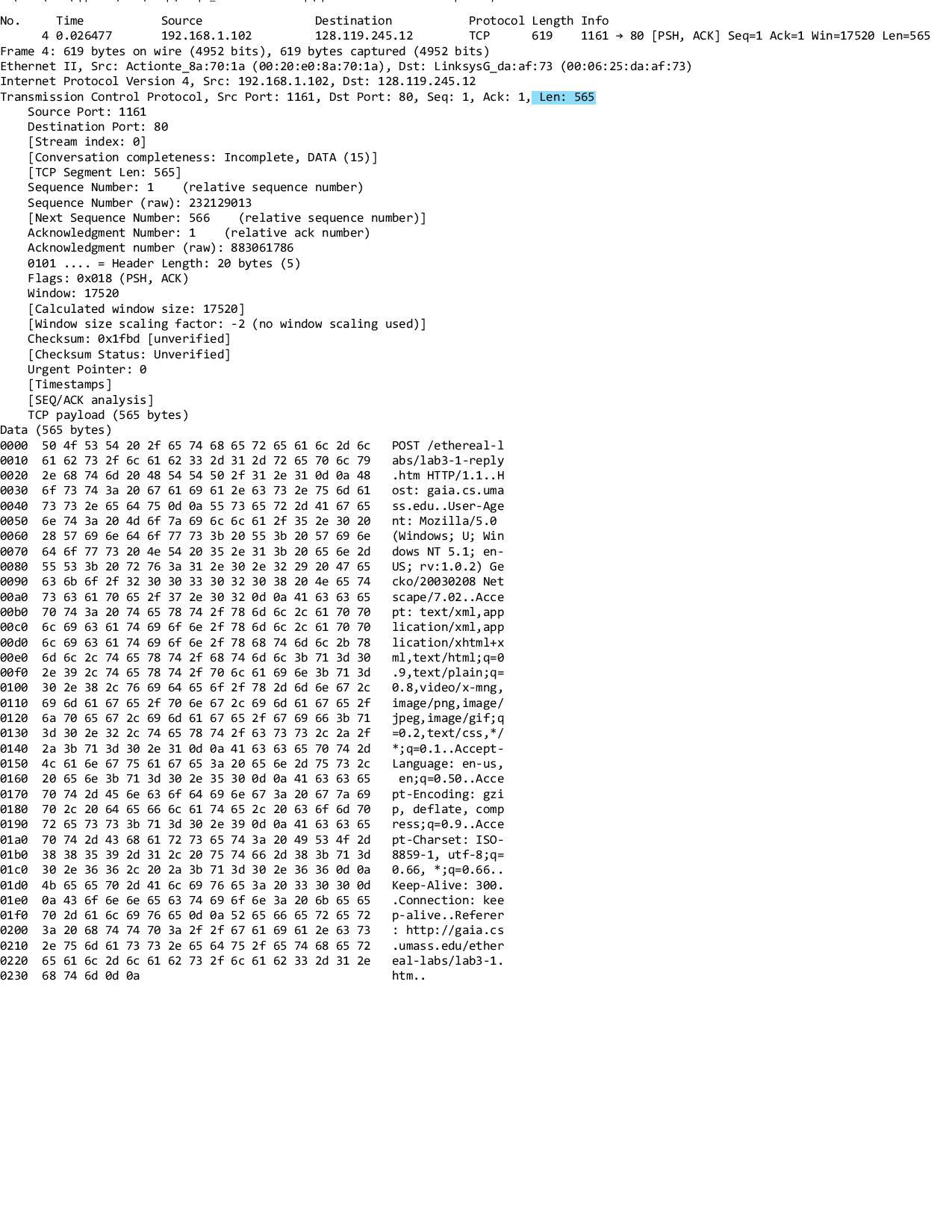
Segment 3 length: 1460

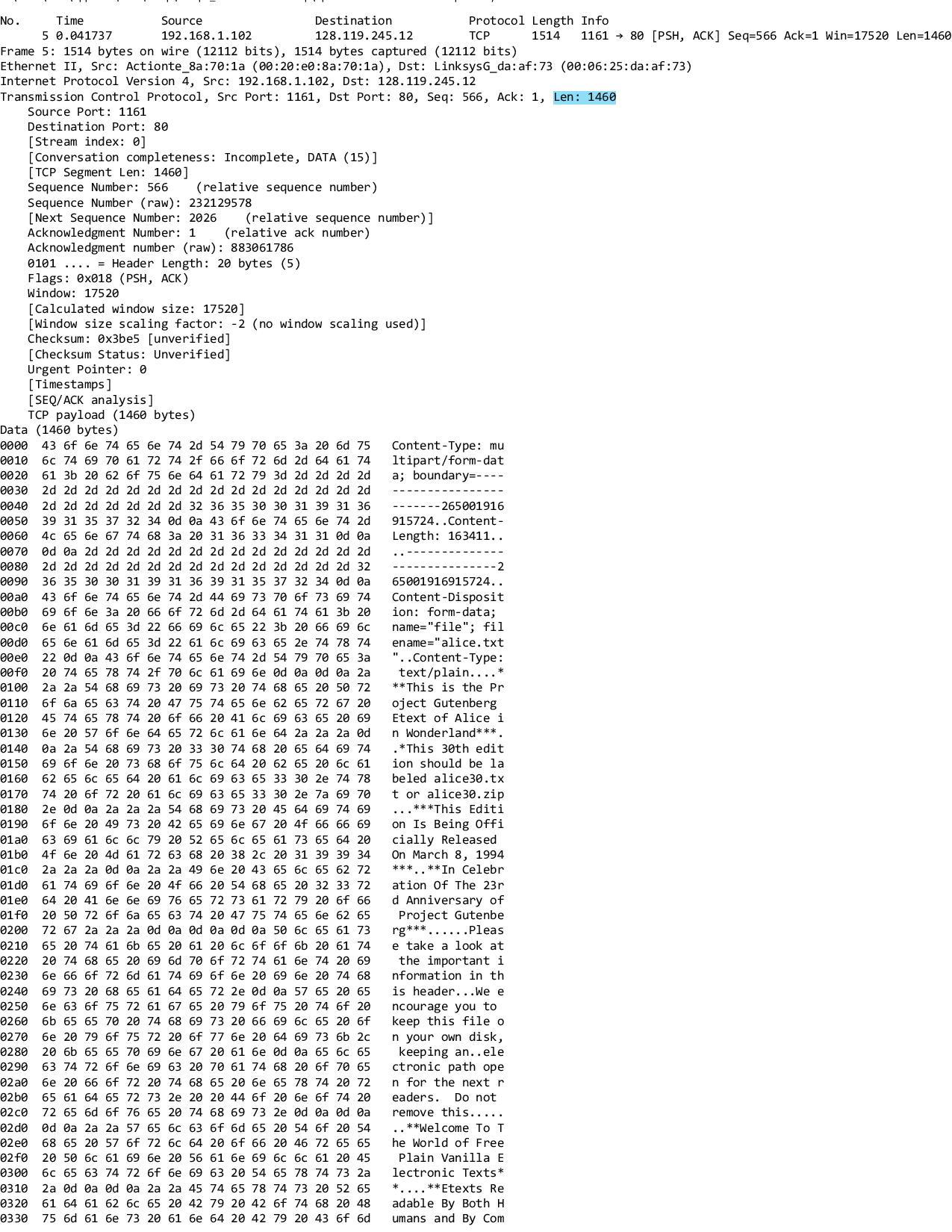
Segment 4 length: 1460

Segment 5 length: 1460

