Class: CompSci 4C03
Student Name: Jatin Chowdhary
Student ID: 400033011
Date: March 1st, 2021

Assignment #3
Wireshark

Question 1:
What is the IP address and TCP port number used by the client computer (source) that is transferring the file to *www.cas.mcmaster.ca*? 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"

Answer 1:
The IP address and TCP port number used by the client computer are **192.168.0.94** and **49613**, respectively. This can be seen in *Figure 1*. The red boxes in the picture below highlight the IP address and TCP port number used by the client computer.

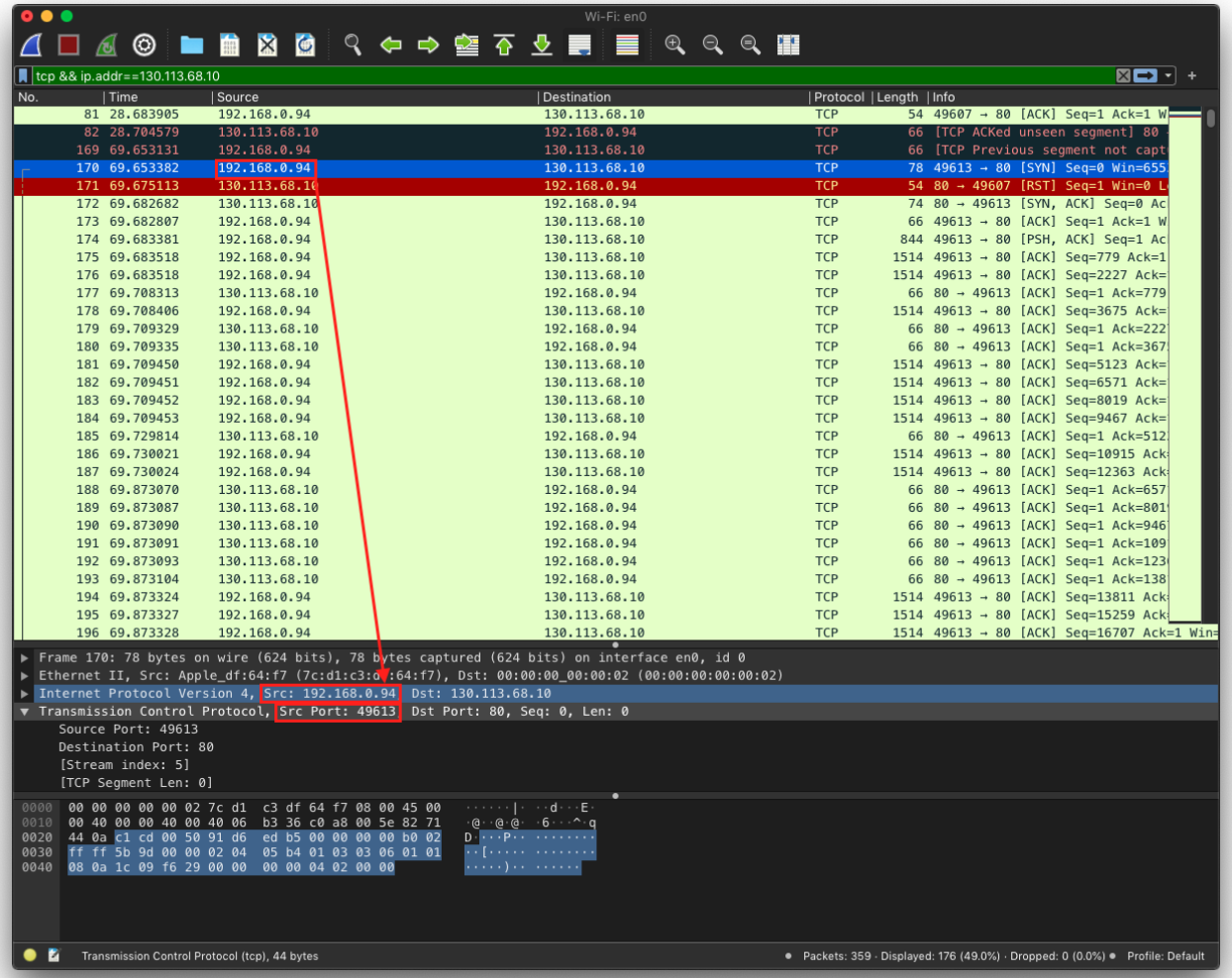

Figure 1: The red box at the top shows the IP address of the source. From here, an arrow extends downward and points to the source's IP address and TCP port number, in the packet header details window.

Question 2:
What is the IP address of *www.cas.mcmaster.ca*? On what port number is it sending and receiving TCP segments for this connection?

Answer 2:
The IP address of *www.cas.mcmaster.ca* is **130.113.68.10**. It is using the port number **80** to send and receive TCP segments for this connection. The red boxes in the figure below highlight the IP address and port number of the destination host.

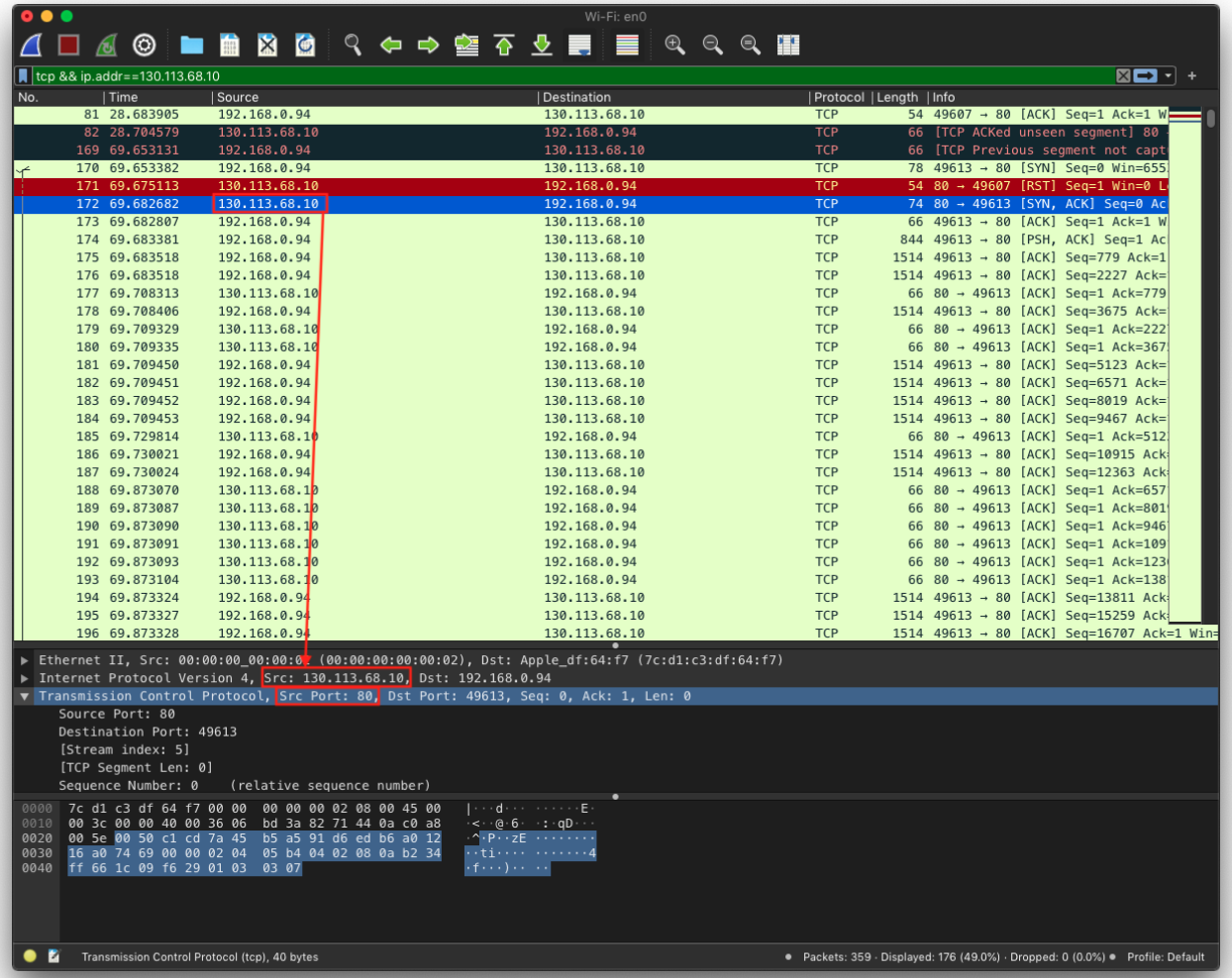

Figure 2: The red box at the top in the packet-listing window circles an incoming packet from the destination IP address. An arrow extends to the packet-header window, and two red boxes circle the IP address and TCP port number of the destination host.

Question 3:
What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and *www.cas.mcmaster.ca*? What is it in the segment that identifies the segment as a SYN segment?

Answer 3:
The sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and *www.cas.mcmaster.ca* is 0. It is 0 because it is used to initiate the connection. The segment can be identified as a SYN segment by the flags. The *syn* flag indicates that this TCP segment is a SYN segment. The red boxes in the figure below highlight the *syn* flags in the TCP segment.

