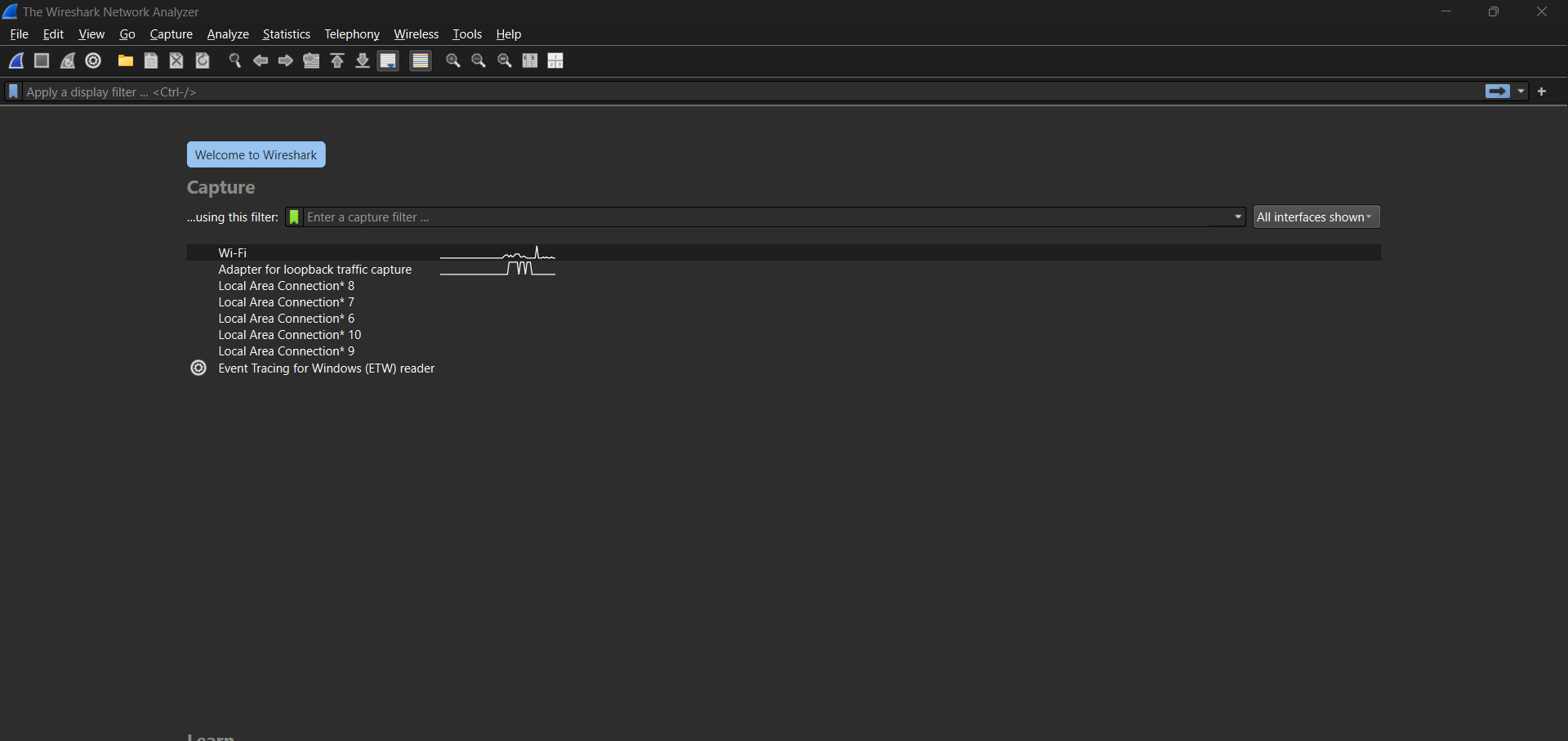
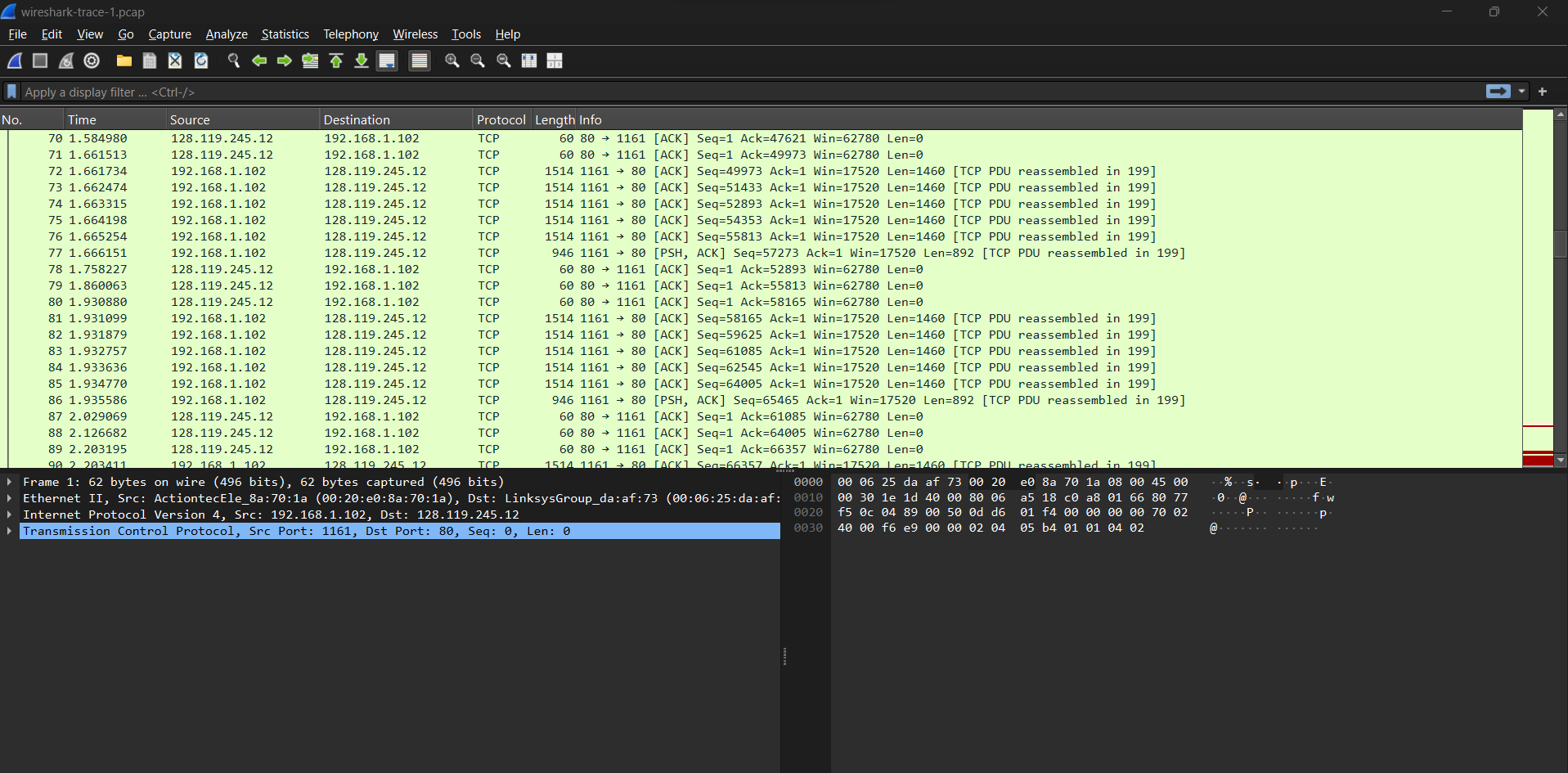
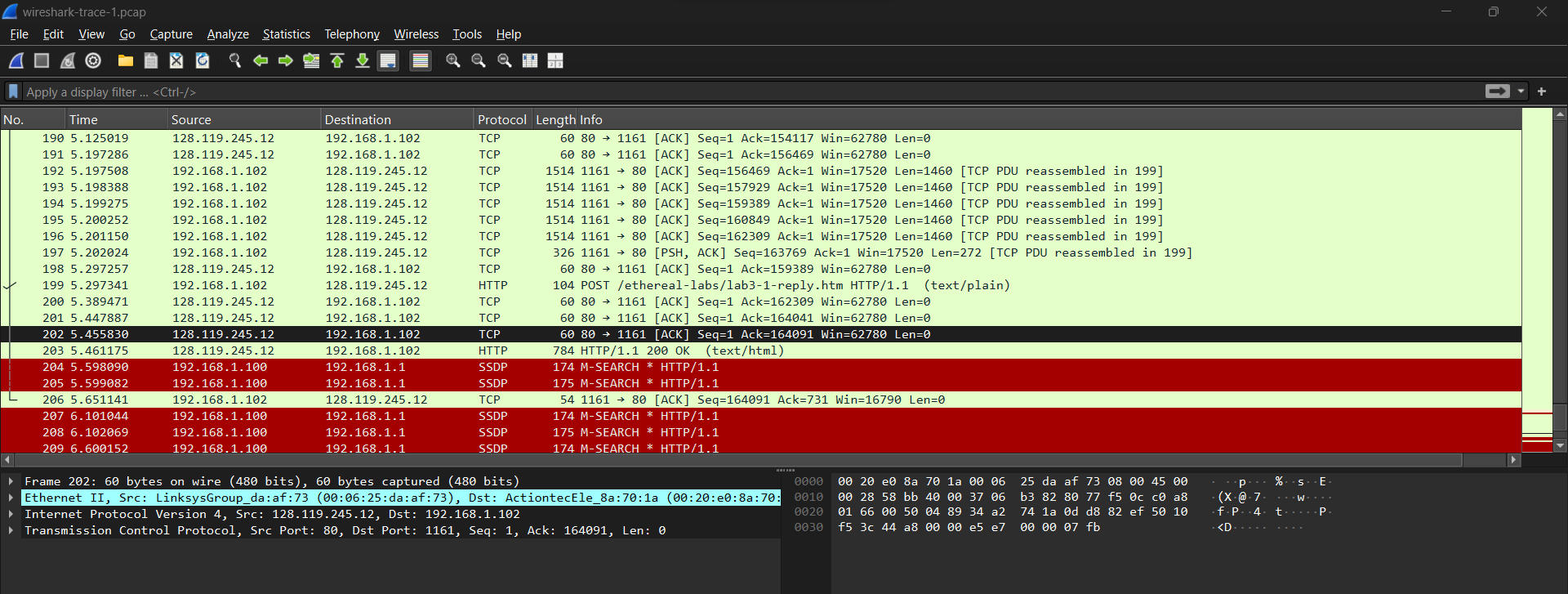
**Setting up Wireshark and Filtering TCP Packets**

 Step 1: Download and open the **Wireshark-trace-1** file in **Wireshark.**

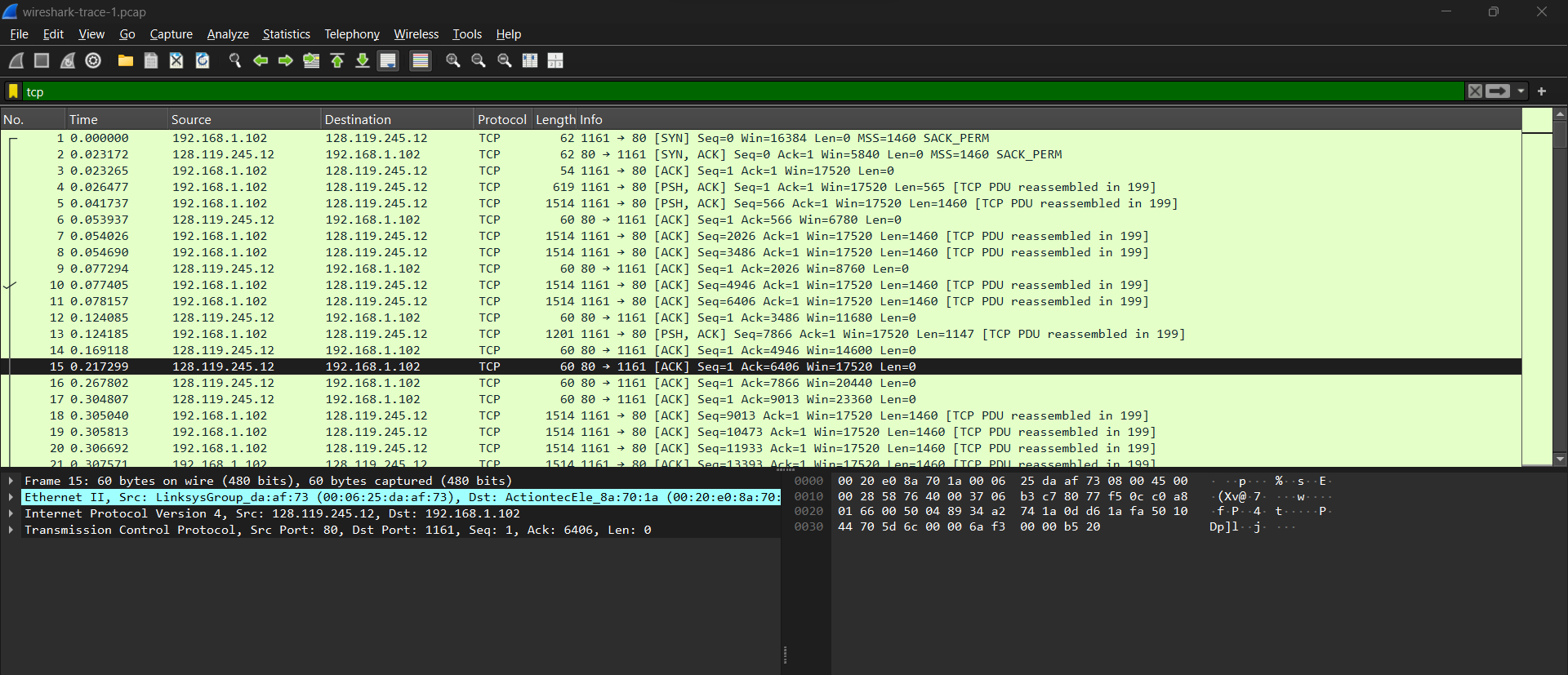
**Find** the **first** TCP three-way **handshake** **containing** **the** SYN, SYN-ACK, and ACK packets.