Segment 6 length: 1460

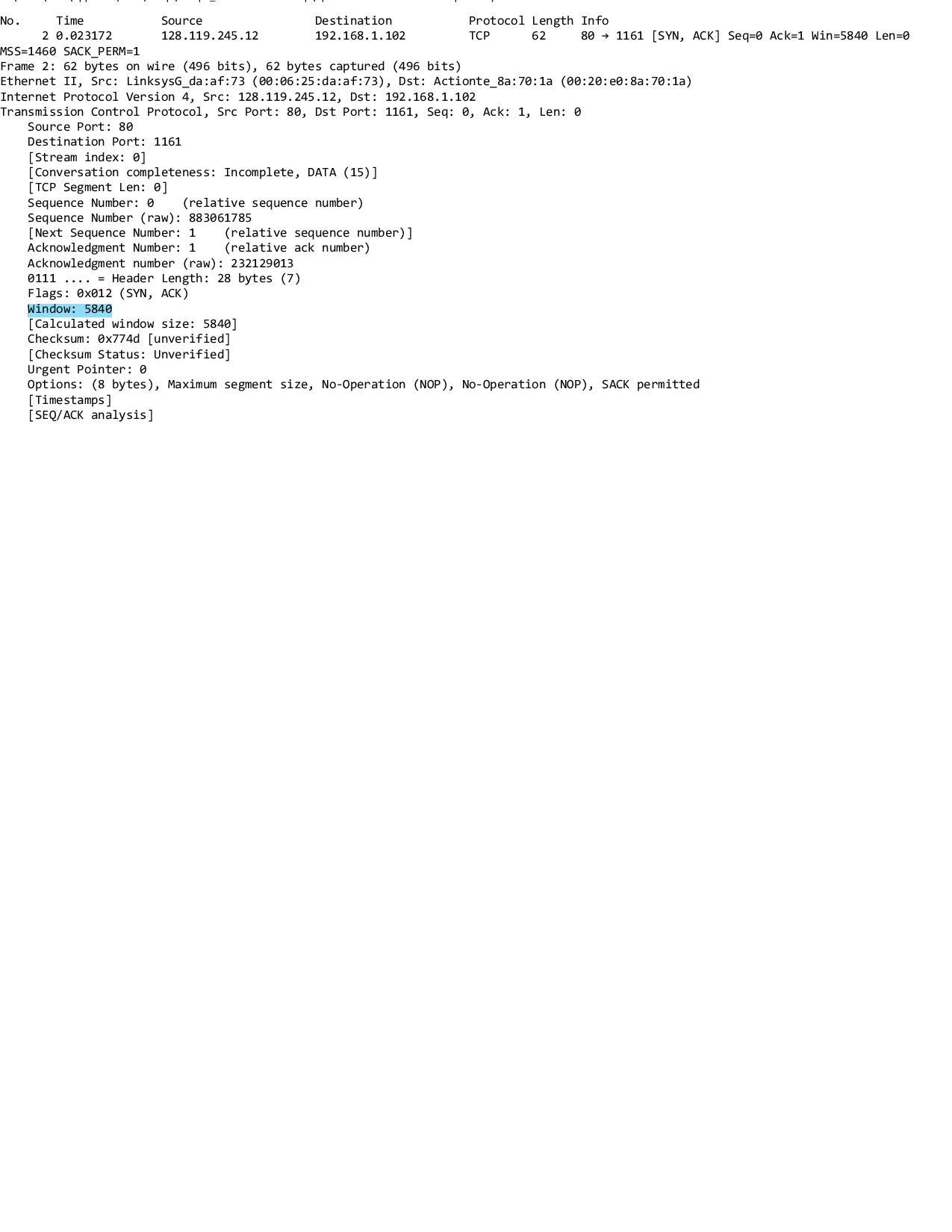
\*The first is 565 and the rest are all 1460. I could take the screenshots for all segments, but that would be too tedious. I have two posted below. Segments 2-6 all have the same length shown in the second screenshot, so this should be good enough to make the point.