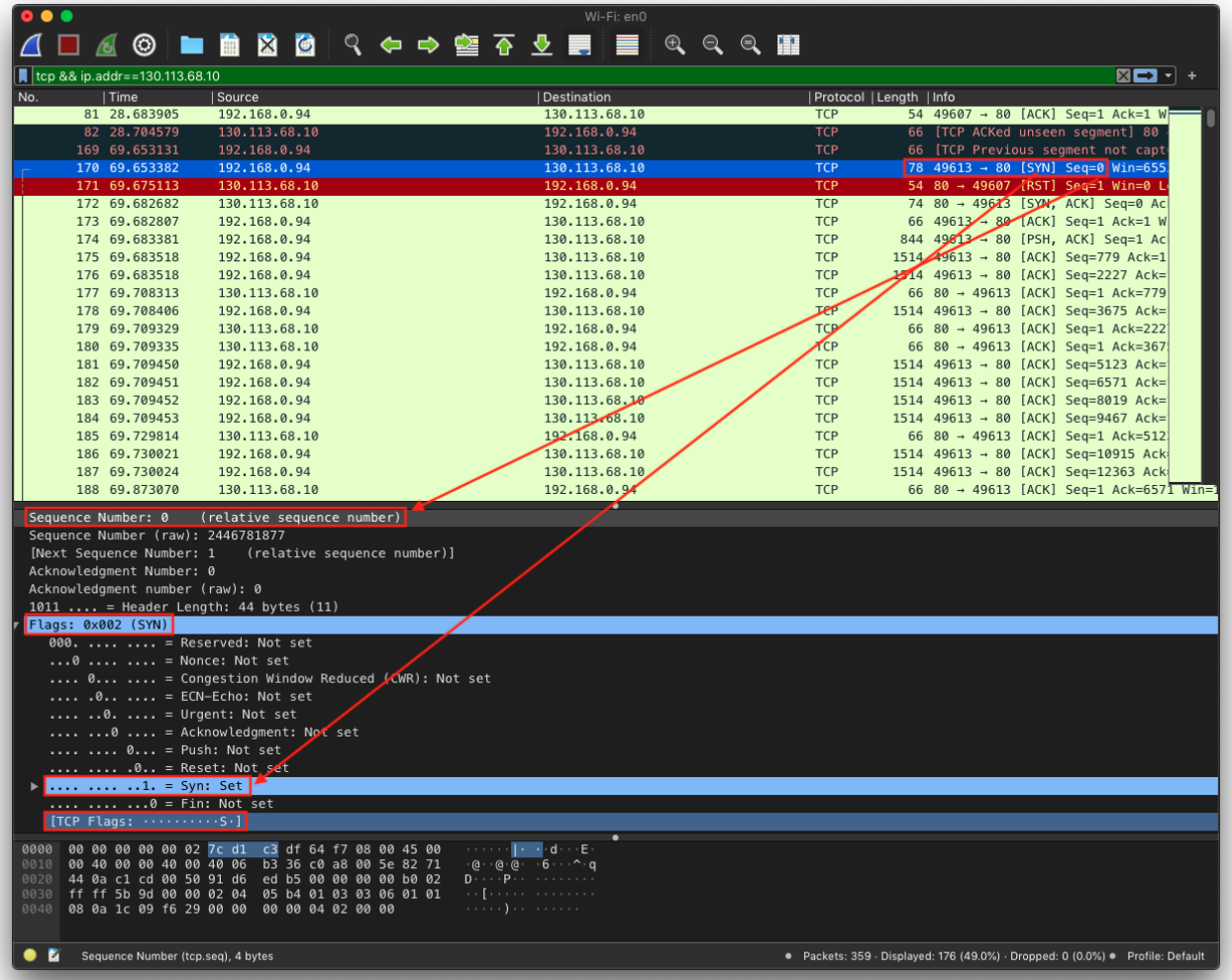

Figure 3: The red boxes in the packet-listing window circles the TCP SYN segment. Two arrows extend to the packet-header window and circle the sequence number and TCP flags

Question 4:
What is the sequence number of the SYNACK segment sent by *www.cas.mcmaster.ca* to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did *www.cas.mcmaster.ca* determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

Answer 4:
The sequence number of the SYNACK segment sent by *www.cas.mcmaster.ca* in response to the client's SYN segment is 0. The value of the acknowledgement field in the SYNACK segment is 1. This number is determined by the server (*www.cas.mcmaster.ca*), and to calculate this number it increments the previous SYN segment's sequence number by 1. The sequence number from the previous segment becomes the acknowledgement number of the corresponding packet. This is cumulative acknowledgement. The red boxes in the figure below highlight the sequence number, acknowledgement number, and flags of the SYNACK segment.

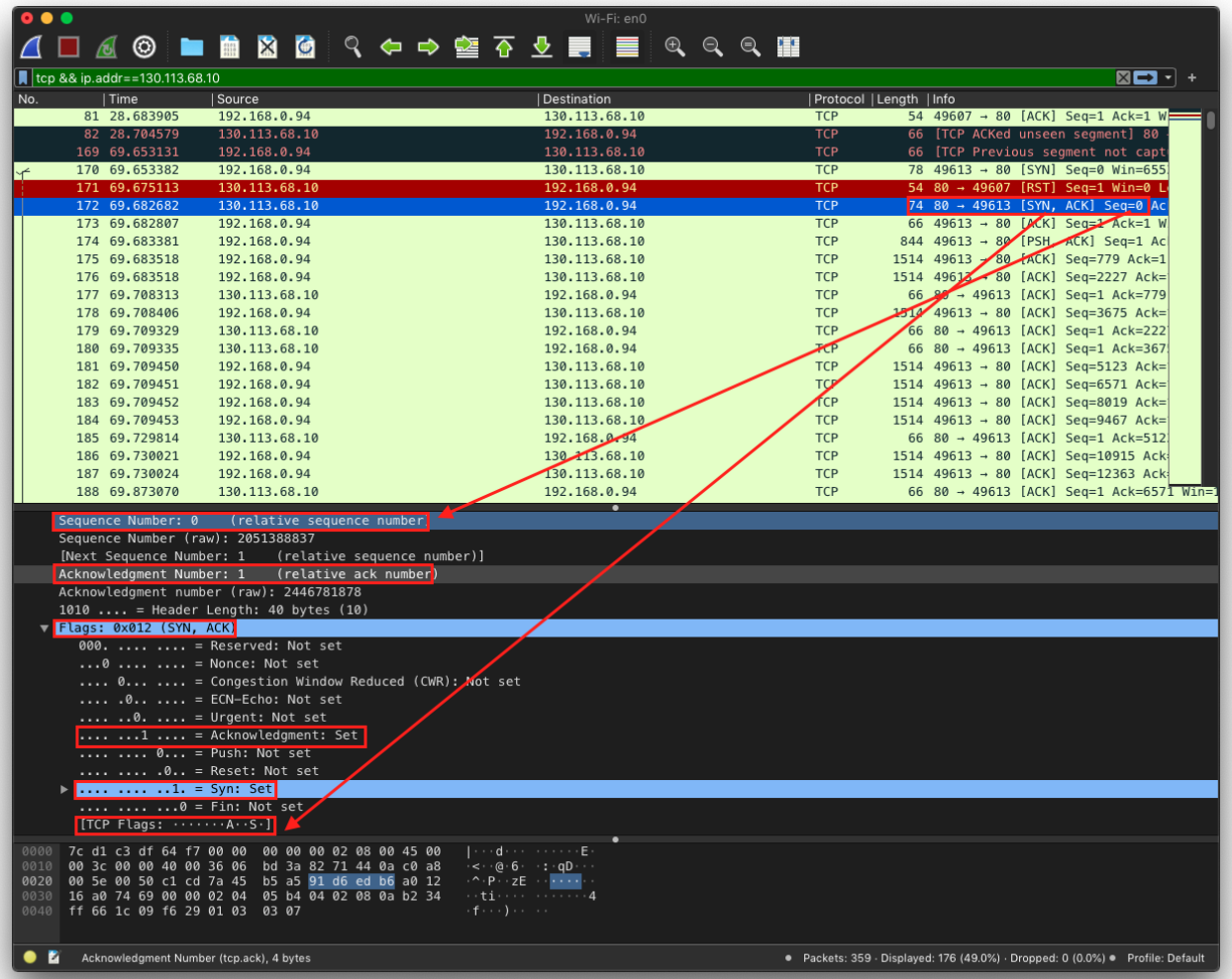

Figure 4: The red box in the packet-listing window highlights the SYNACK TCP segment. Two arrows extend from here and point to the sequence number, and TCP flags of the corresponding segment. Other information, such as acknowledgement number, is also highlighted in red boxes.

Question 5:
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.

Answer 5:
The sequence number of the TCP segment containing the HTTP POST command is 1. This is demonstrated in the figure below; the red boxes indicate their respective fields.