Open the file wireshark-trace-1 in the wireshark

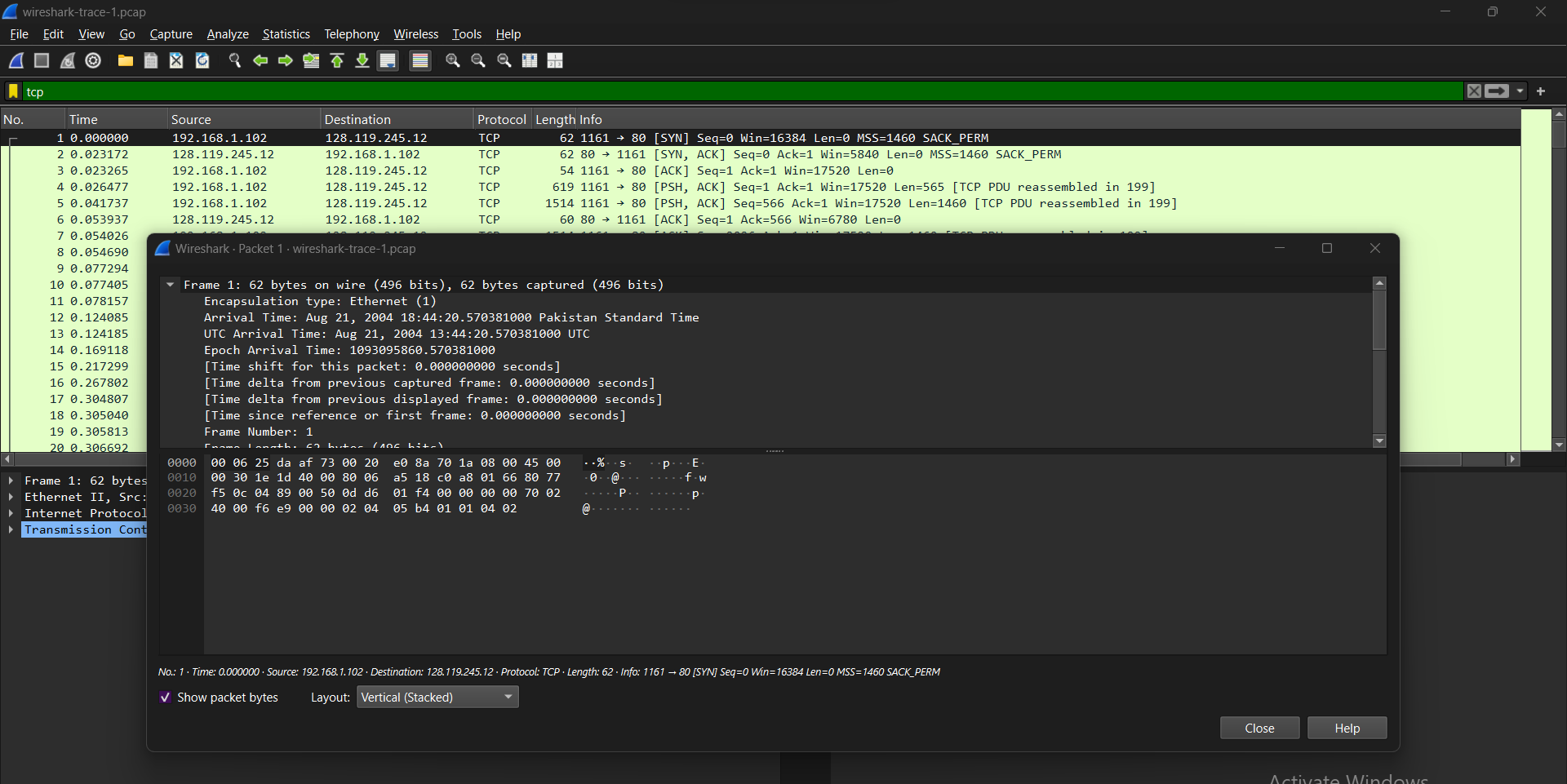




**Step 2:** Filter the packets by entering tcp into the display filter field (top of the Wireshark window) to view only TCP packets.



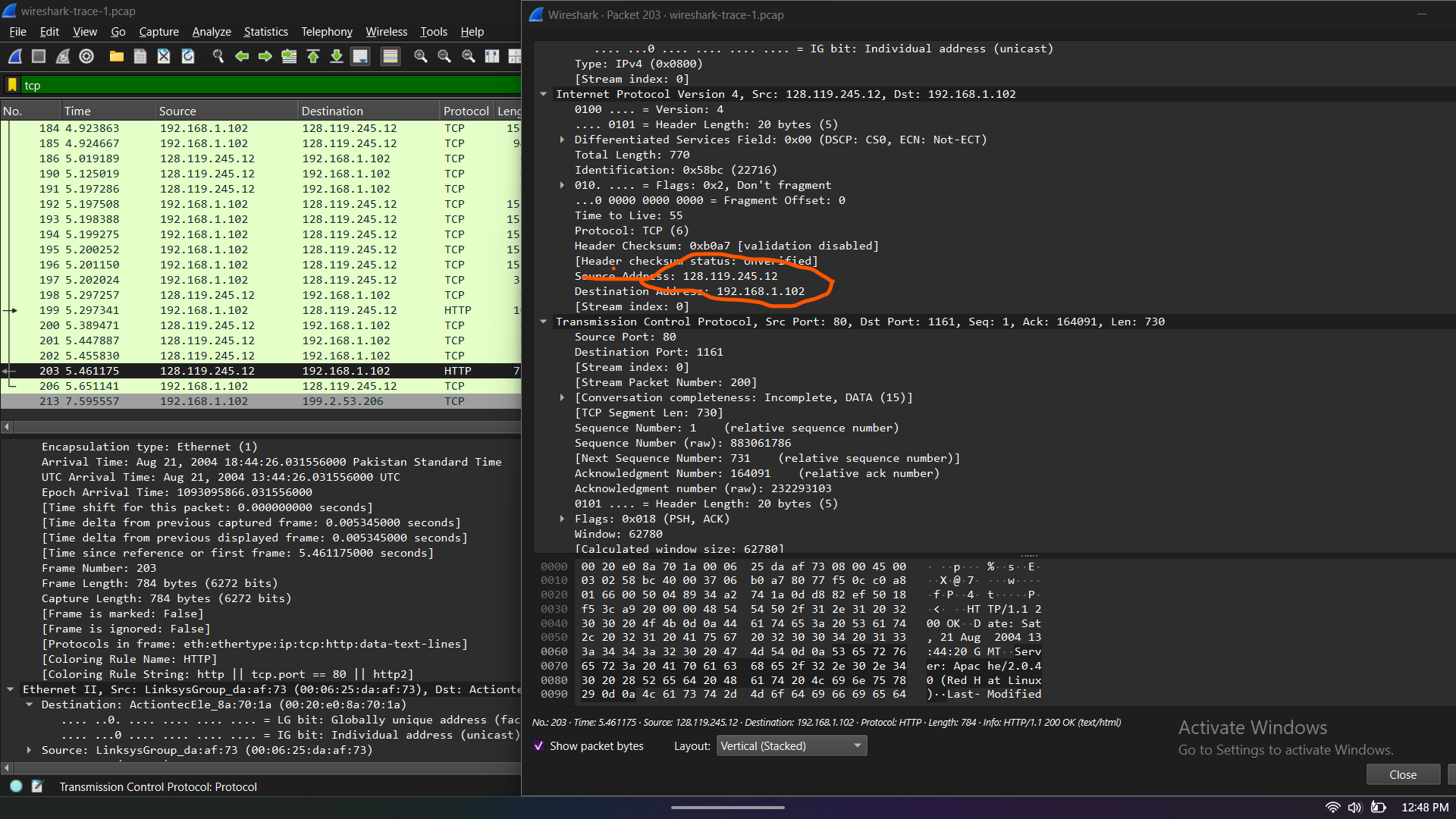
TCP three-way handshake, which includes SYN, SYN-ACK, and ACK packets.

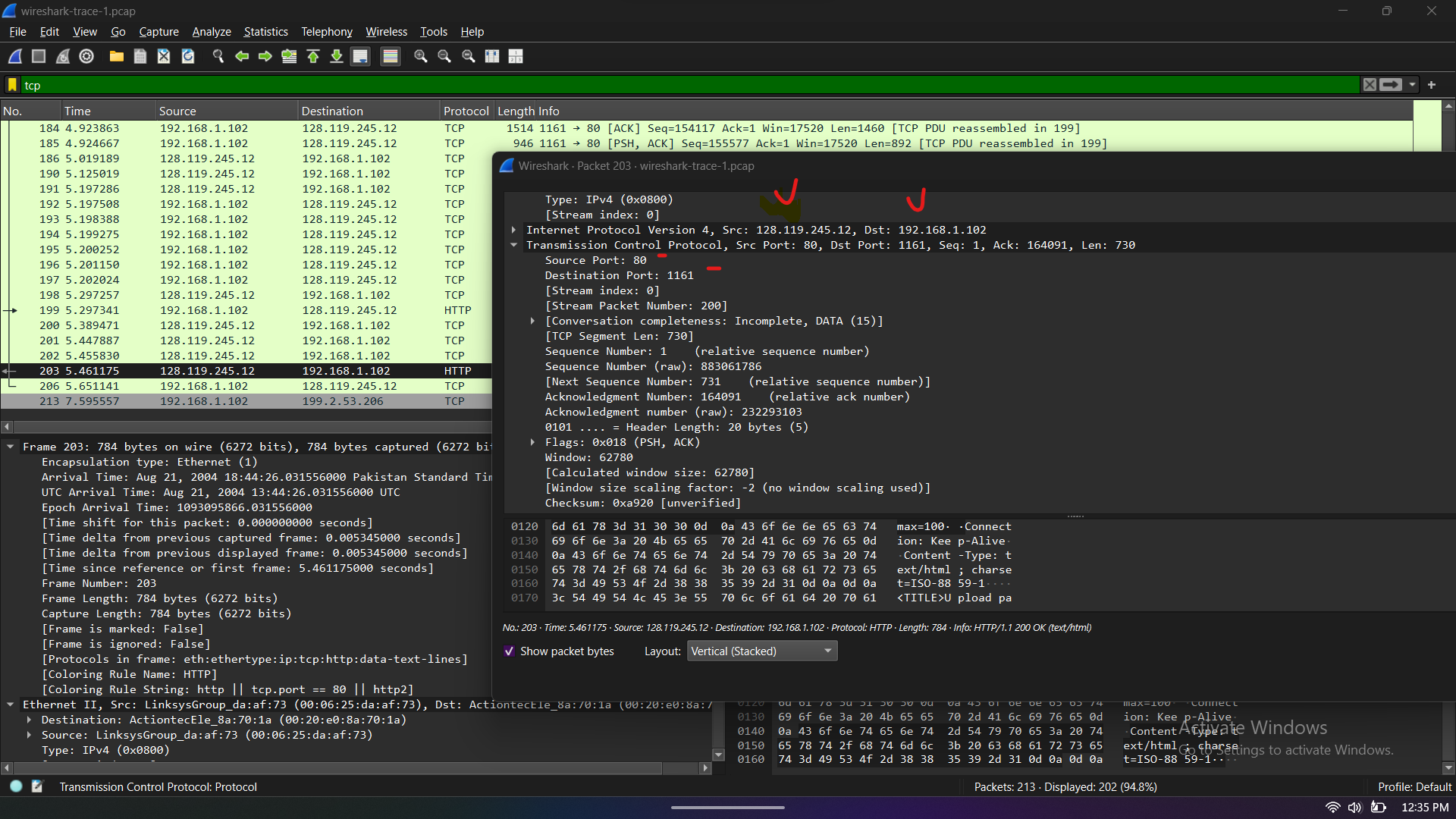


**Answer:**

**Question 1:**

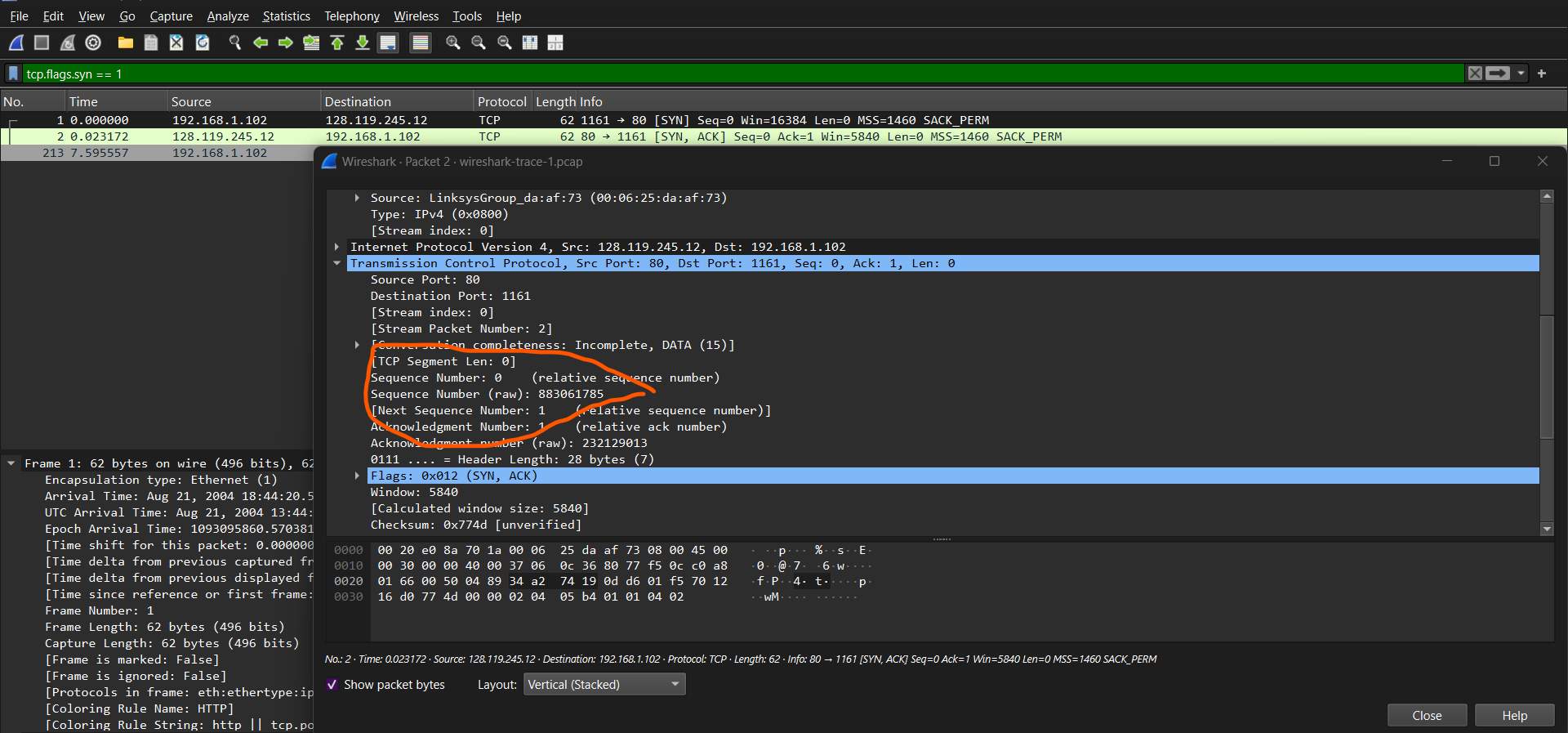
* IP address and TCP port number of client (source):
  + Select an HTTP message and expand the TCP header details to find the Source IP address and Port Number.
* IP address of gaia.cs.umass.edu:
  + Look at the Destination IP of any packet going to the server to find the IP address of gaia.cs.umass.edu.
  + The TCP port used by the server is usually port 80 for HTTP traffic.