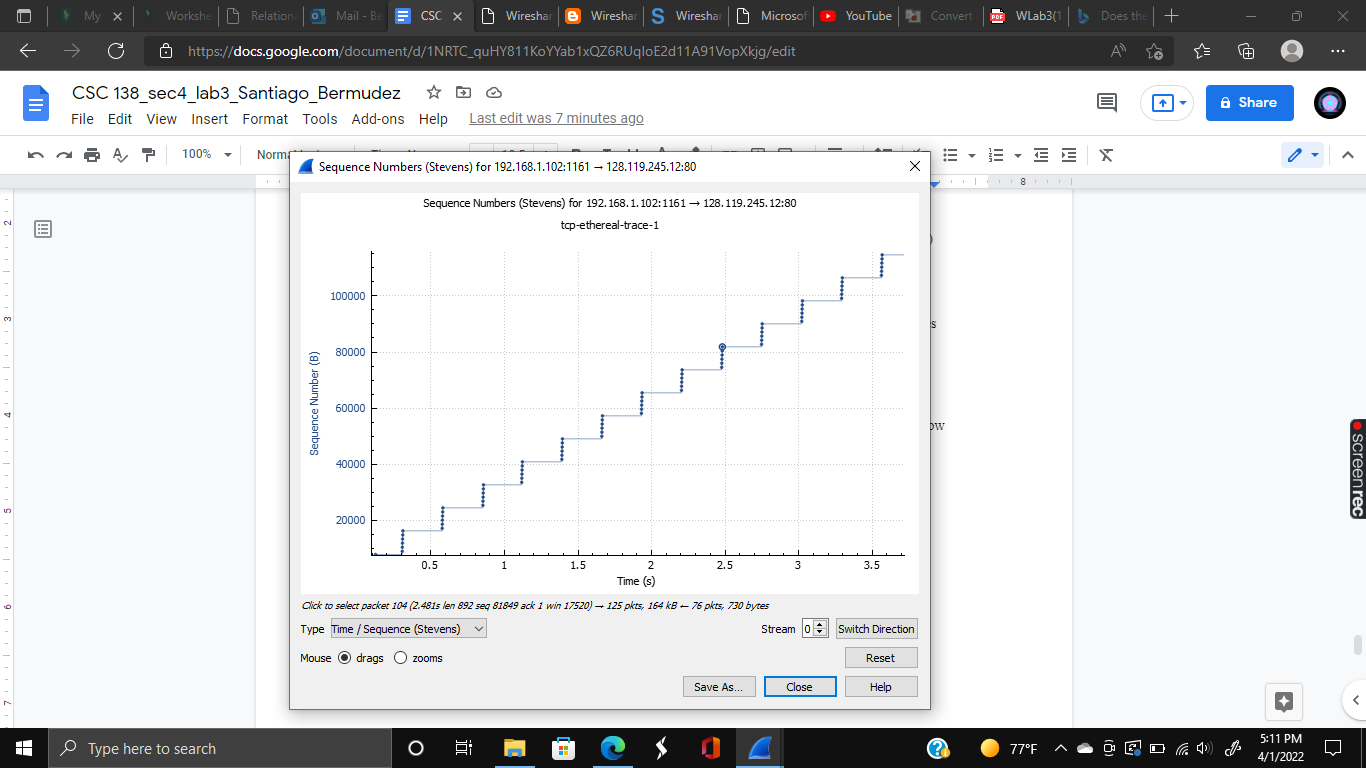
9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

The minimum amount of available buffer space advertised is 5840. The lack of receiver buffer space does not throttle the sender as the receiver window grows at a rate such that we never reach full capacity of the window.



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

No, there are no retransmitted segments in the trace file. We can tell this as in the trace for the TimeSequence-Graph (Stevens), there are no packets with the same sequence numbers being found at different times. You can also say that the graph grows steadily upward and does not dip down at any point.



11. 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).

ACK 1: Acknowledged sequence number: 566 Acknowledged data: 566

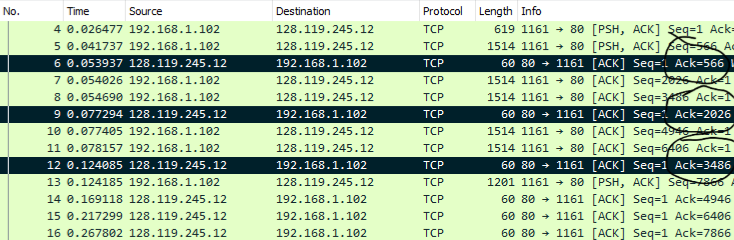
ACK 2: Acknowledged sequence number: 2026 Acknowledged data: 1460

ACK 3: Acknowledged sequence number: 3486 Acknowledged data: 1460

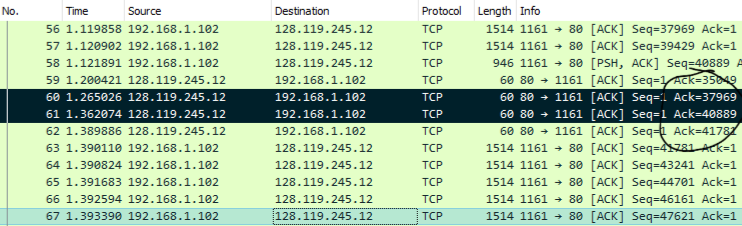
And so on…

Apart from the first ACK, the receiver typically acknowledges an amount of 1460 as you look through each ACK. There are cases where the receiver ACKs every other received segment and this is shown when more than one packet is acknowledged as one ACK(\*See example below).