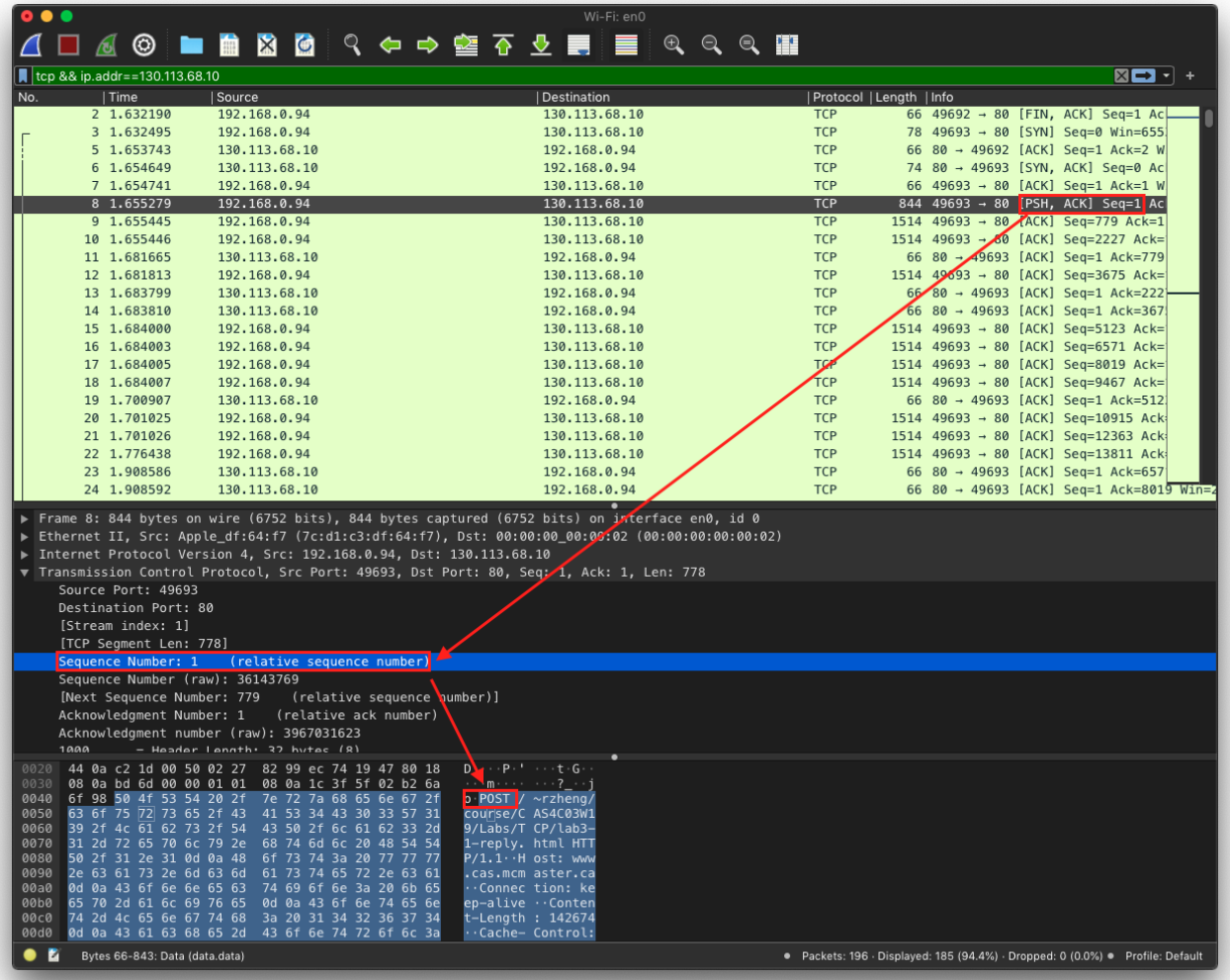

Figure 5: The red box in the packet-listing window highlights the TCP segment containing the HTTP POST command. A line extends from here into the packet-header window, pointing to a red box that circles the sequence number. Another arrow extends from here into the packet-contents window, proving that this TCP segment does in fact contain the HTTP POST command.

Question 6:
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.

Answer 6:
The first six segments sent from the source host to the destination host, including the HTTP POST segment, are packets: 8, 9, 10, 12, 15, 16. These packets contain data regarding the *alice.txt* file. The sequence numbers corresponding to these first six segments in the TCP connection are:
TCP Segment #1 (08) Sequence Number = 1
TCP Segment #2 (09) Sequence Number = 779
TCP Segment #3 (10) Sequence Number = 2227
TCP Segment #4 (12) Sequence Number = 3675
TCP Segment #5 (15) Sequence Number = 5123
TCP Segment #6 (16) Sequence Number = 6571

This is illustrated in figure 6, below.

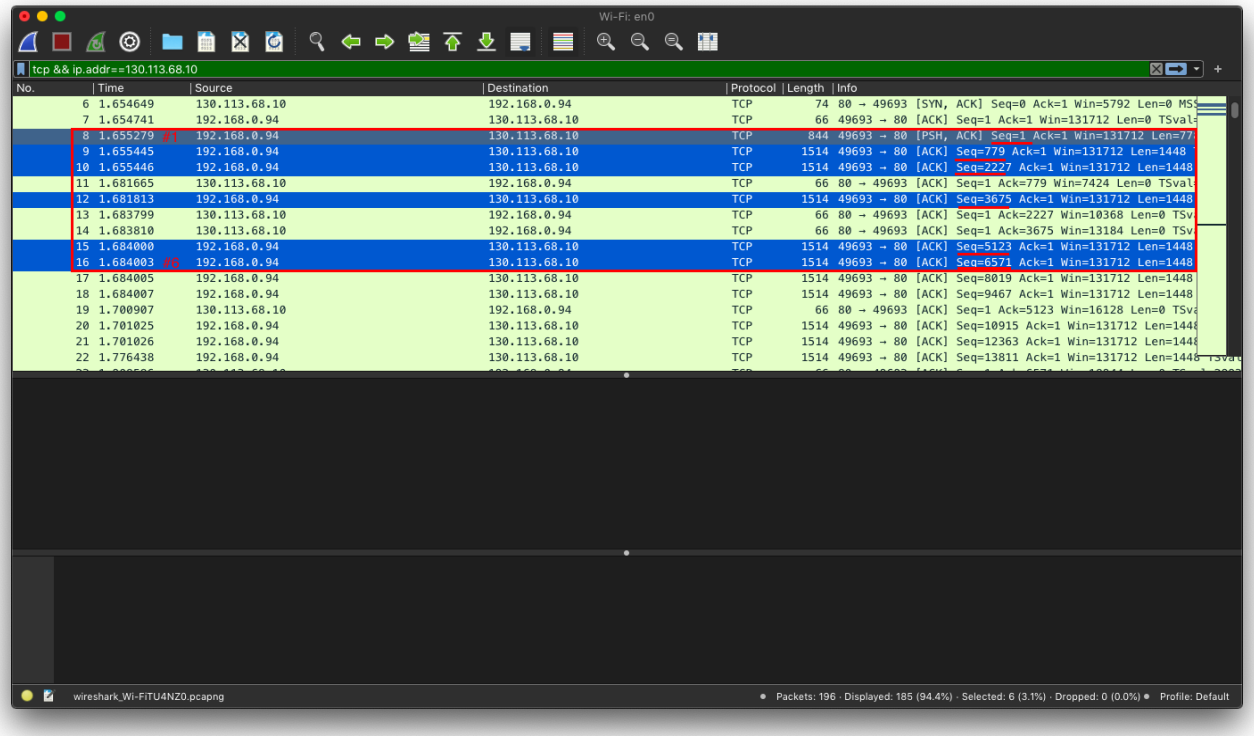

Figure 6: The packets highlighted in blue represent the first six TCP segments sent from source host to destination host, including the TCP segment containing the HTTP POST command. The red box encases these six TCP segments. On the right side, the sequence numbers are underlined in red for the first six TCP segments. On the left, the first and last segment are labelled.

Assuming that the TCP segment containing HTTP POST is the first segment, in reference to subsequent segments, each subsequent segment was sent at the following time:
TCP Segment #1 (08) Sent Time = 1.655279
TCP Segment #2 (09) Sent Time = 1.655445
TCP Segment #3 (10) Sent Time = 1.655446
TCP Segment #4 (12) Sent Time = 1.681813
TCP Segment #5 (15) Sent Time = 1.684000
TCP Segment #6 (16) Sent Time = 1.684003
Note: These times are according to Wireshark, and correspond with the time that Wireshark was able to capture the packets from the moment it was allowed to start capturing packets. In other words, these values are relative to Wireshark. However, subtracting the n^{th} packet from the 0^{th} packet will give you the sent times in reference to the first TCP segment. Also, the times listed above are in seconds.

This information is shown in figure 7, below.

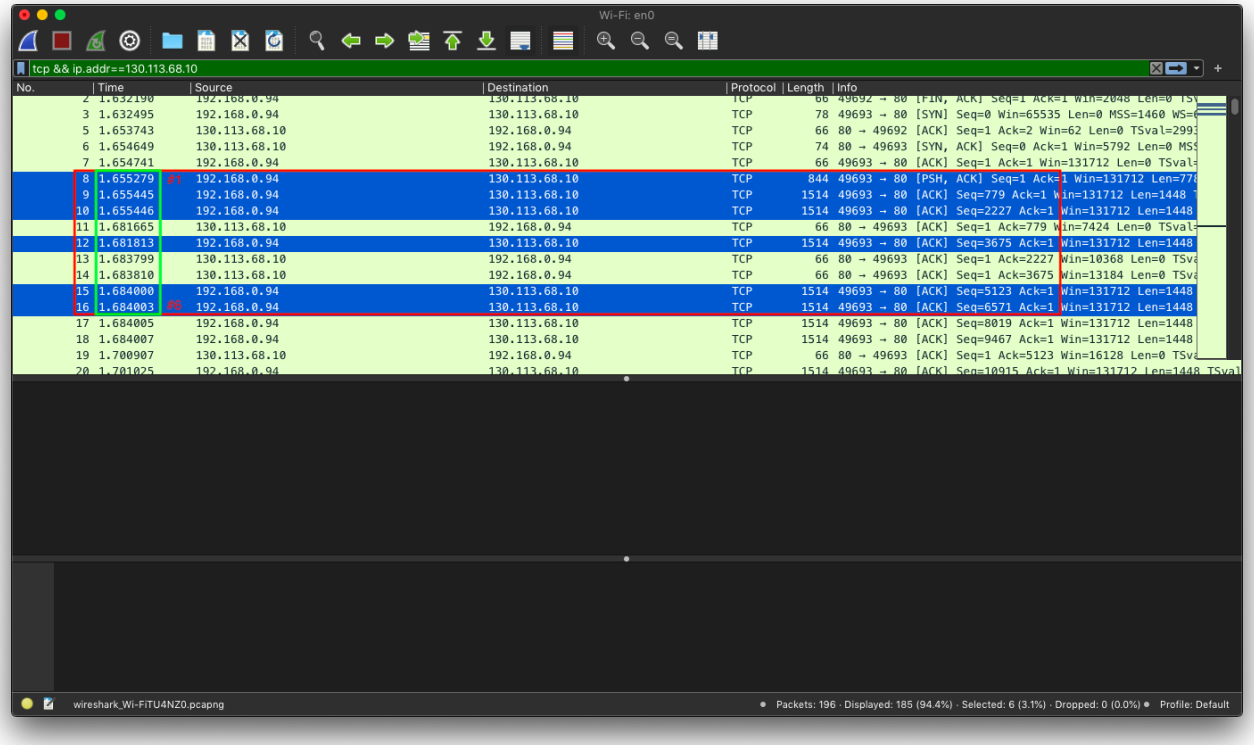

Figure 7: The packets highlighted in blue correspond to the first 6 TCP segments starting from the TCP segment containing the HTTP POST command. The red box encases these 6 packets. On the right side, the green box encloses the time that each packet was sent. On the right side, a label marks the first and last TCP segment.

The acknowledgement for each segment was received at:
TCP Packet #11 Acknowledgement For Packet #08 Received Time = 1.681665
TCP Packet #13 Acknowledgement For Packet #09 Received Time = 1.683799
TCP Packet #14 Acknowledgement For Packet #10 Received Time = 1.683810
TCP Packet #19 Acknowledgement For Packet #12 Received Time = 1.700907
TCP Packet #23 Acknowledgement For Packet #15 Received Time = 1.908586
TCP Packet #24 Acknowledgement For Packet #16 Received Time = 1.908592
Note: These times are in seconds, and are relative values. These times correspond to when Wireshark received the packet acknowledgements. Subtracting the n^{th} time from the 0^{th} time yields the time each acknowledgement took.