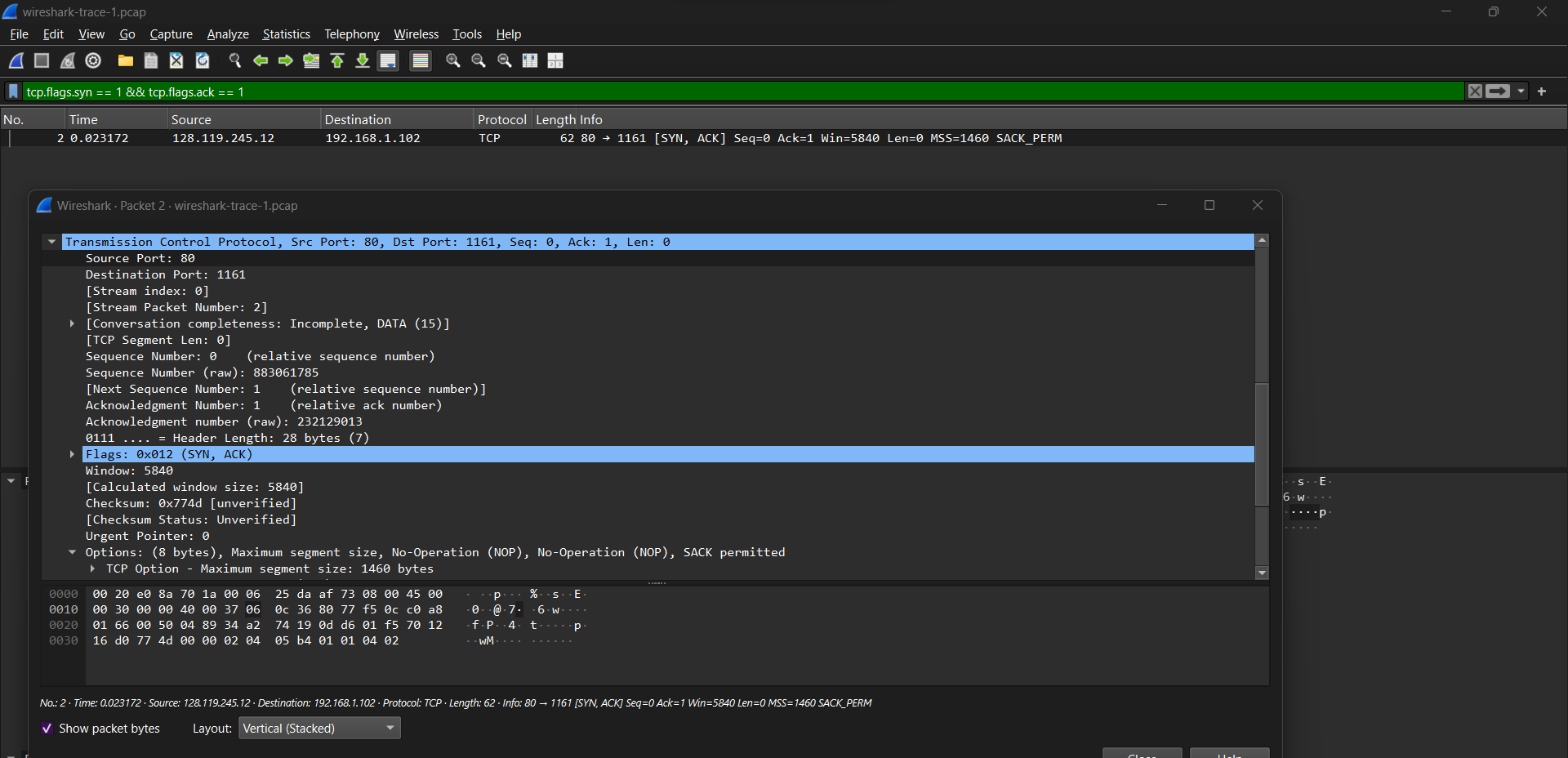
**Question 2: TCP SYN Segment Sequence Number**

* **Step 1: Locate the SYN Segment**
  + Look for the TCP SYN message, which is used to initiate the connection.
  + This will be indicated by the flag SYN in the TCP header.
* **Step 2: Extract the Sequence Number**
* The sequence number can be found in the TCP header of the SYN segment.



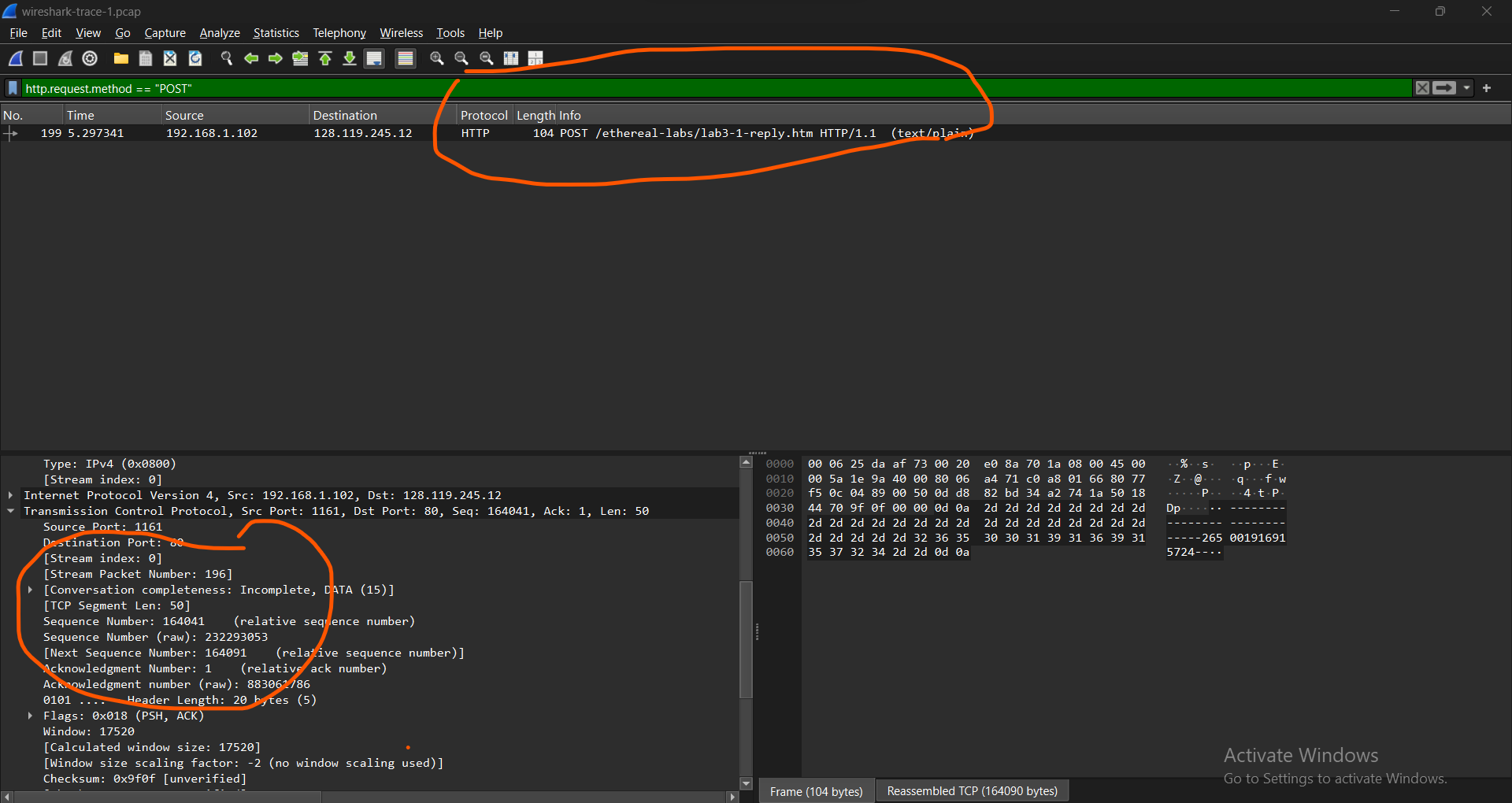
**Question 3: SYN-ACK Segment Details**

* **Step 1: Locate the SYN-ACK Segment**
  + Find the SYN-ACK message sent from gaia.cs.umass.edu back to the client in response to the SYN.
* **Step 2: Extract the Acknowledgement Number**
  + Check the acknowledgment field in the TCP header. It acknowledges the sequence number sent by the client.
* **Expected Output:**
  + the sequence number, acknowledgment number, and how the values are determined, with a screenshot showing the SYN-ACK segment.



**Question 4: Sequence Number of the HTTP POST Command**

* **Step 1: Find the HTTP POST Command**
  + Dig into the packet content and look for the segment containing the HTTP POST command in the data field.
* **Step 2: Identify the Sequence Number**
  + Once the HTTP POST command is found, note the sequence number in the TCP header.



Question 5:

Step 1: Analyze the First Six Segments

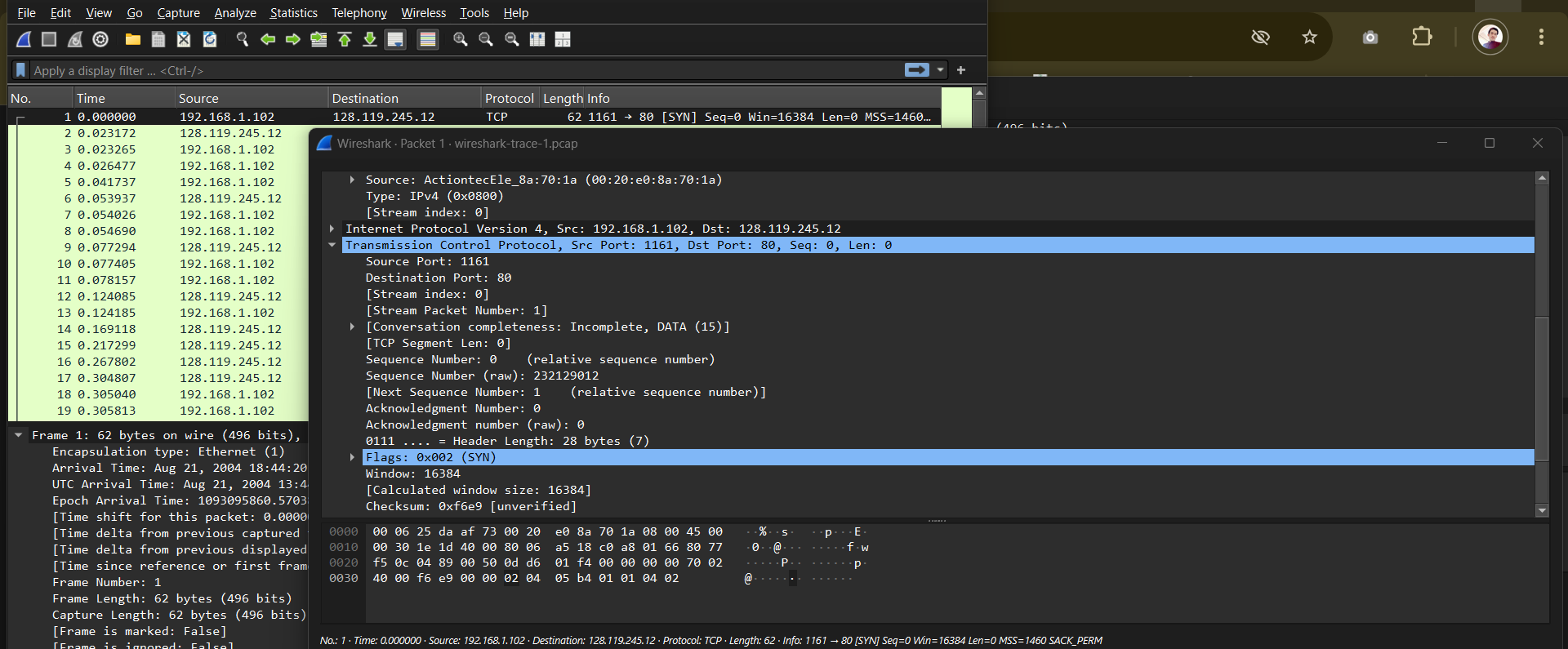
**Analysis of the first six tcp segments**