Screenshot for ACKs (\*You can see the difference in ACKs circled below):



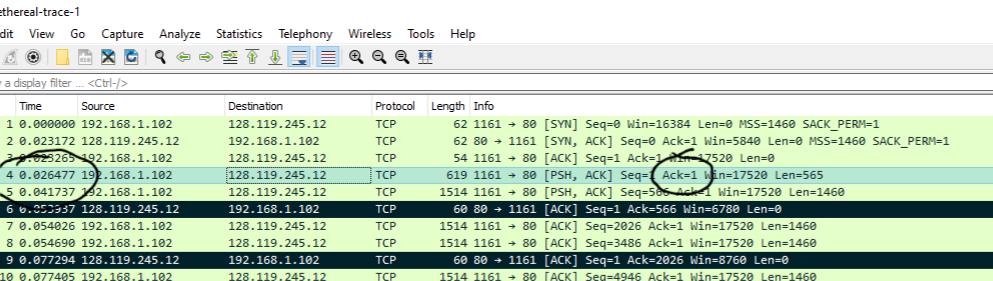
Below is an example where the receiver ACKs other received segments in that the difference is 40889 - 37969 = 2920. 2920 is double the normal amount, so two packets were acknowledged here.

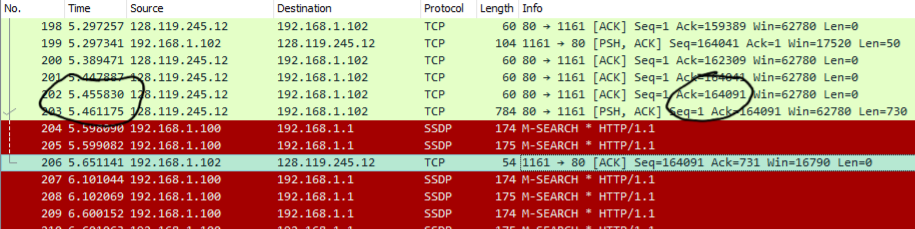


12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

The throughput is the amount of data transmitted divided by the time incurred. We can determine the amount of data transmitted by looking at the difference between the sequence number of the first TCP segment and the acknowledged sequence number of the last ACK. This gets us 164091 - 1byte = 164090 bytes.

We can determine the time by looking at the difference between the first TCP segment and the last ACK. This gets us 5.455830 - 0.026477 = 5.429353 seconds. The throughput would then be 164090bytes/5.429353 seconds = 30.222KBytes/Sec.



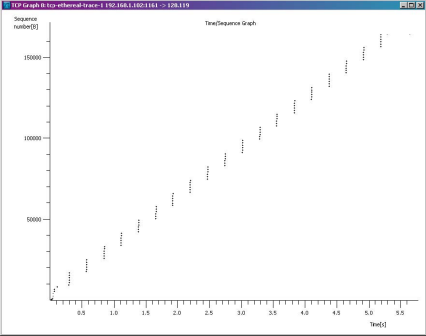


4 The TCP segments in the tcp-ethereal-trace-1 trace file are all less that 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500 byte value is the standard maximum length allowed by Ethernet. If your trace indicates a TCP length greater than 1500 bytes, 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. We recommend that if you have this inconsistency, that you perform this lab using the provided trace file.