This information is shown in figure 8, below

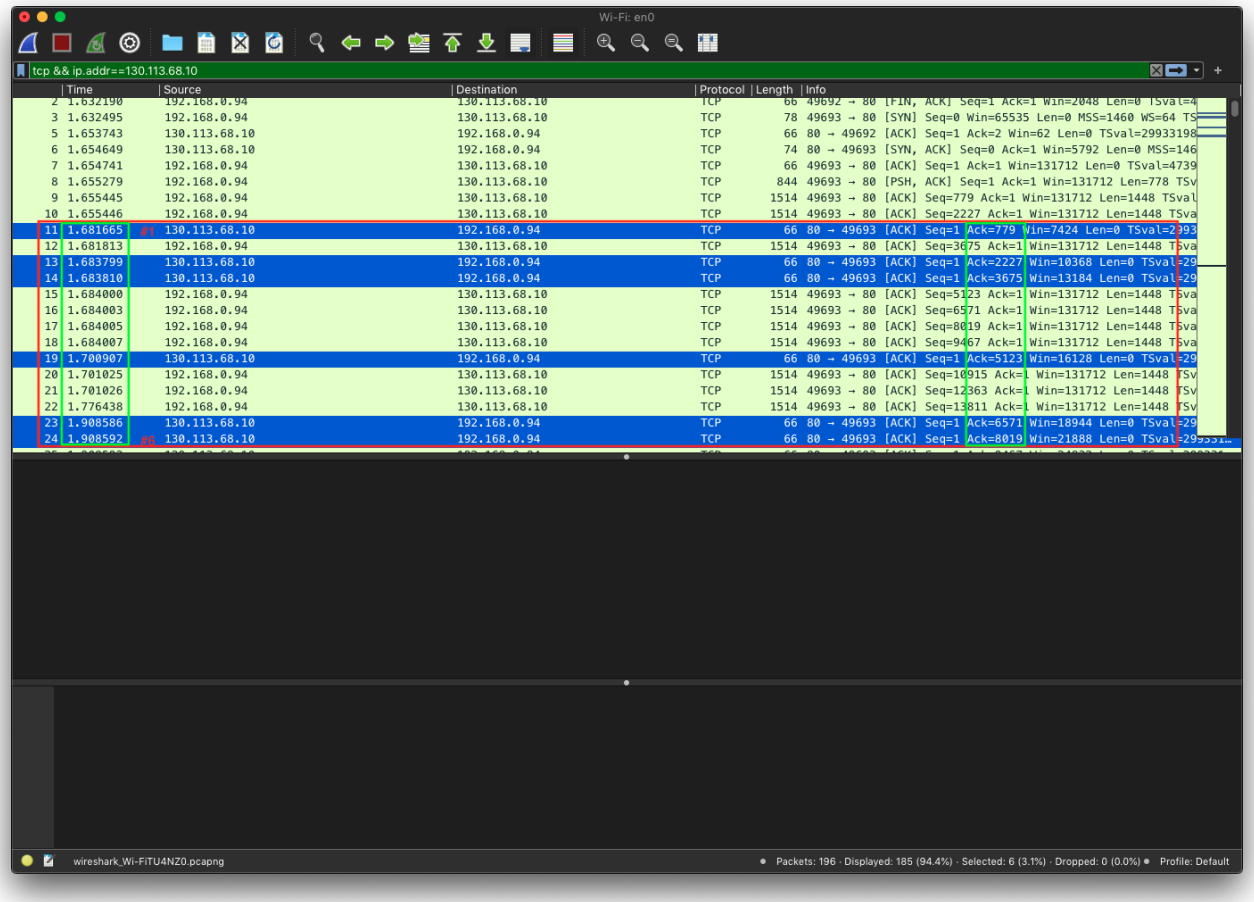

Figure 8: The packets highlighted in blue are acknowledgement packets that correspond to the sent TCP packets from figure 7. The received time is on the left in a green box, and the acknowledgement number is on the right in a green box. All of this information is in the packet-listing window.

Combining the numbers from the previous figures, figure 7 and 8, the RTT for each segment is:
TCP RTT From Packet #08 To Packet #11 = 1.681665 – 1.655279 = 0.026386
TCP RTT From Packet #09 To Packet #13 = 1.683799 – 1.655445 = 0.028354
TCP RTT From Packet #10 To Packet #14 = 1.683810 – 1.655446 = 0.028364
TCP RTT From Packet #12 To Packet #19 = 1.700907 – 1.681813 = 0.019094
TCP RTT From Packet #15 To Packet #23 = 1.908586 – 1.684000 = 0.224586
TCP RTT From Packet #16 To Packet #24 = 1.908592 – 1.684003 = 0.224589
Note: The calculated and reported values are in seconds.

On page 281 of the textbook, the EstimatedRTT equation is:
EstimatedRTT = (0.875 * EstimatedRTT) + (0.125 * SampleRTT)
This equation will be used to calculate the subsequent values. However, the first EstimatedRTT is assumed to be the measured RTT, which is the RTT from packet #08 to packet #11.

The EstimatedRTT value after the receipt of each ACK is:

TCP EstimatedRTT From Packet #08 To Packet #11
= **0.026386**
TCP EstimatedRTT From Packet #09 To Packet #13
= (0.875 × 0.026386) + (0.125 × 0.028354)
= **0.026632**
TCP EstimatedRTT From Packet #10 To Packet #14
= (0.875 × 0.026632) + (0.125 × 0.028364)
= **0.026848**5
TCP EstimatedRTT From Packet #12 To Packet #19
= (0.875 × 0.0268485) + (0.125 × 0.019094)
= **0.025879**1875
TCP EstimatedRTT From Packet #15 To Packet #23
= (0.875 × 0.0258791875) + (0.125 × 0.224586)
= **0.050717**5390625
TCP EstimatedRTT From Packet #16 To Packet #24
= (0.875×0.0507175390625) + (0.125×0.224589)
= **0.072451**4716796875

Note: Only the first 6 digits after the decimal point are relevant. This is because our initial values are good for 6 figures, thus our calculated values are also only good for 6 figures. The first 6 figures of each EstimatedRTT are bolded.

Question 7:
What is the length of each of the first six TCP segments?

Answer 7:
Assuming the first TCP segments starts from the TCP segment containing the HTTP POST command, as done in the previous question, the length of the first 6 TCP segments are:
TCP Segment #1 Length = 778
TCP Segment #2 Length = 1448
TCP Segment #3 Length = 1448
TCP Segment #4 Length = 1448
TCP Segment #5 Length = 1448
TCP Segment #6 Length = 1448

This is shown in the figure below; the lengths of the first 6 TCP segments are circled in a green box. The six TCP segments are highlighted in blue and circled in a red box.

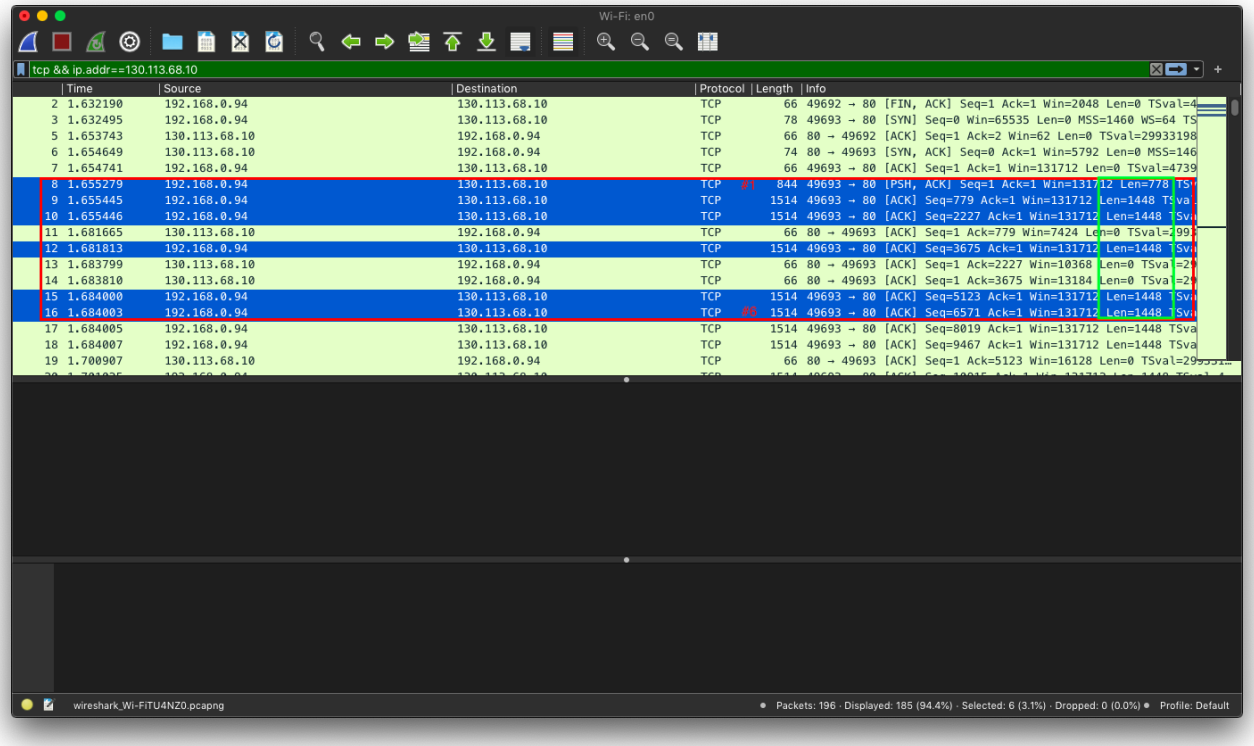

Figure 9: The first six TCP segments are highlighted in blue, and encased in a red box. The sequence numbers for these packets are circled in a green box on the right side. The first and last packet are labelled in the centre of the window. All of this information is in the packet-listing window.

Question 8:
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?

Answer 8:
The minimum amount of available buffer space advertised at the received for the entire trace is 2058. It has a scaling factor of 64. The calculated window size is 131712. This is shown in figure 10. The destination host does not throttle the source host for a lack of receiver buffer. I know this because I sorted the packet-listing window for destination IP and observed the *WIN* values for each received packet. The value does not change and remains 131712. This proves that the receiver does not throttle the sender. This is shown on figure 11, and 12.