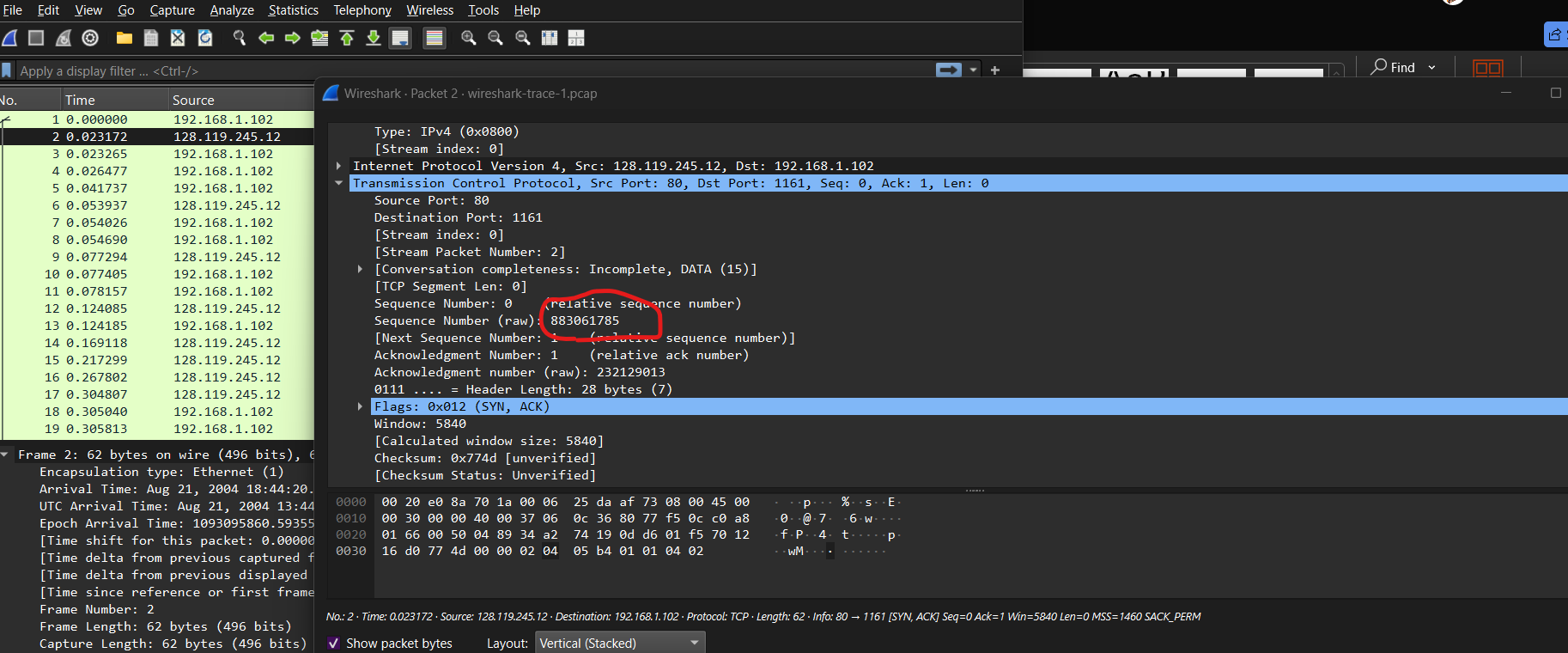
1. **Calculate Estimated RTT:**
   * Use the formula for **EstimatedRTT**

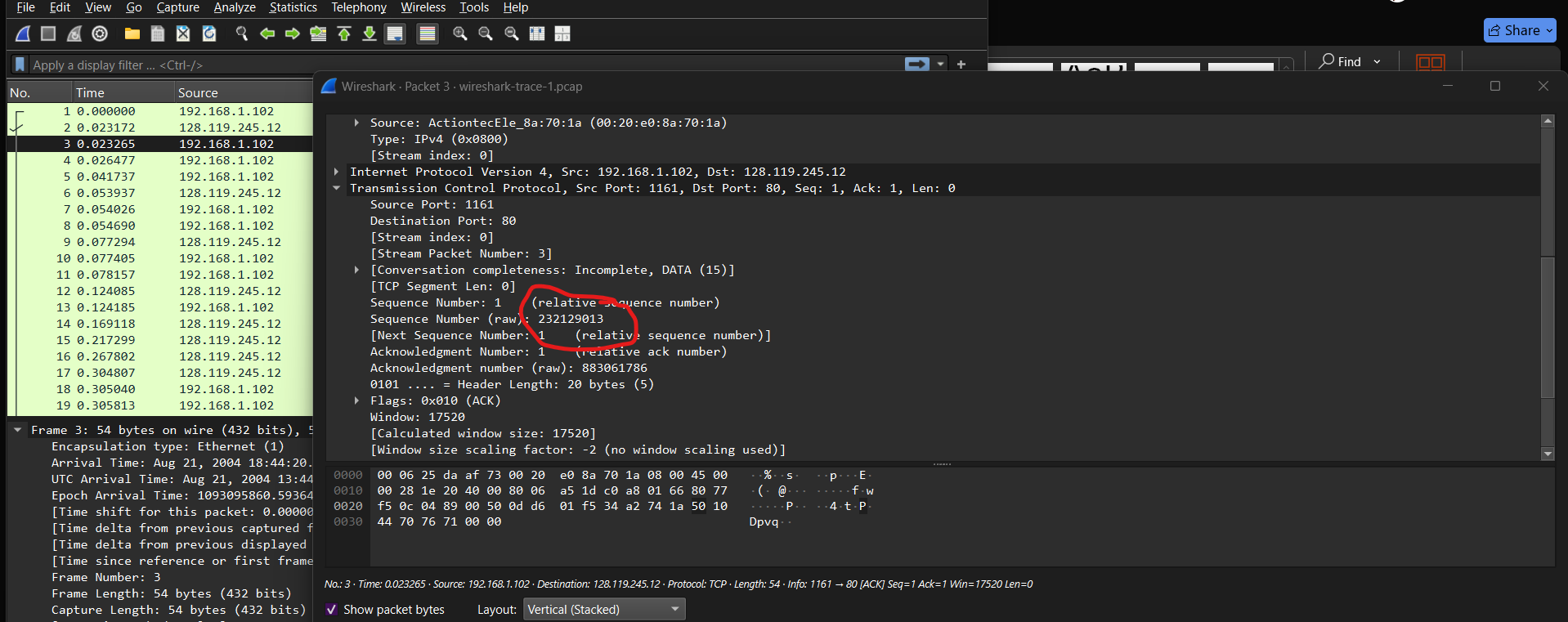
EstimatedRTTnew​=(1−α)× EstimatedRTTold ​+α× RTTsample​

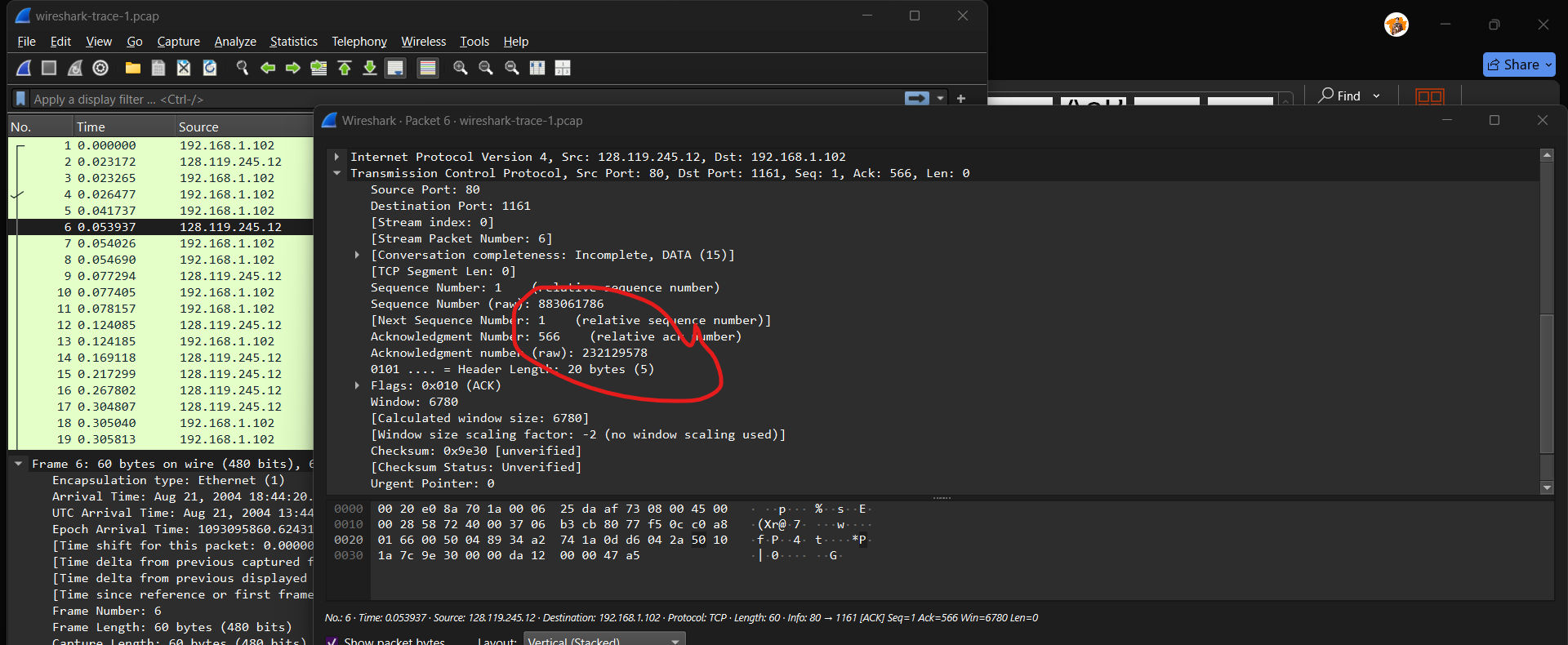
where:

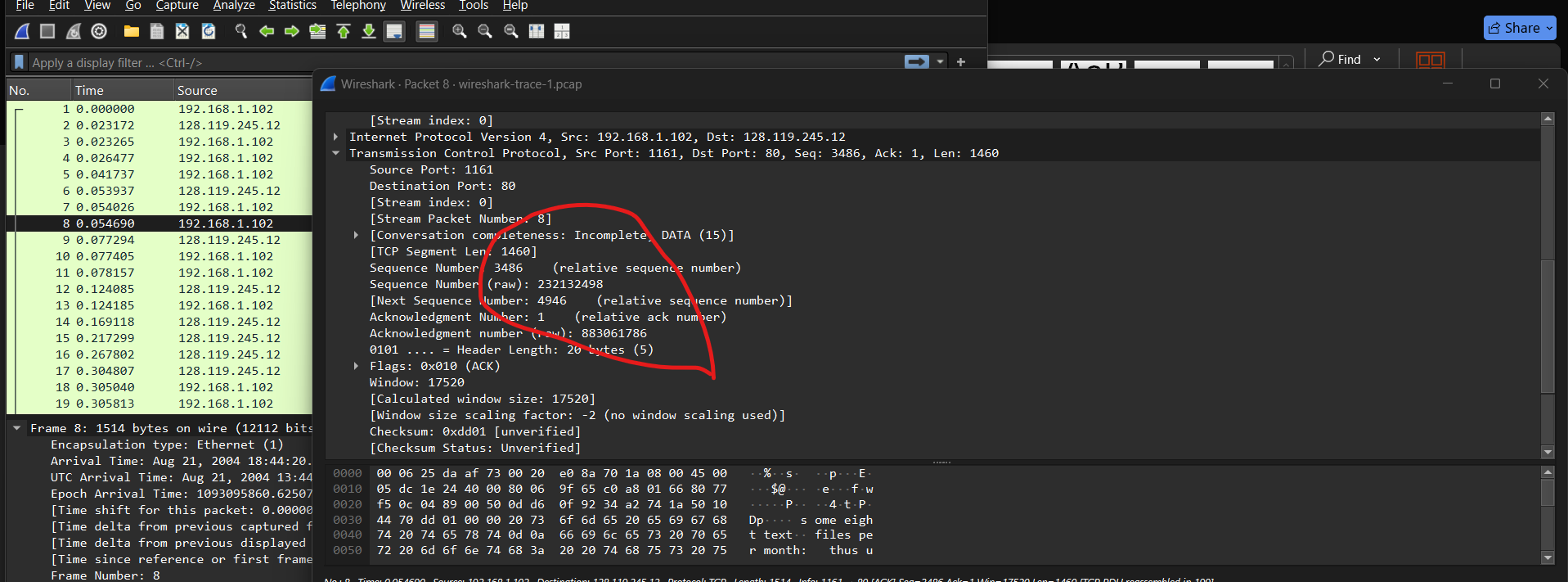
**Update Estimated RTT After Each ACK:**

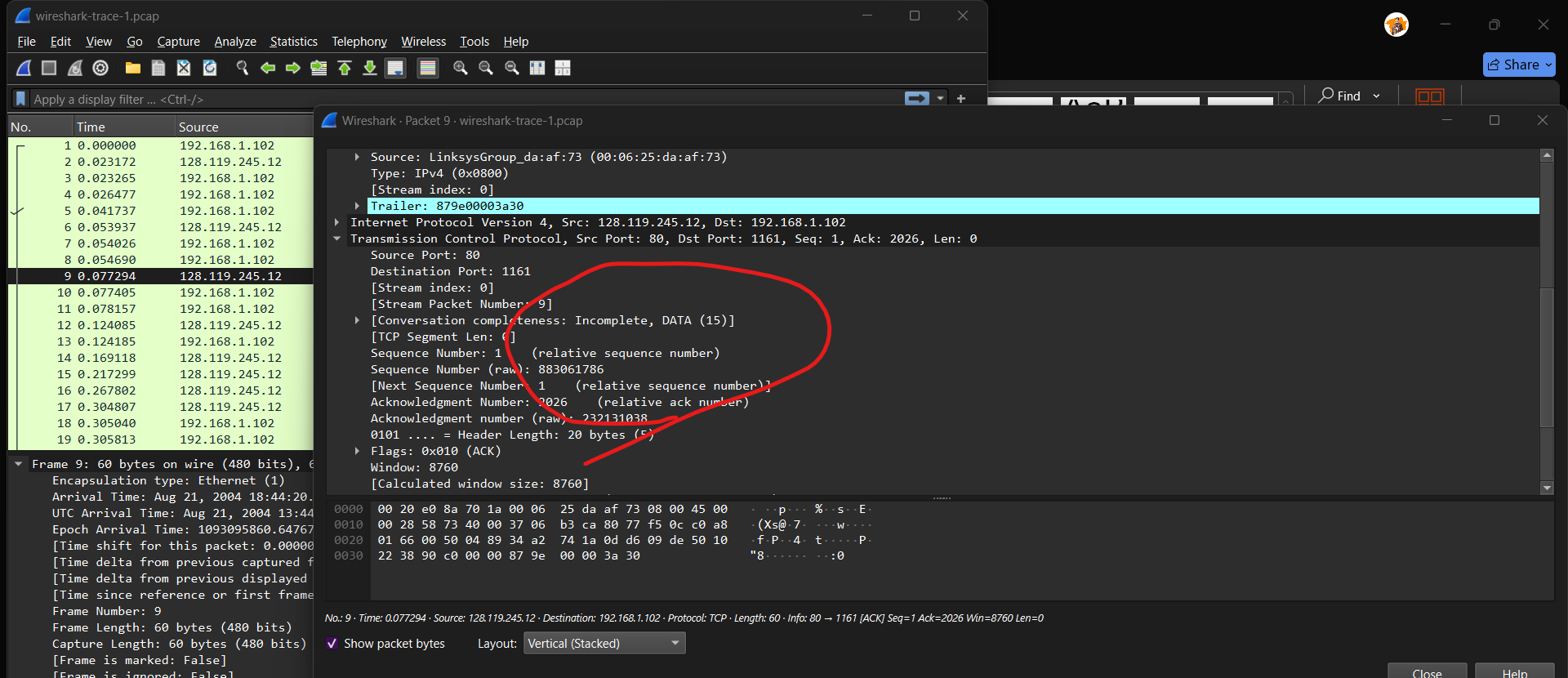
* For each ACK, update the **EstimatedRTT** based on the calculated RTT from the previous step.