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 (see earlier footnote ):

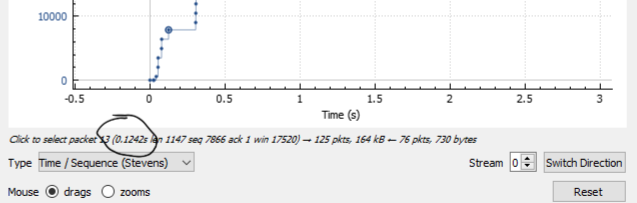


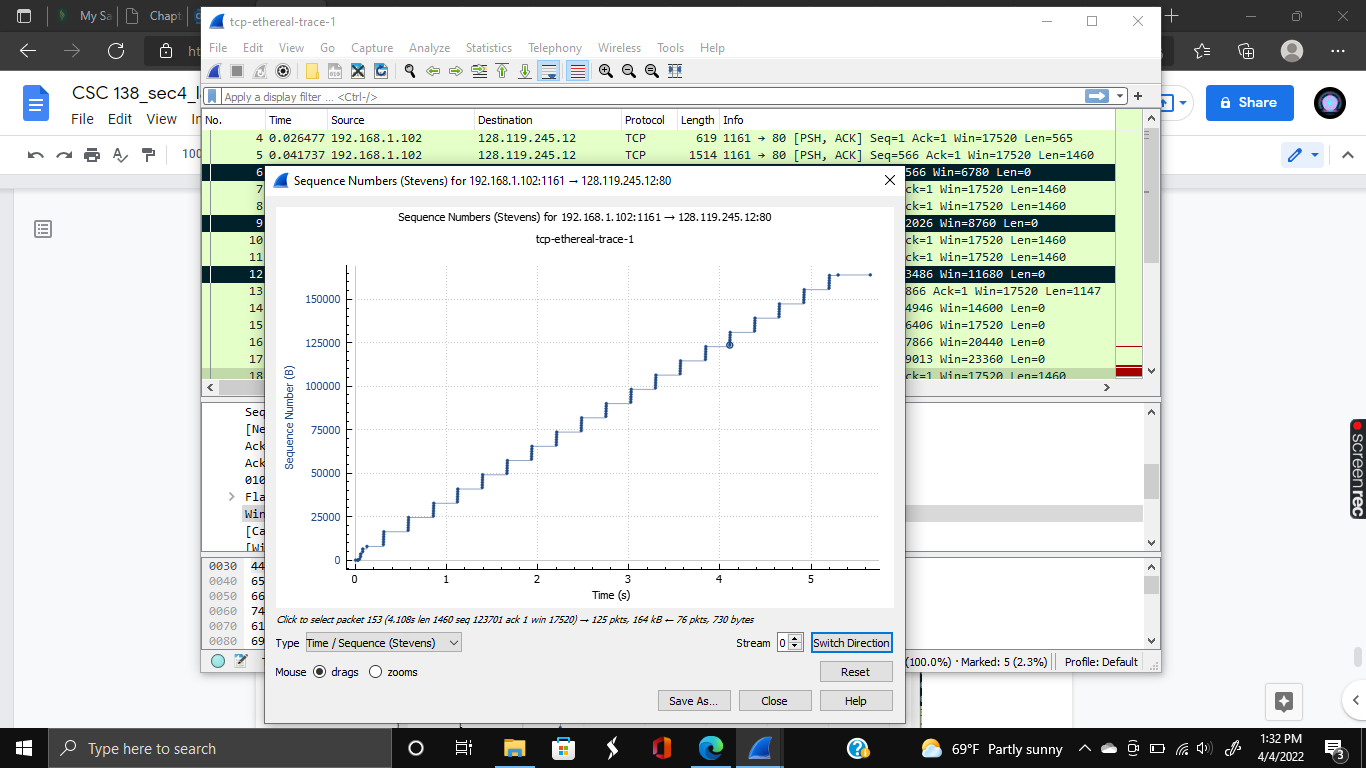
Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Answer the following questions for the TCP segments the packet trace *tcp-etherealtrace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

13. 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.

The TCP’s slowstart phase begins around 0 seconds and ends at around 0.1242 seconds according to the graph. Congestion avoidance takes over after that. It only seems to slow down towards the end, when it runs out of data to transmit. Here, we do not see the expected linear increase behavior for congestion avoidance.





14. 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 TCP’s slowstart phase begins around 0 seconds and ends at around 0.256 seconds according to the graph. Congestion avoidance does not seem to take over here.