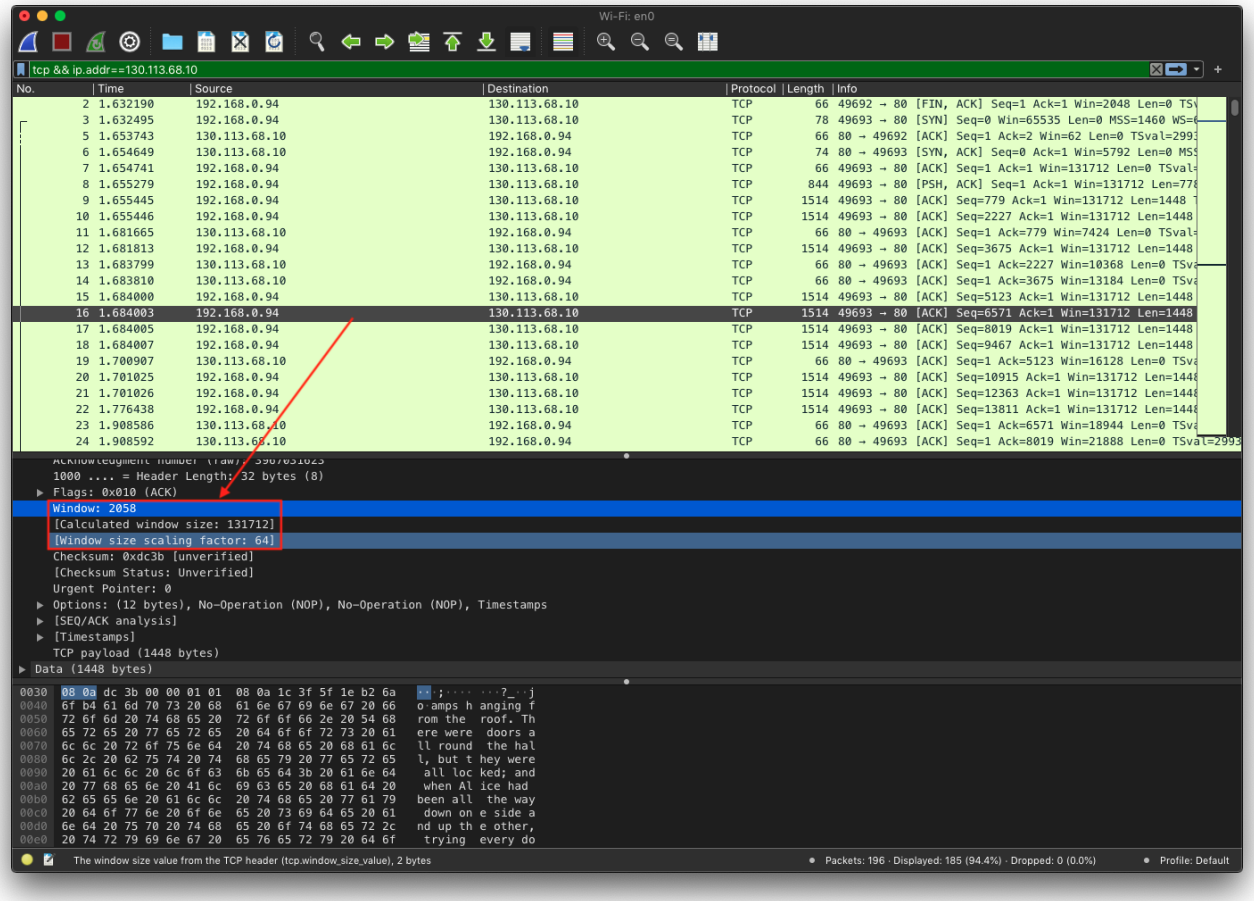

Figure 10: In the packet-listing window, a random sent TCP packet is selected and is inspected. An arrow points to the attributes it holds. The window size and scaling factor are circled in a red box in the packet-header window.

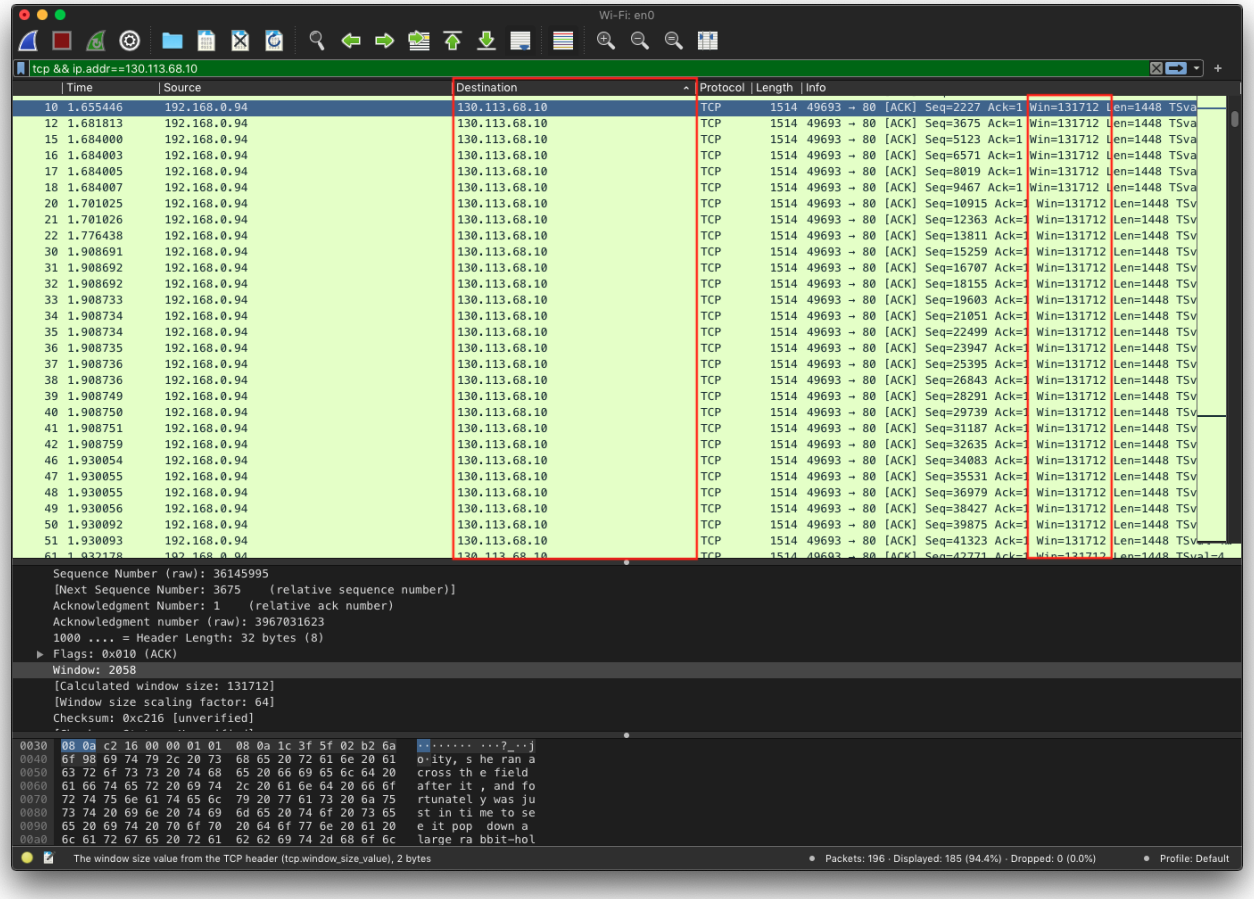

Figure 11: The packet listing window is sorted to display destination IP details at the top. On the left, a red box outlines the destination IP, and on the right, a red box outlines the window size for the receiver buffer space. The size of the buffer remains constant. Thus, the sender is not throttled.

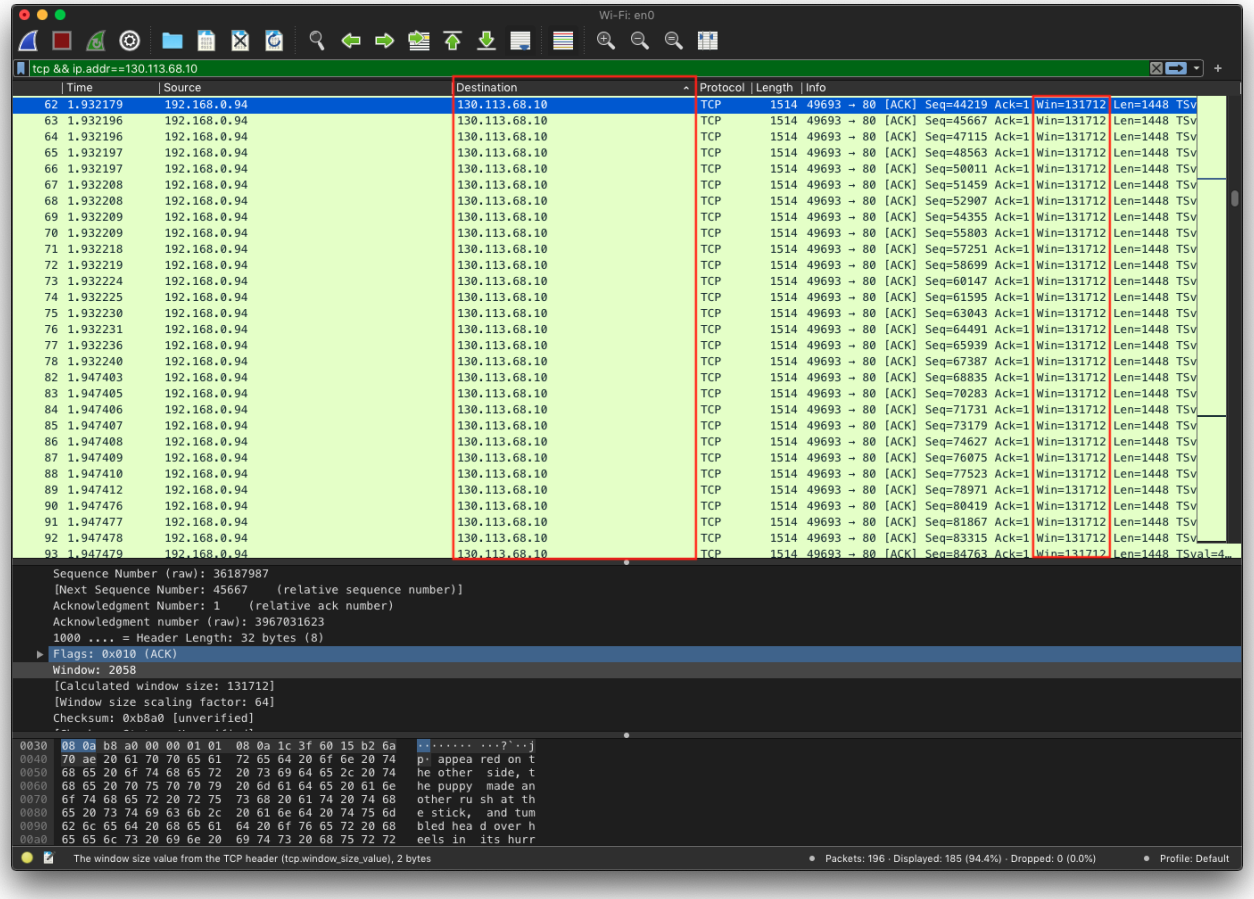

Figure 12: The packet listing window is sorted to display destination IP details at the top. On the left, a red box outlines the destination IP, and on the right, a red box outlines the window size for the receiver buffer space. The size of the buffer remains constant. Thus, the sender is not throttled.