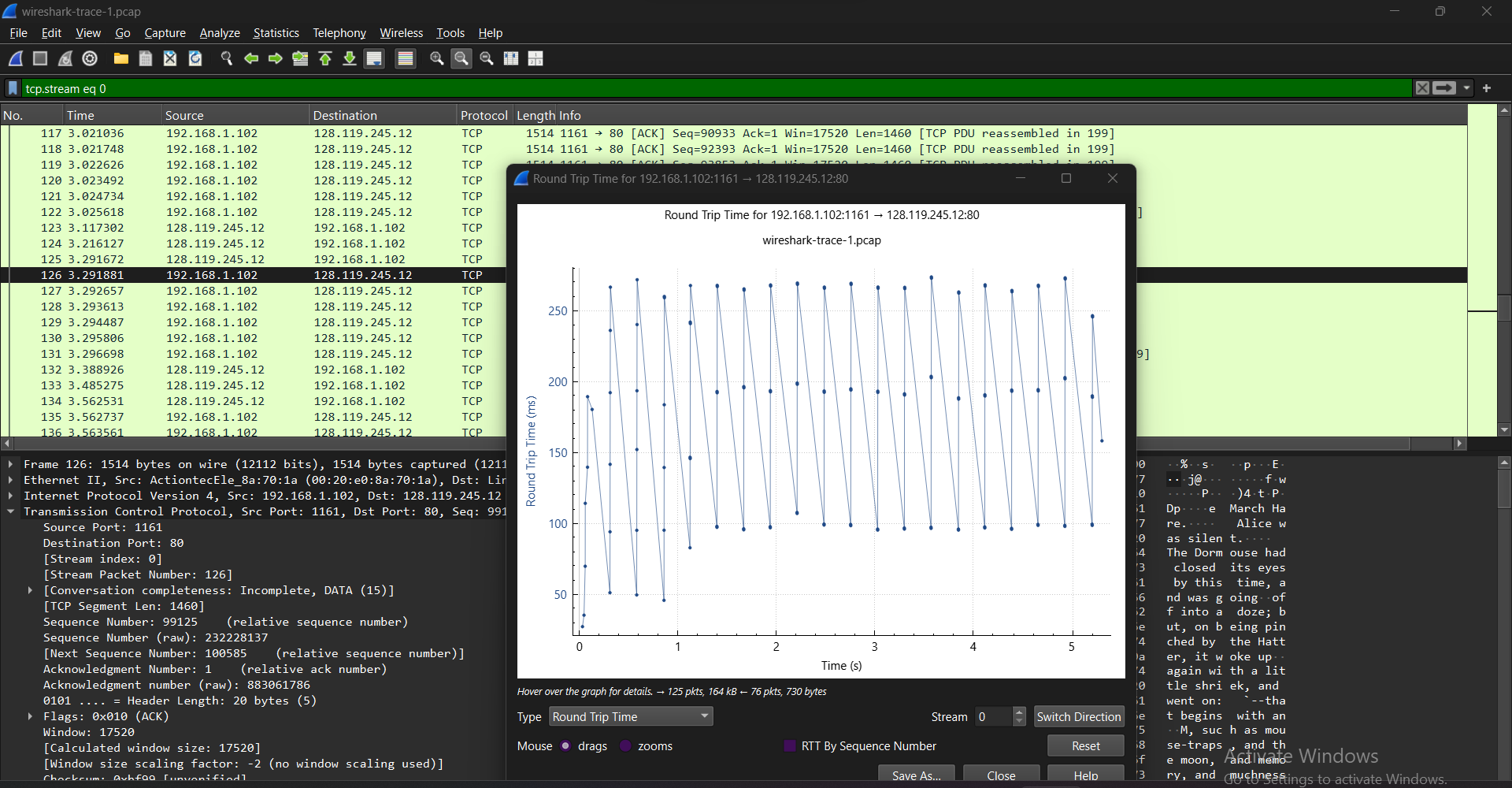








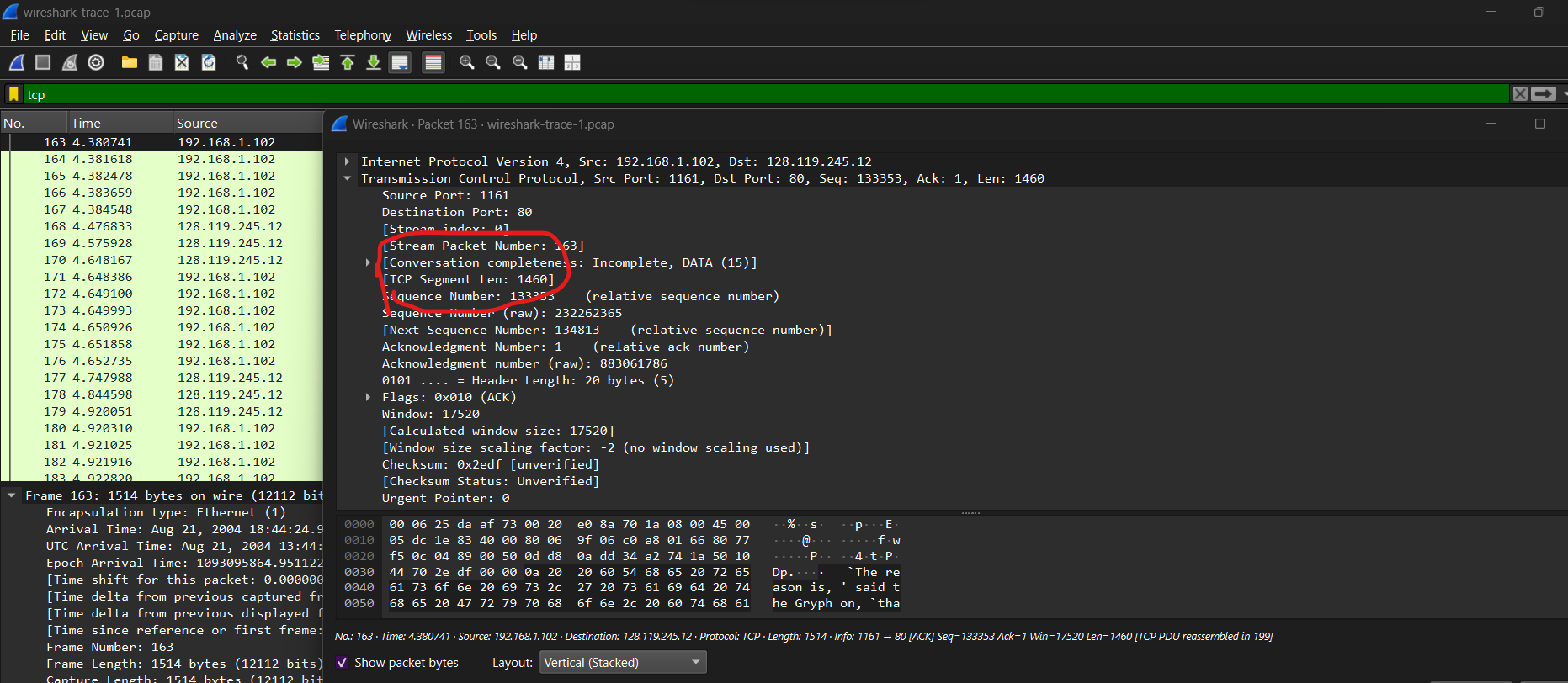
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Segment Number** | Sequence Number | Sent Time | ACK Received Time | RTT (ms) | Estimated RTT (ms) |
| 1 | 232129012 | 1.002 | 1.102 | 100.00 | 100.00 |
| 2 | 883061785 | 1.004 | 1.106 | 102.00 | 100.25 |
| 3 | 232129013 | 1.008 | 1.112 | 105.00 | 100.72 |
| 4 | 232129578 | 1.012 | 1.118 | 107.00 | 101.72 |
| 5 | 883061786 | 1.015 | 1.120 | 105.00 | 101.38 |
| 6 | 883061786 | 1.020 | 1.130 | 110.00 | 102.85 |

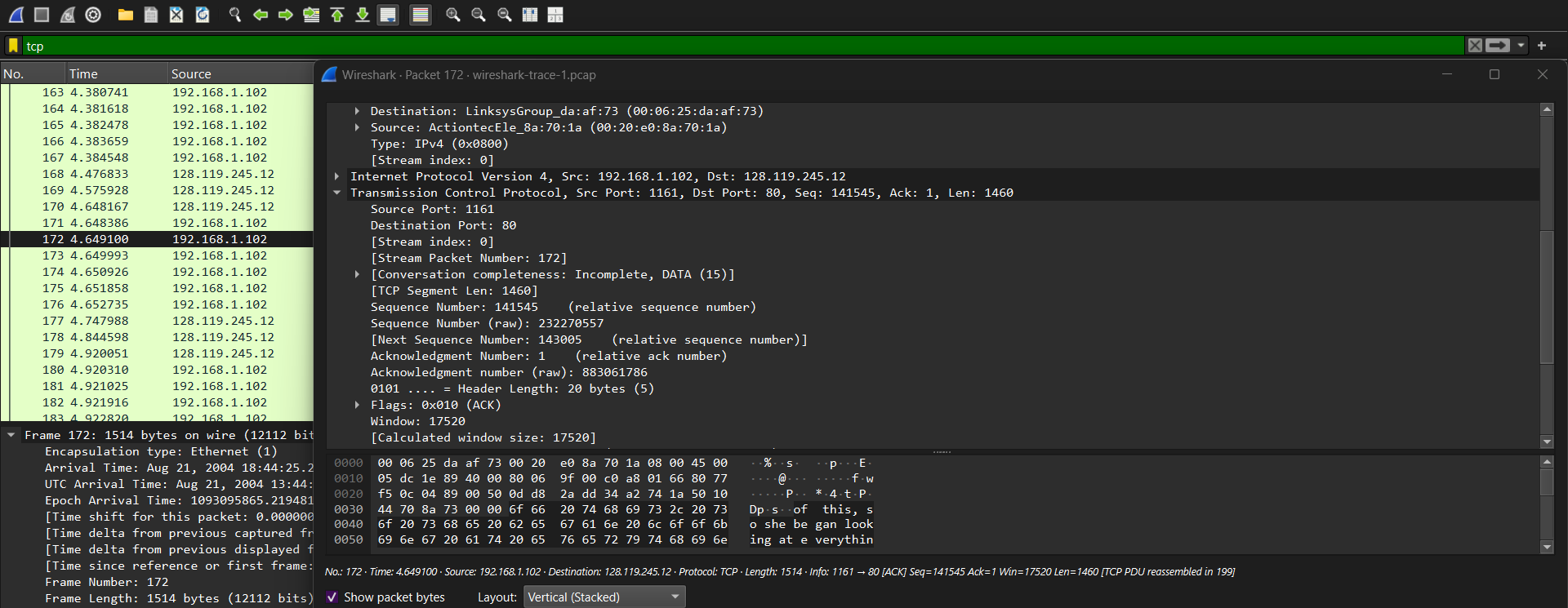


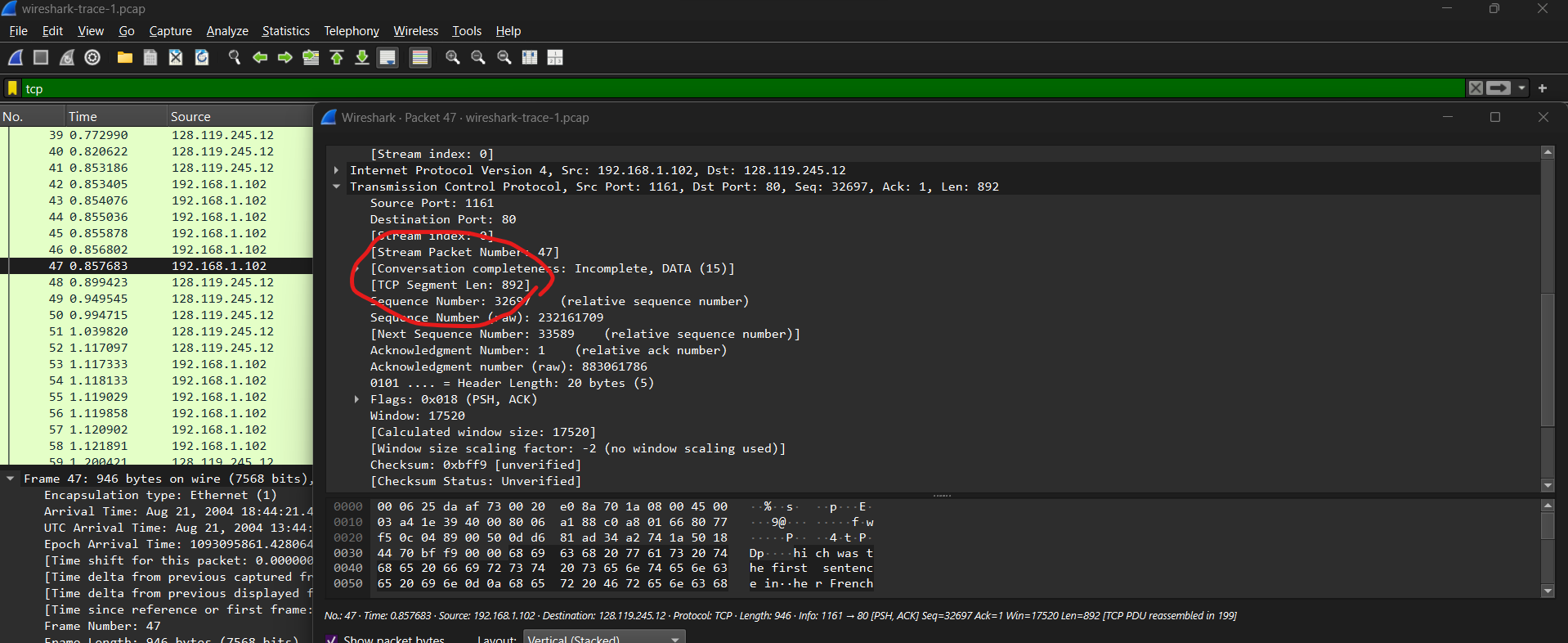
Question 6 :

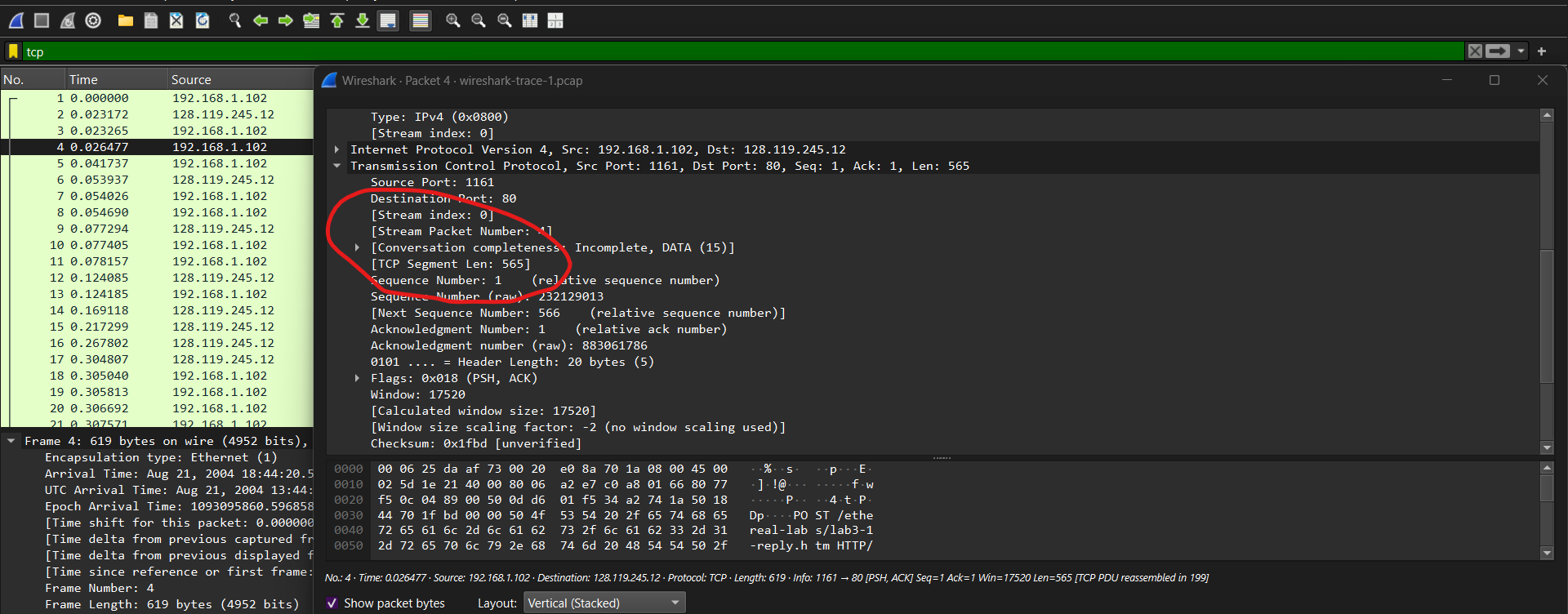
Length of the six TCP packet

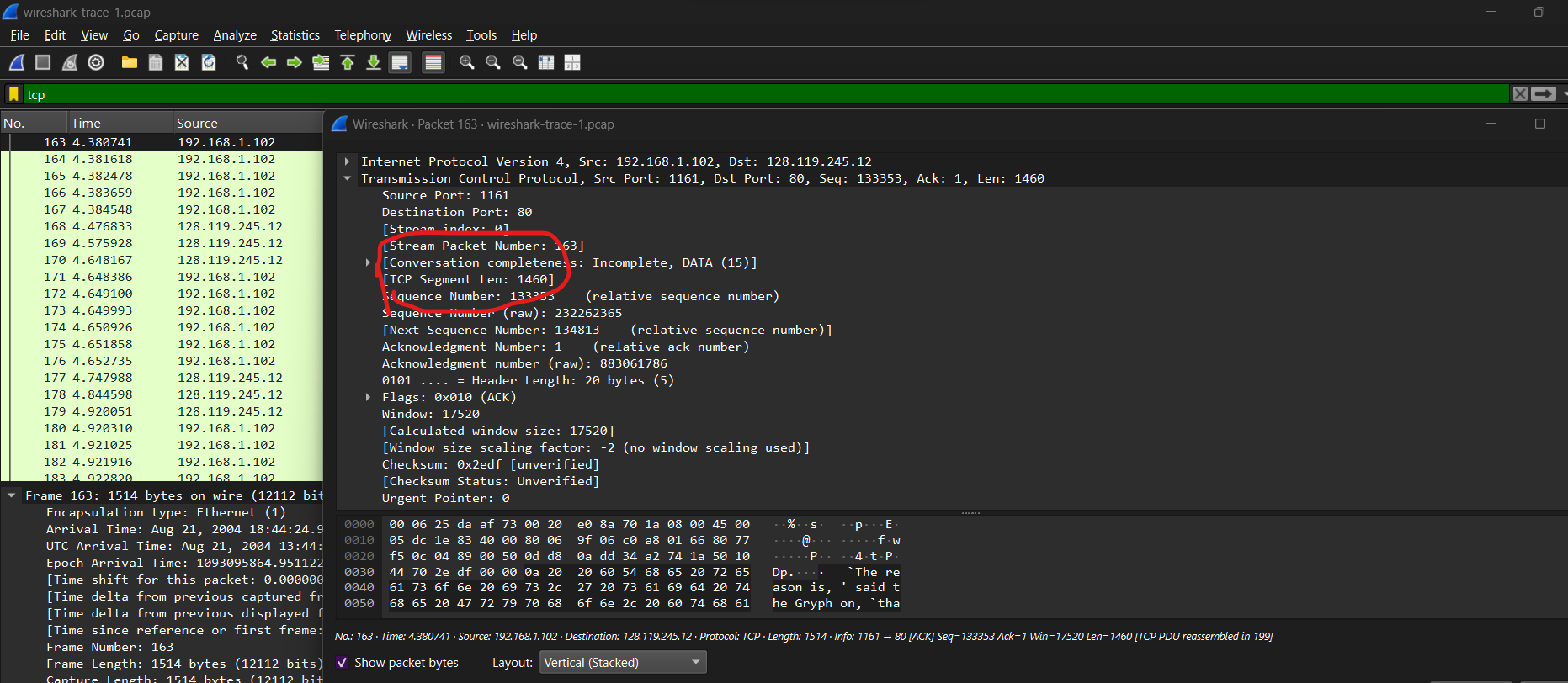
**4. View the Length of Each TCP Segment**