Question 9:
Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

Answer 9:
No, there are no retransmitted segments in the trace file. To verify this, I sorted the packets by source IP. Then, I scrolled through the packets and tried to find two packets with the same sequence number. This is because if a packet was retransmitted then it will have a sequence number identical to a previous packet. However, this is not the case and all sequence numbers are increasing. Thus, there are no retransmitted segments in the trace file; the sender did not retransmit a packet to the receiver. Sample outputs are shown below in figure 13, 14, 15, and 16.

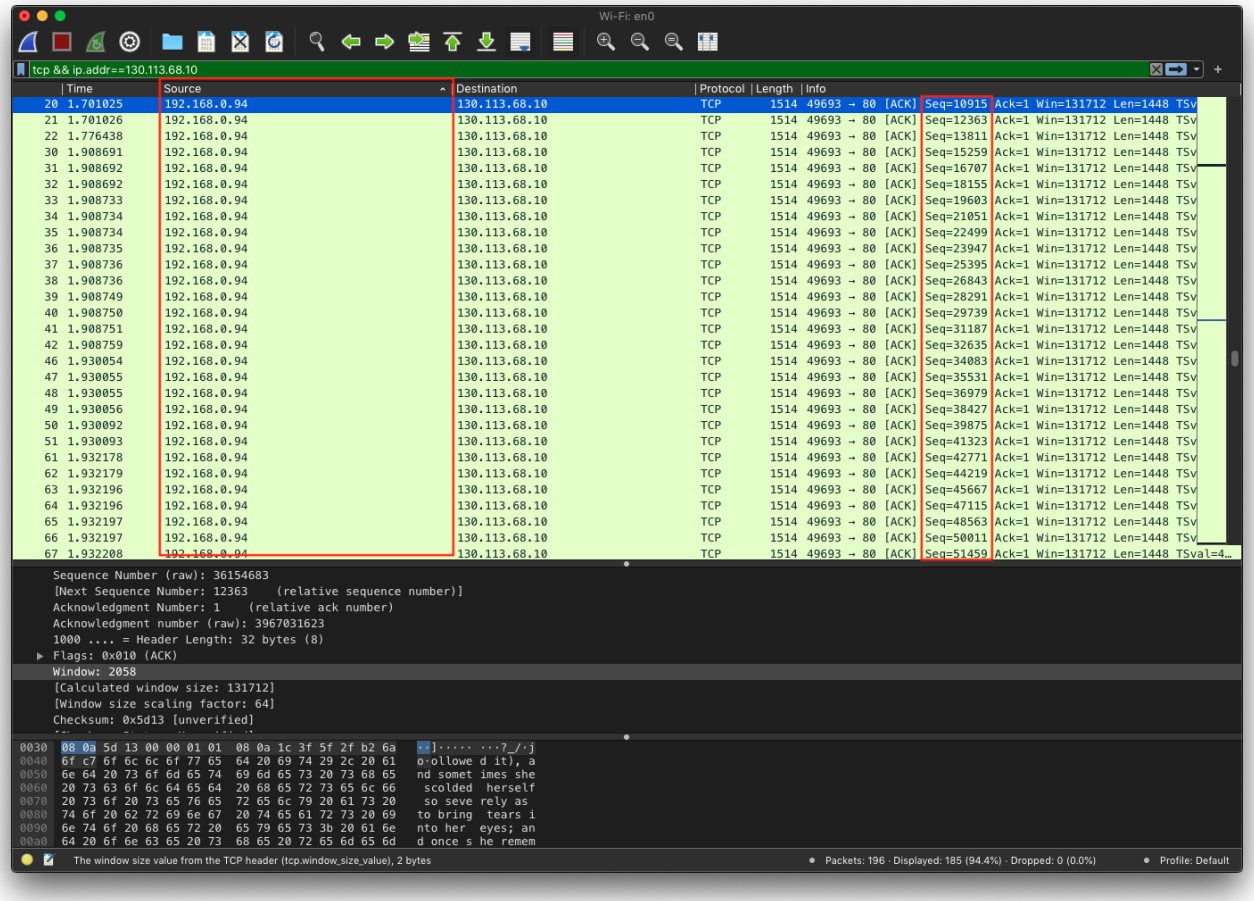

Figure 13: The packet-listing window is sorted and displays TCP segments sent from the source host to the destination host. The source host is encased in a red box on the left side of the packet-listing window, and the corresponding sequence number is in a red box on the right side of the packet-listing window. As you can see, there are no duplicate sequence numbers, thus there are no retransmitted segments.

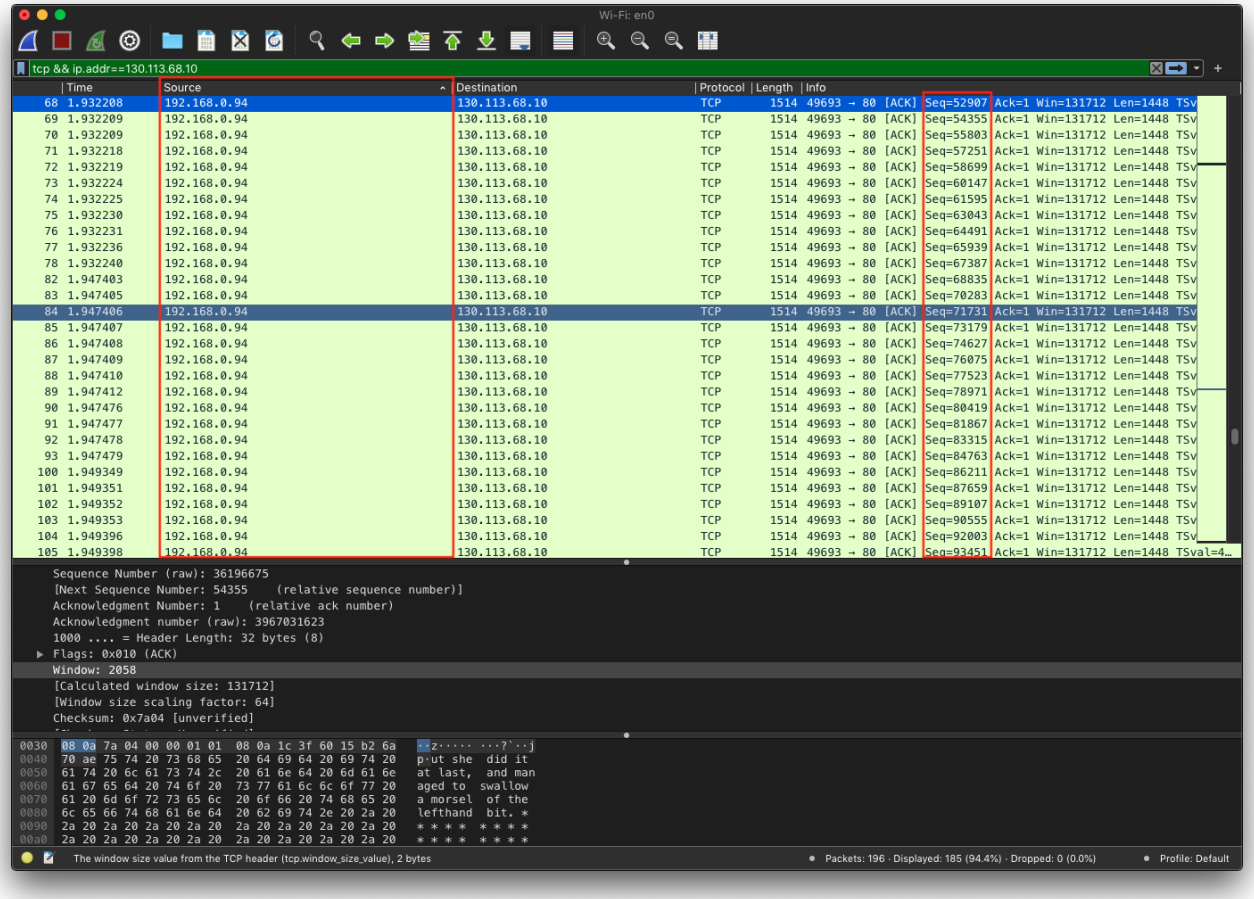

Figure 14: The packet-listing window is sorted and displays TCP segments sent from the source host to the destination host. The source host is encased in a red box on the left side of the packet-listing window, and the corresponding sequence number is in a red box on the right side of the packet-listing window. As you can see, there are no duplicate sequence numbers, thus there are no retransmitted segments.

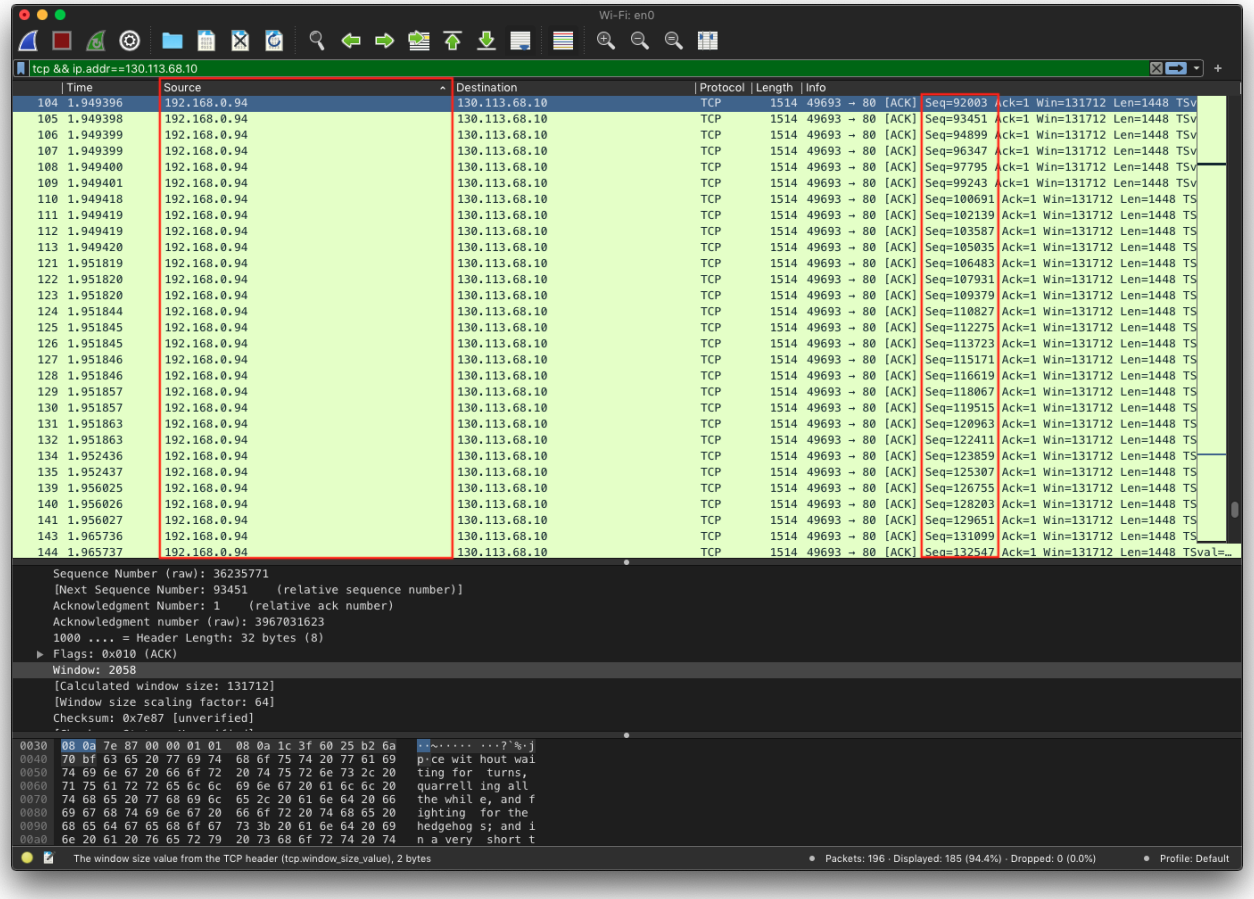

Figure 15: The packet-listing window is sorted and displays TCP segments sent from the source host to the destination host. The source host is encased in a red box on the left side of the packet-listing window, and the corresponding sequence number is in a red box on the right side of the packet-listing window. As you can see, there are no duplicate sequence numbers, thus there are no retransmitted segments.

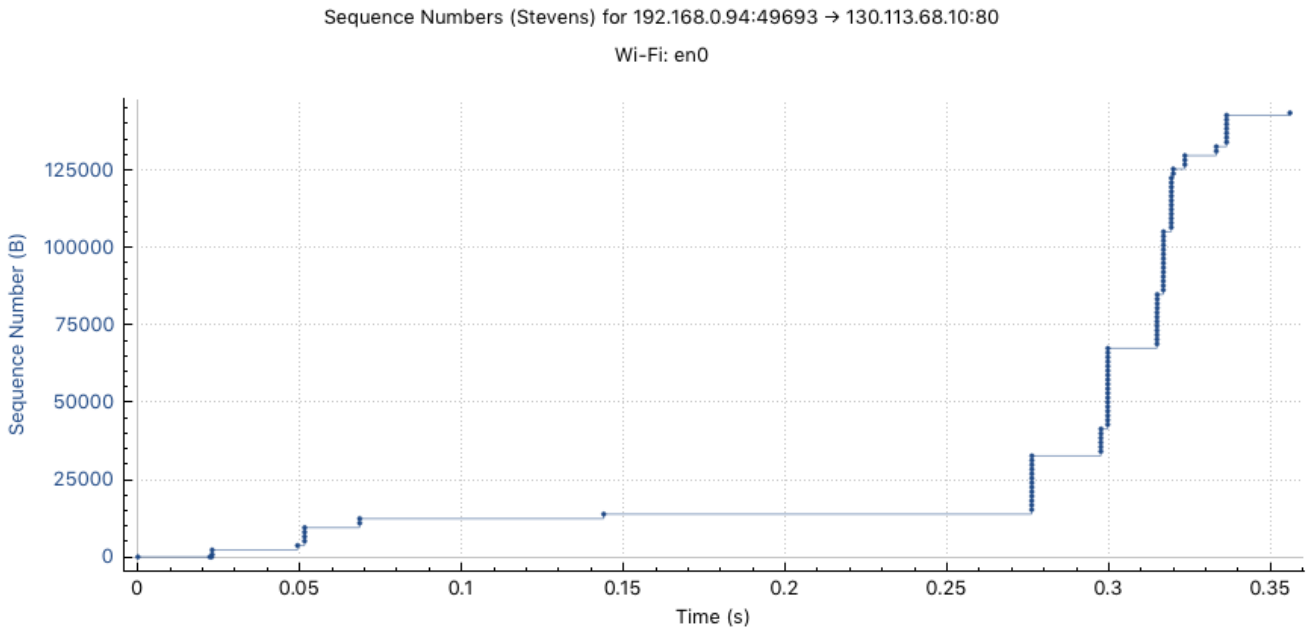

Figure 16: This sequence number graph shows that the sequence numbers for the packets being sent from sender to receiver are increasing over each period of time. There is no case where the sequence number does not increase with respect to time. Thus, identical packets are no being retransmitted; which means that there is no packet loss

Question 10:
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).

Answer 10:
Typically, the receiver acknowledges 1448 bytes per ACK. This is the length of the TCP segment. In other words, every TCP segment is being acknowledged. The receiver is not acknowledging every other received TCP segment, on average. However, close to the end of the transfer, the receiver does start to acknowledge every other segment. In the figure below, packet #167 and #170 acknowledge twice the amount of data in a single acknowledgement. Precisely, they acknowledge 2896 bytes of data in a single TCP ACK, which is exactly twice the size of one TCP packet. This is shown in the figure 17, below.

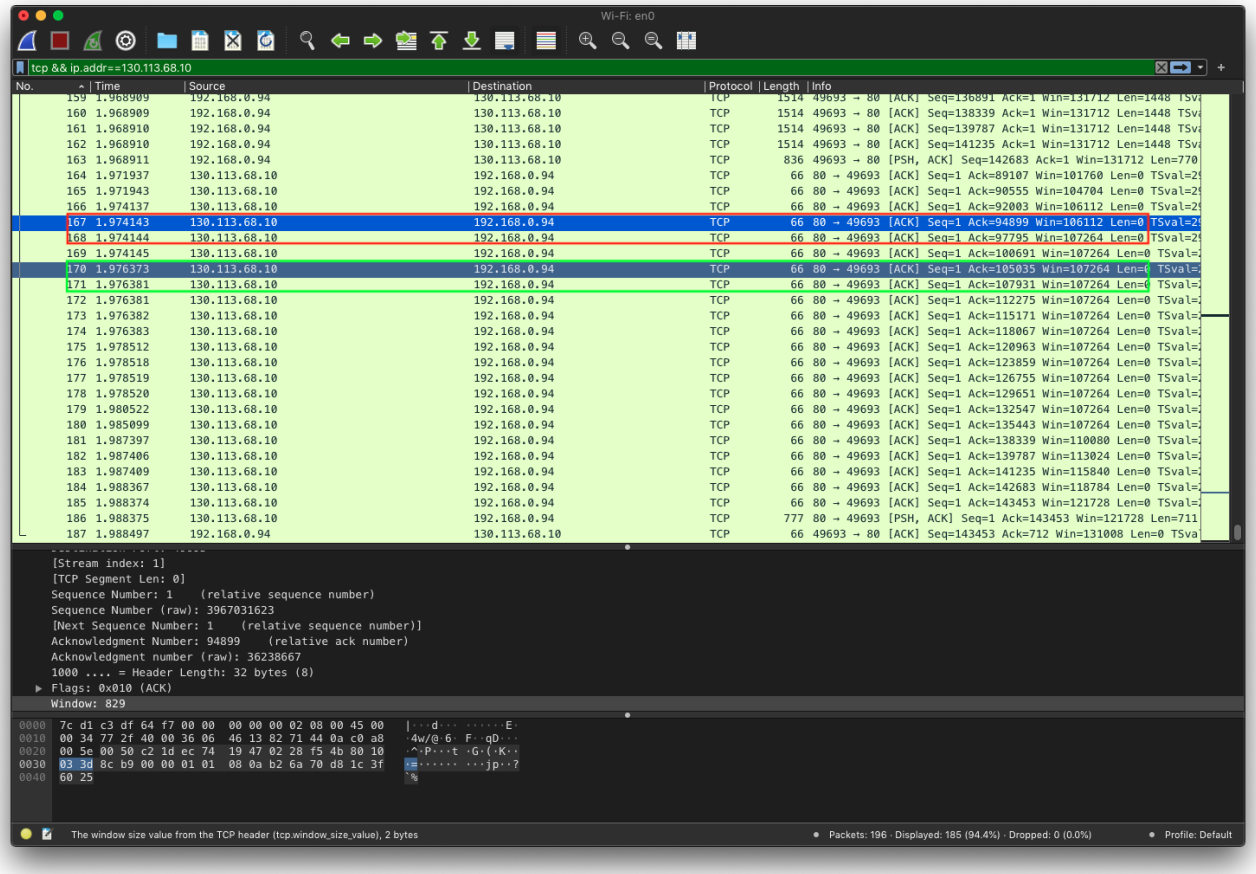

Figure 17: In the packet-listing window, packets #169 - #171 acknowledge twice the amount of bytes. This is evident by subtracting the acknowledgement number of the previous ACKS. Doing so yields a value of 2986, which is exactly twice the size of the average data ACK. This demonstrates that the receiver is acknowledging every other packet. The packets circled in a red and green box demonstrate this. All this information is in the packet-listing window and near the end of the transmission.