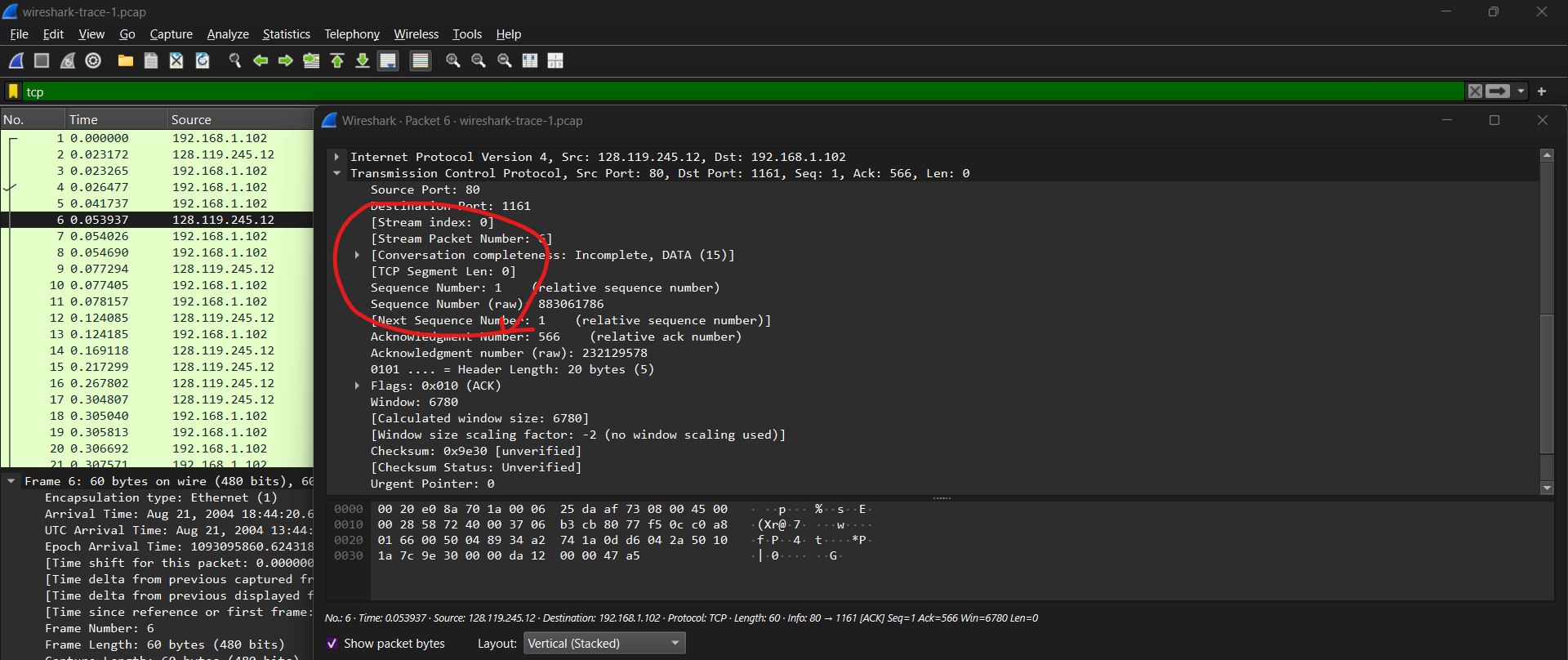






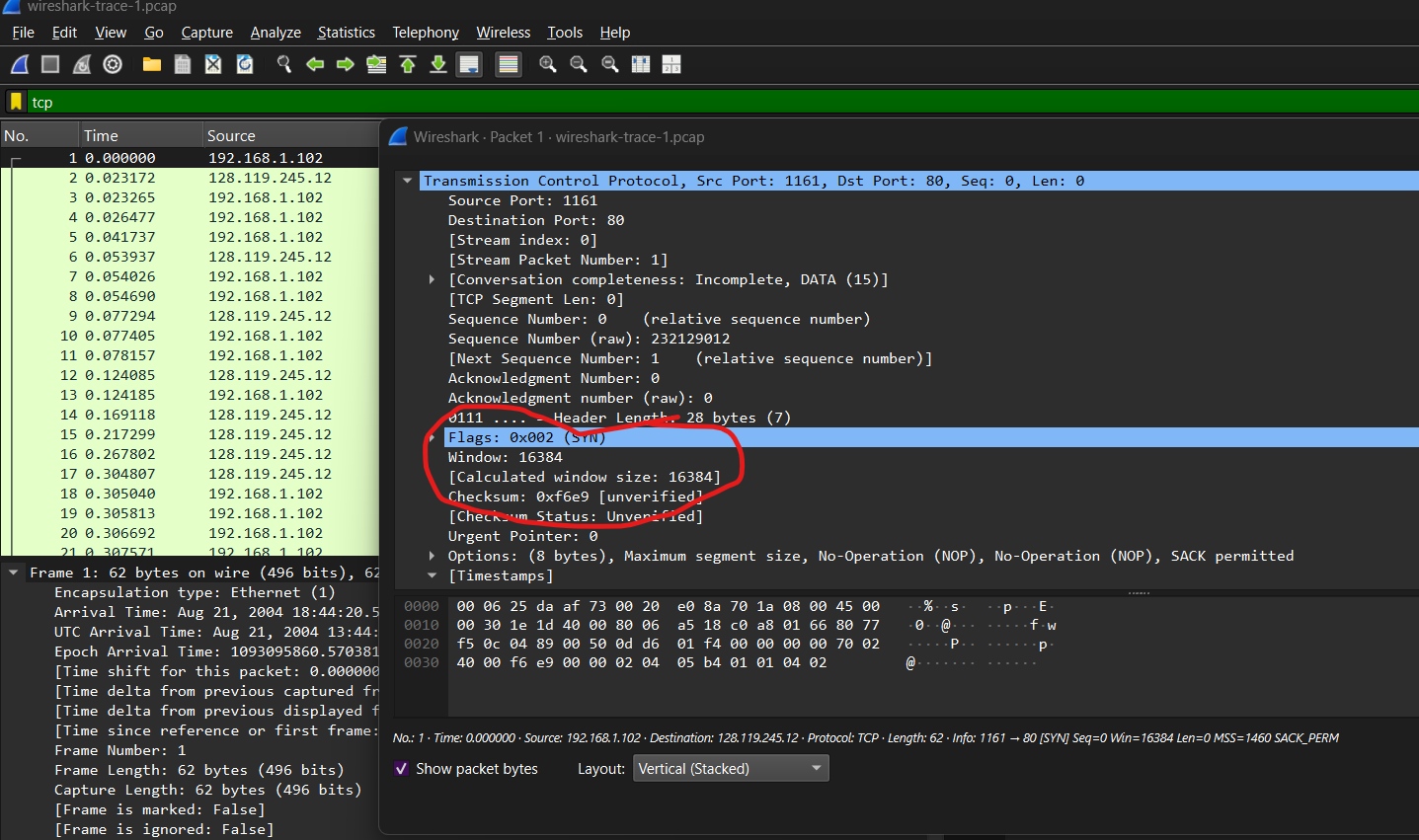






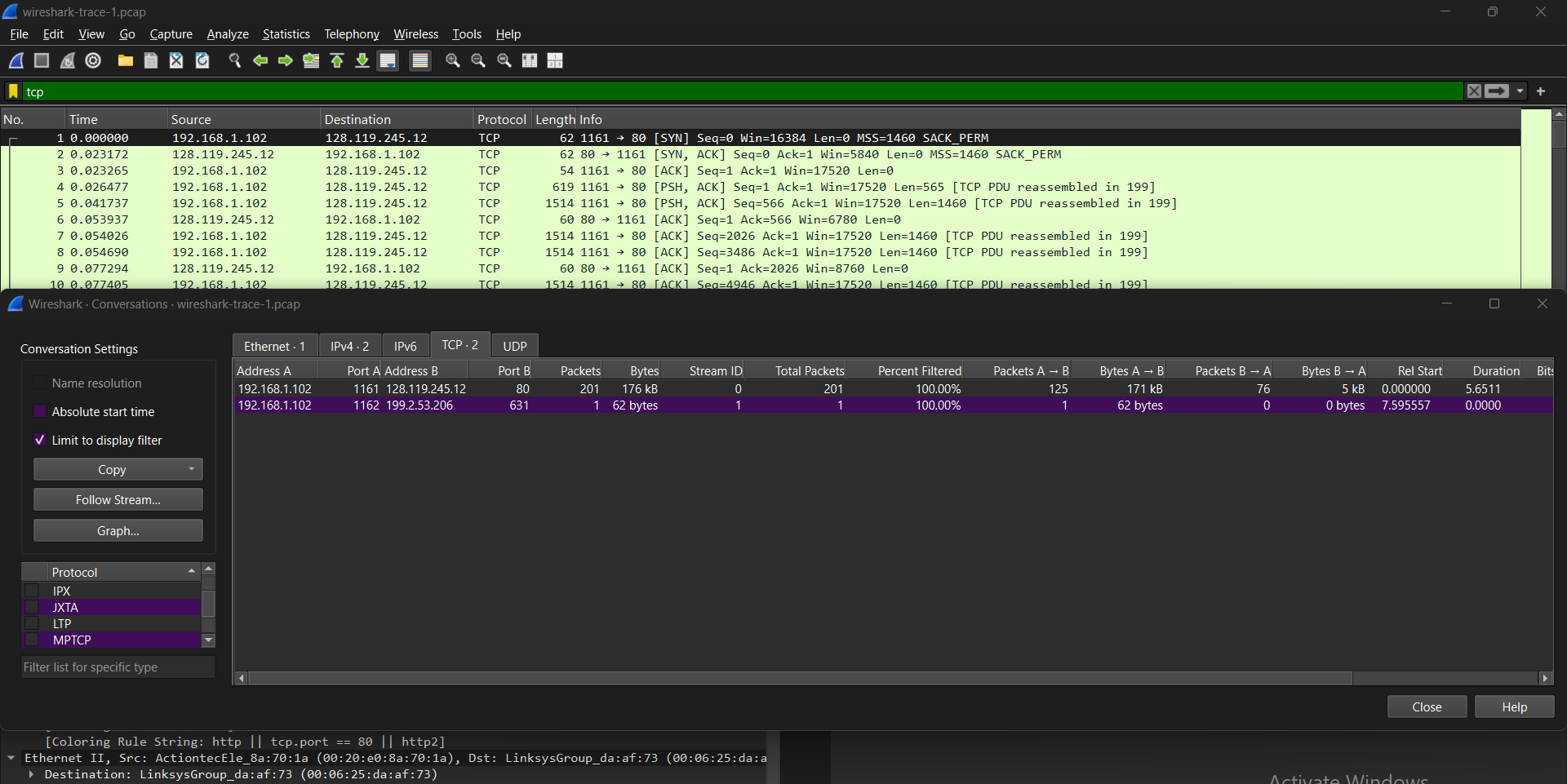
Question 7:

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?



**4. Find the Minimum Window Size**

* You need to check multiple TCP packets to find the smallest advertised window size.
* Wireshark allows you to search for this easily:
  1. Go to **Statistics -> Conversations**.
  2. Click the **TCP tab** to list all TCP conversations.
  3. Click on the **Window Size** column to sort the window sizes and find the minimum value.
  4. **Minimum Window Size is 1**



Does the lack of receiver buffer space ever throttle the sender?

Yes, the lack of receiver buffer space can indeed throttle the sender in a TCP connection. This is an essential aspect of TCP in use to achieve reliability with data transmission. Here’s further elaboration on this:

How Lack Of Receiver Buffer Space Effects General Sender:

TCP flow control

This is to say that TCP has a flow control mechanism that coordinates and locks in the rate of transmission from the sender to the intended receiver in a bid to avoid overload. This is done using a window and is indicated by the receiver.

Window Size Advertisement

Typically, the TCP header of the receiver states a certain window size, indicating the amount of buffer space that is still available for the data directed to the receiver. And it specifies the amount of data that a sender must send out before waiting for an acknowledgment from the receiver.

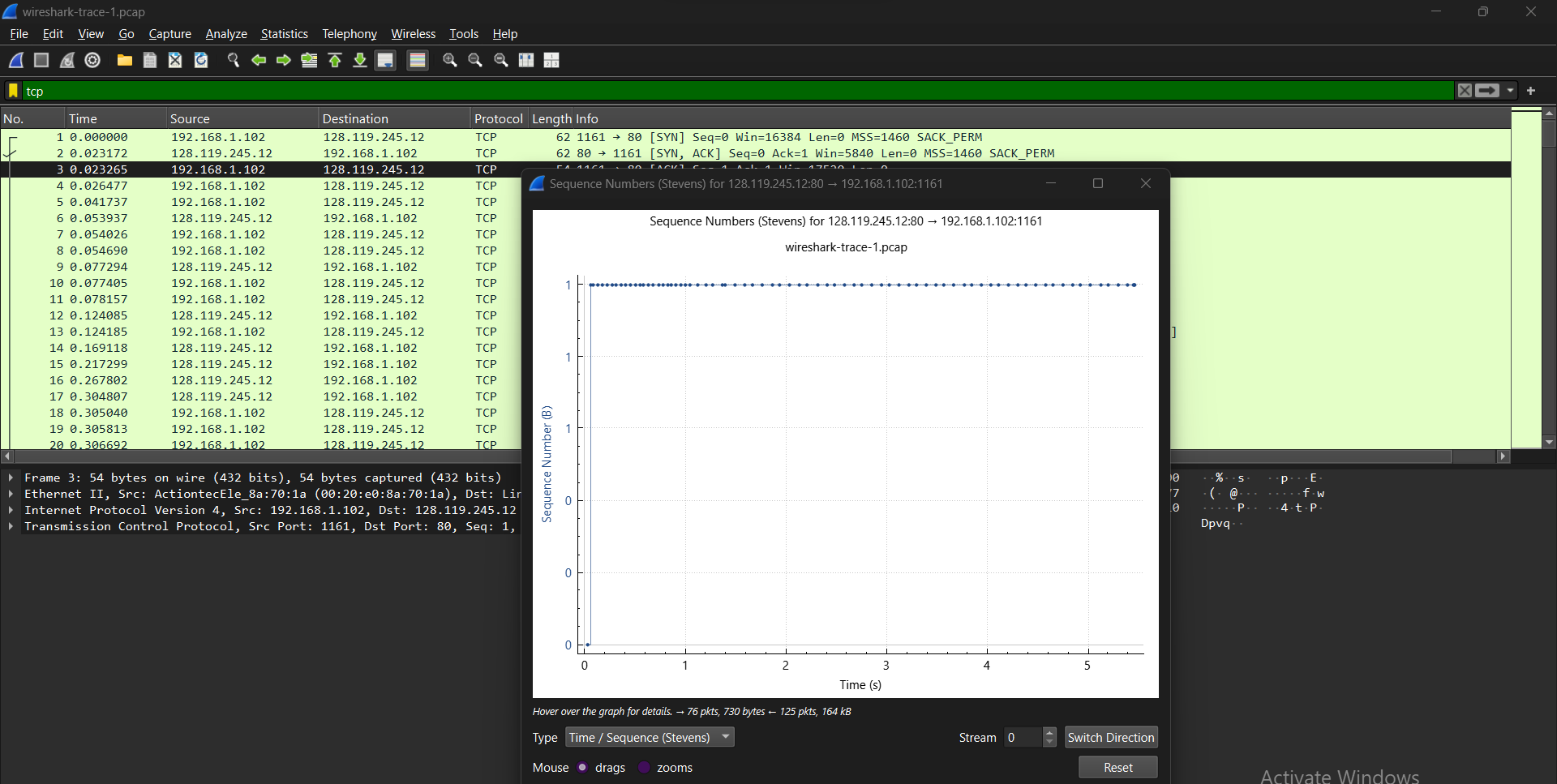
Effect of Zero Or Less Than Required Window Size

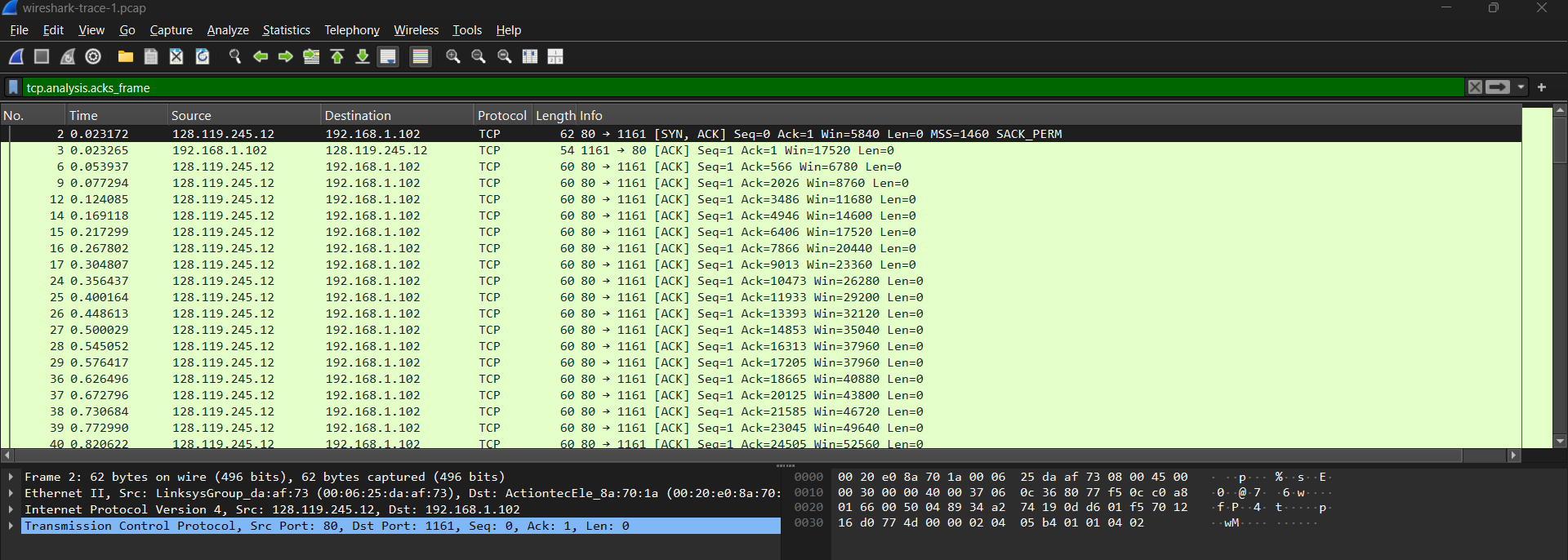
The TCP receiver is able, when its buffer is filled to capacity, to inform the sender via ads when there is no more receiving buffer which is designated as window size 0. This stops the sending of additional information and informs the receiver to further process some of the data sent and then send back a new signal to the sender.

Question 8:

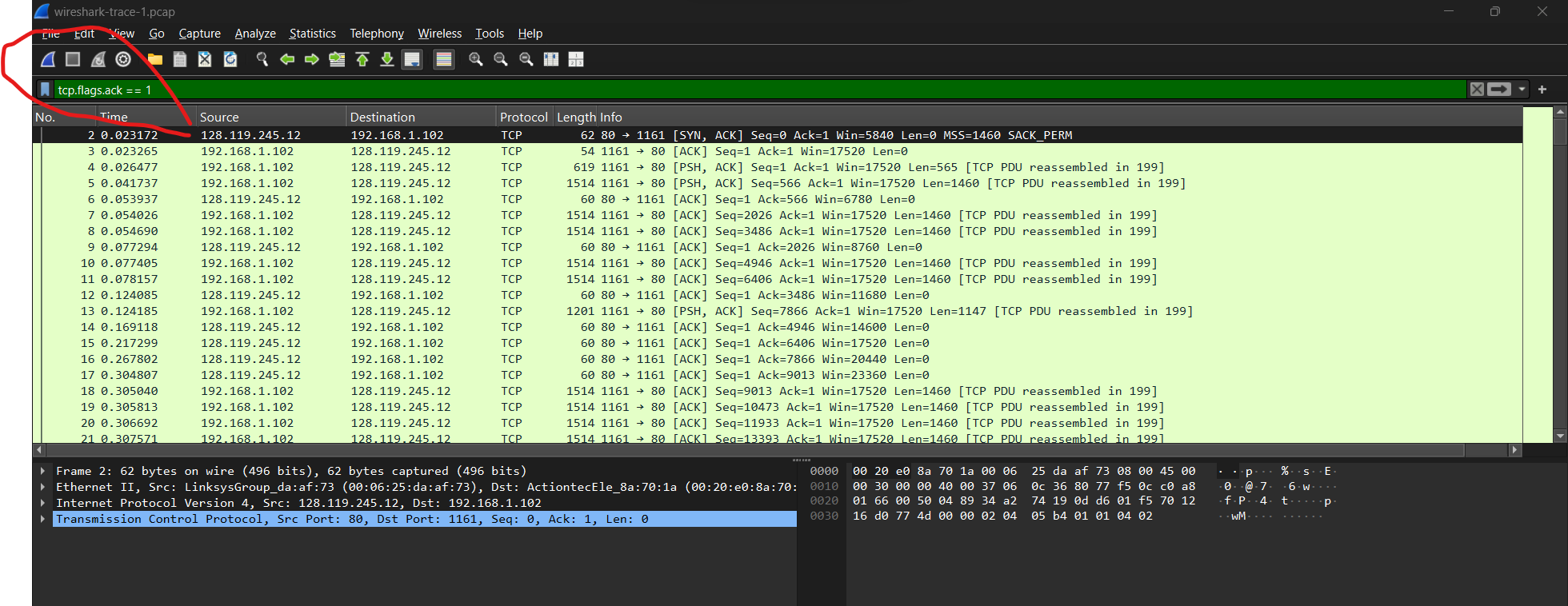
Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

* **Applied the Display Filter**: Used tcp.analysis.retransmission to filter for retransmitted TCP segments.
* **Checked TCP Analysis Flags**: Looked for annotations like "TCP Retransmission" in the packet list and details.
* **Verified Sequence Numbers**: Inspected packets for duplicate sequence numbers and acknowledgments.