Question 11:
What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

Answer 11:
The throughput for the TCP connection is roughly **402,955.60 b/s** *(bytes per second)*
This value was calculated in the following manner:

The last acknowledgement of the TCP connection has an acknowledgement number of 143453. This means that 143453 bytes of information was sent from source host (sender) to destination host (receiver). The start time of the TCP connection, according to Wireshark, is 1.632495 seconds, and the time of the final acknowledgement is 1.988497 seconds. Subtracting final ACK time from start time gives us 0.356002 seconds; this represents the total time of the TCP connection/transfer. So, we have the total amount of information, in bytes, that was sent, and time it took to send that information. Dividing total time from total bytes will give us the throughput in bytes per second.

For example:
Total Bytes / Total Time
= (143453 bytes) / (0.356002 seconds)
= 402,955.6014 bytes/seconds

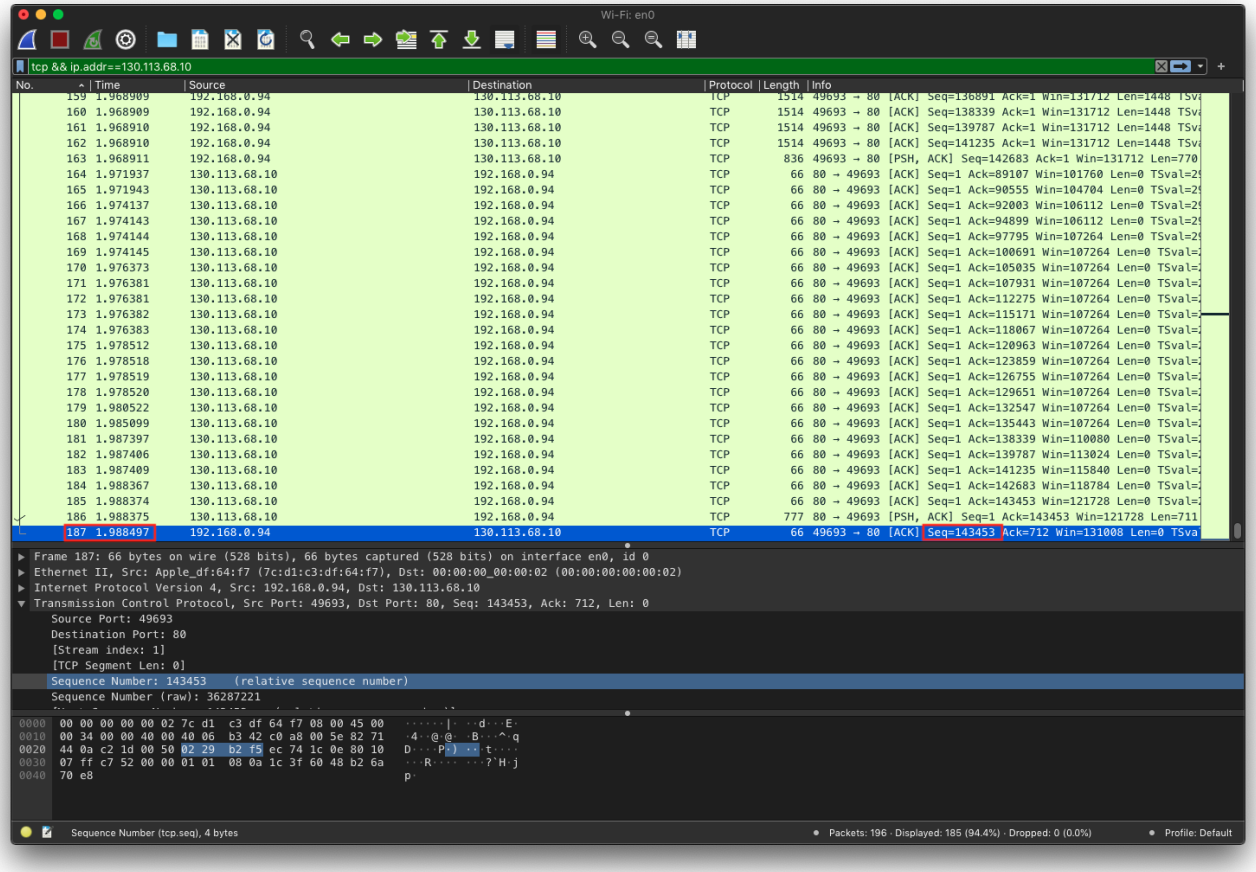

Figure 18: This is the bottom of the packet-listing window. It contains the final ACK that was sent from receiver to sender. As you can see, the final time is 1.988497 seconds, and the final sequence number is 143453. The relative packet number for the final ACK is 187. This is circled in red boxes.

Question 12:
Use the Window scaling graph plotting tool to view the growth of congestion window size for the TCP connection from the client to the *www.cas.mcmaster.ca* 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.

Answer 12:
TCP's slow-start phase begins at 0.015 seconds, and ends at around 0.055 seconds. This is because the amount of data sent stops doubling, and it even decreases a bit. Furthermore, congestion avoidance takes over at around 0.14 seconds, because the amount of data sent decreases by a multiplicative constant. This is when TCP congestion avoidance takes over. This is all illustrated in figure 19, below. The measured data differs from the idealized behavior of TCP studied in class because the Window Scaling graph does not mimic AIMD, additive increase and multiplicative decrease. This is especially present from 0.275 seconds to 0.34 seconds, when the packet data being sent rapidly increases. It does not mimic additive increase. Furthermore, the measured data does not produce a graph that resembles a wave or saw-tooth. Figure 19 resembles a stair type graph more than anything else.

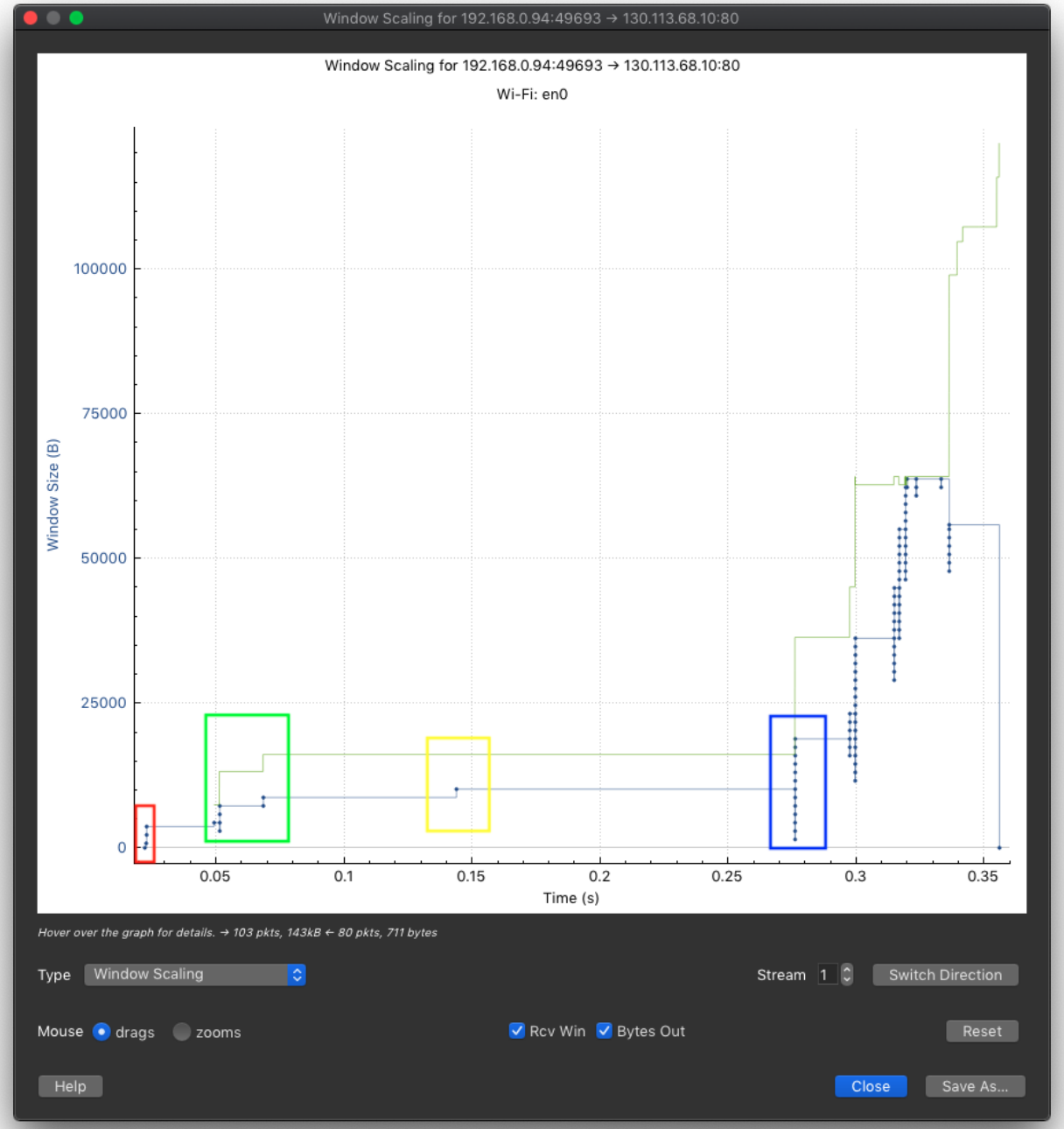

Figure 19: This is a Window Scaling graph for packets sent from sender to receiver. The red square outlines where TCP slow-start begins, and green outlines where it ends. The yellow box is where congestion avoidance takes over and the window size decreases. The blue box is the start of non-idealized TCP behaviour.