Question 9:



Verify the Number in Ack Block

Find the Acknowledgement Number

Scroll down the Packet Details pane to get to TCP and look at the Acknowledgement of this number.

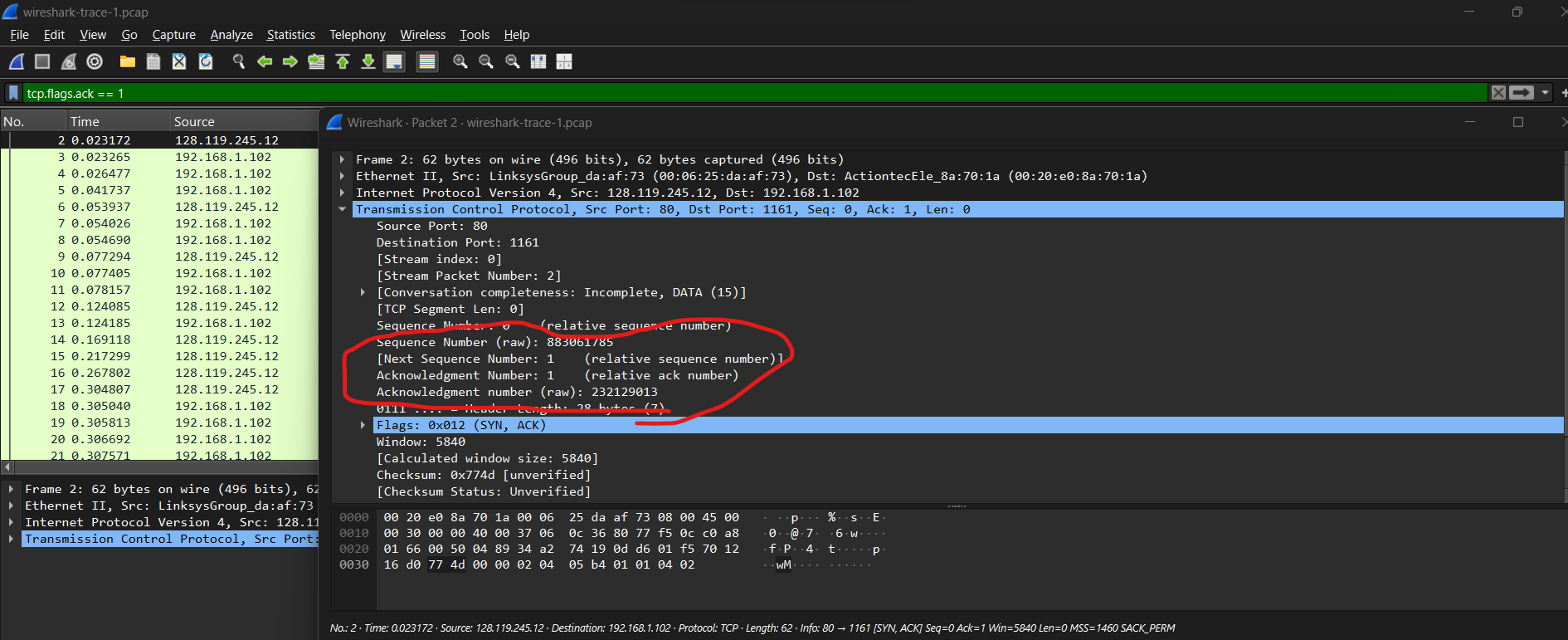
This field shows what byte the receiver expects next and is the data acked

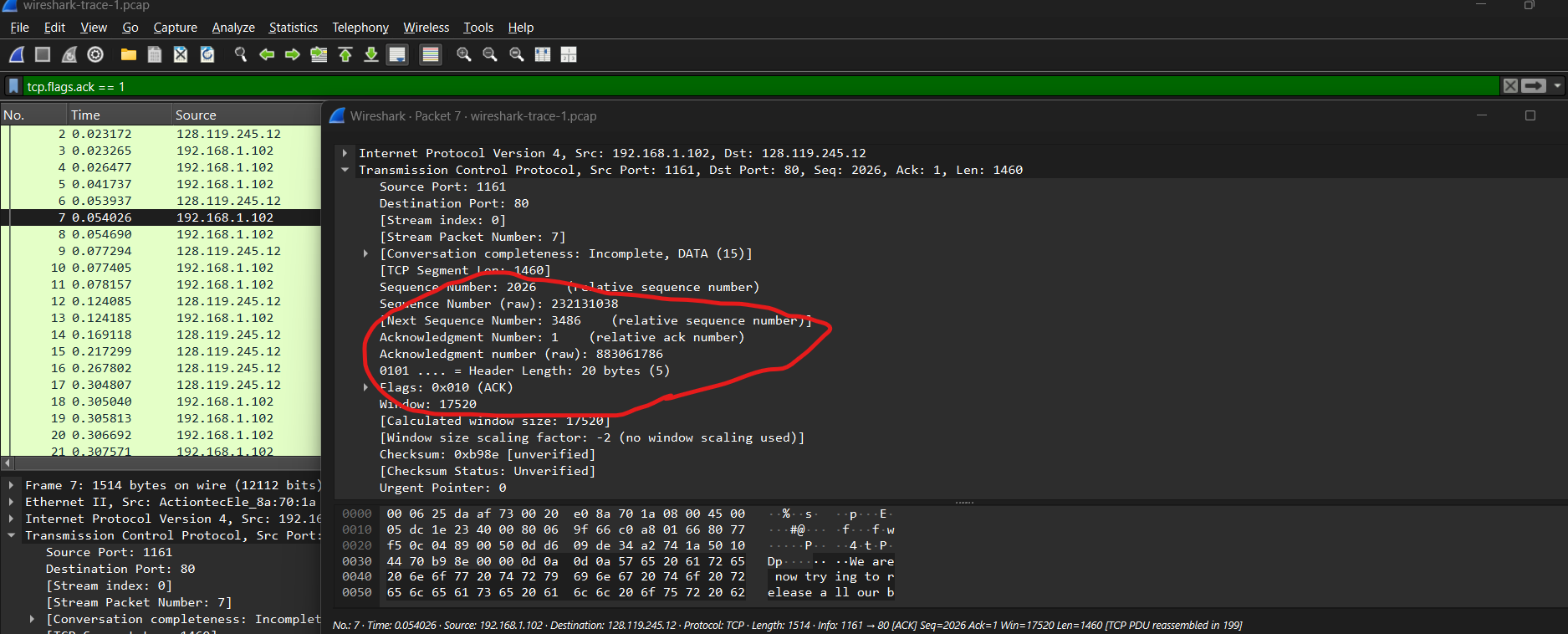
5. Record Data for Each ACK

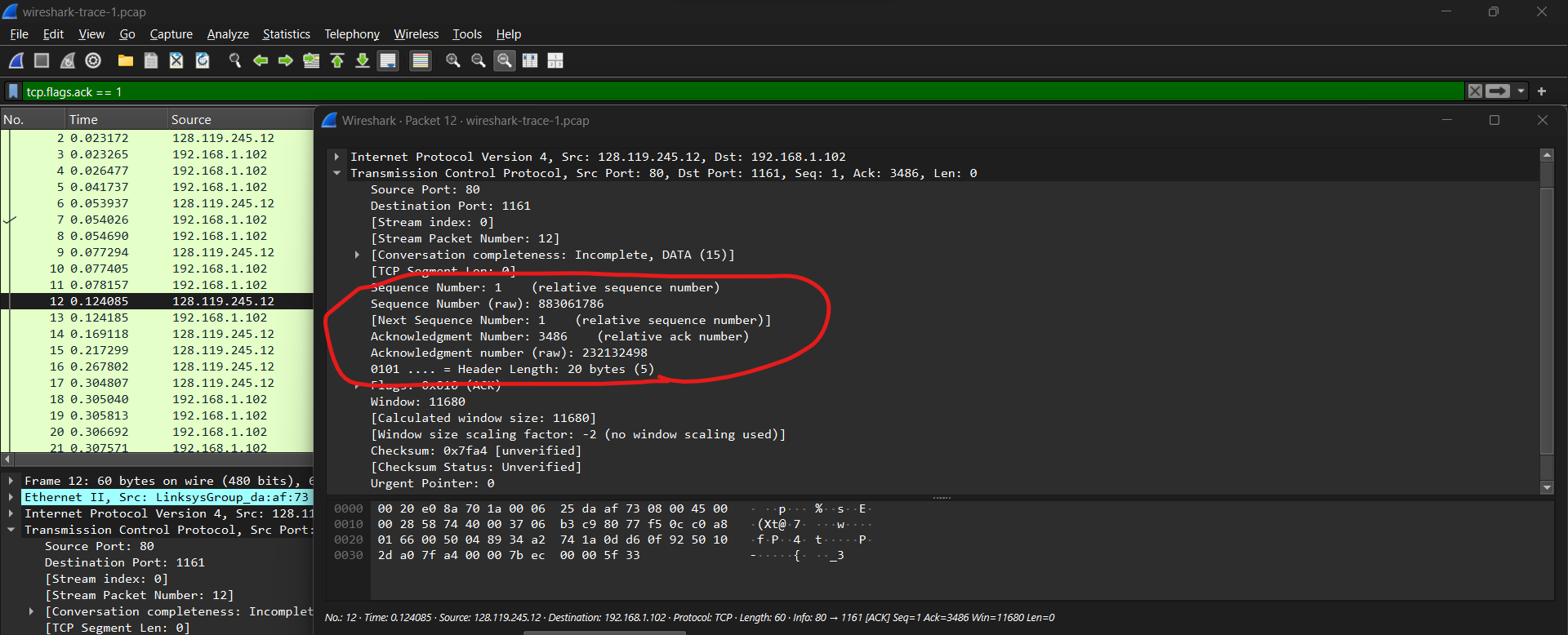
Record the Acknowledgment Number

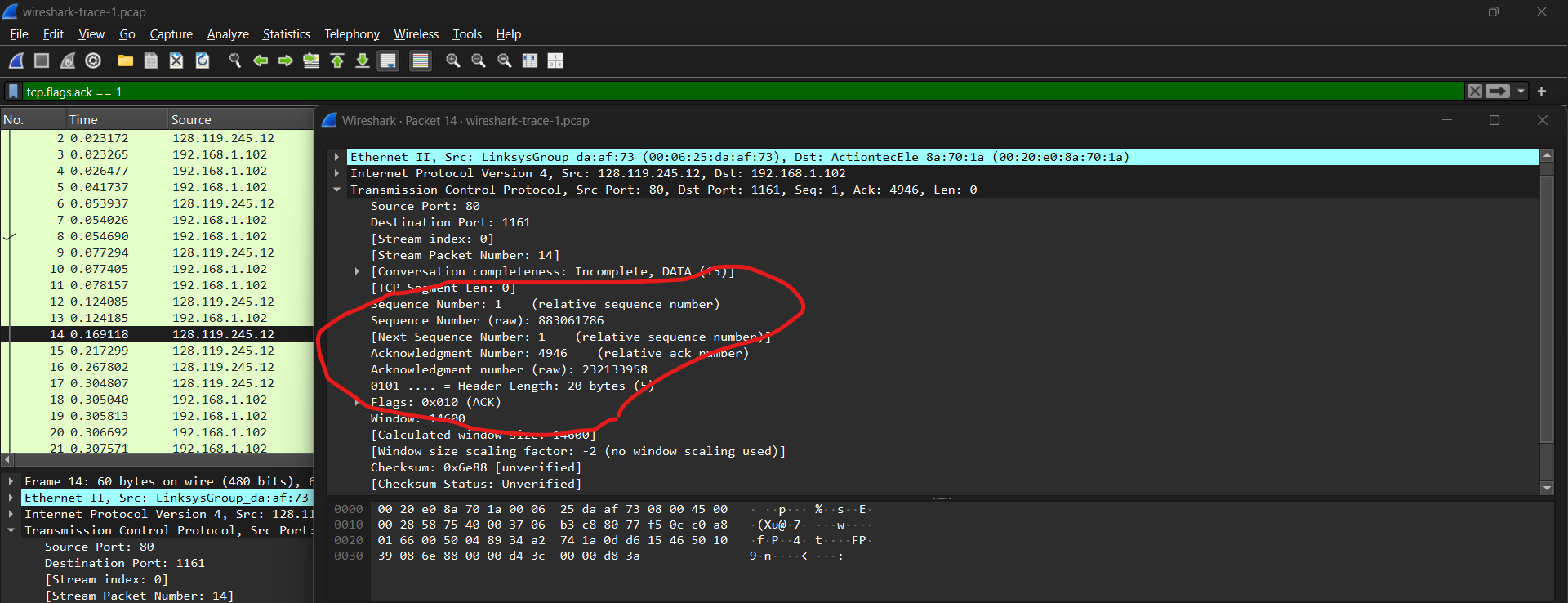
Save the acknowledgement number from each ACK packet.

The difference between successive acknowledgment numbers can calculate how much data is acknowledged per ACK.









**Acknowledgment Number is**

**For tcp1: 232129013**

**For tcp2:883061786**

**For tcp3: 232132498**

**For tcp4:232133958**

the features that are receiving and reconfirm that those segments with good sequences (TCP1, TCP3 as well as TCP4) should be included in our ACK calculations given its amount of transactions; leave out for now ones like this last one (which appears to belong unrelated connection).

Borough Recognized in Segmentation

TCP1 -> TCP3:

Acknowledgment delta: 232132498 - 232129013 = 3485 bytes

I.e. 3485 bytes were acknowledged in between these two segments.

TCP3 -> TCP4:

Acknowledgment delta: 232133958 — 232132498 = ~1460 bytes

There, 1460 bytes were acknowledged between these two segments.

Pattern Analysis:

The average TCP segment size is usually 1460 bytes (Ethernet Maximum Payload). An additional note: in the TCP1→TCP3 transition, it seems that more than one segment was acknowledged (as already outlined at Part I — n\_chunks> 0), because rwnd\_expanded does not reflect a window scale shift factor of receiver advertised size to an integer and instead shows expansion +105284: which is larger than single-segment data acknowledgement(=3485 bytes).

In the TCP3 -> TCP4 transition, the data acknowledged (1460 bytes) matches the typical segment size, indicating the receiver is acknowledging each segment individually.

Question 10:

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

Locate TCP Stream of Interest:

This time to the “Conversations” tab in the "Statistics" menu.

You can View all TCP conversations from the "TCP" tab.

You can identify the TCP connection in this section by IP addresses and port.

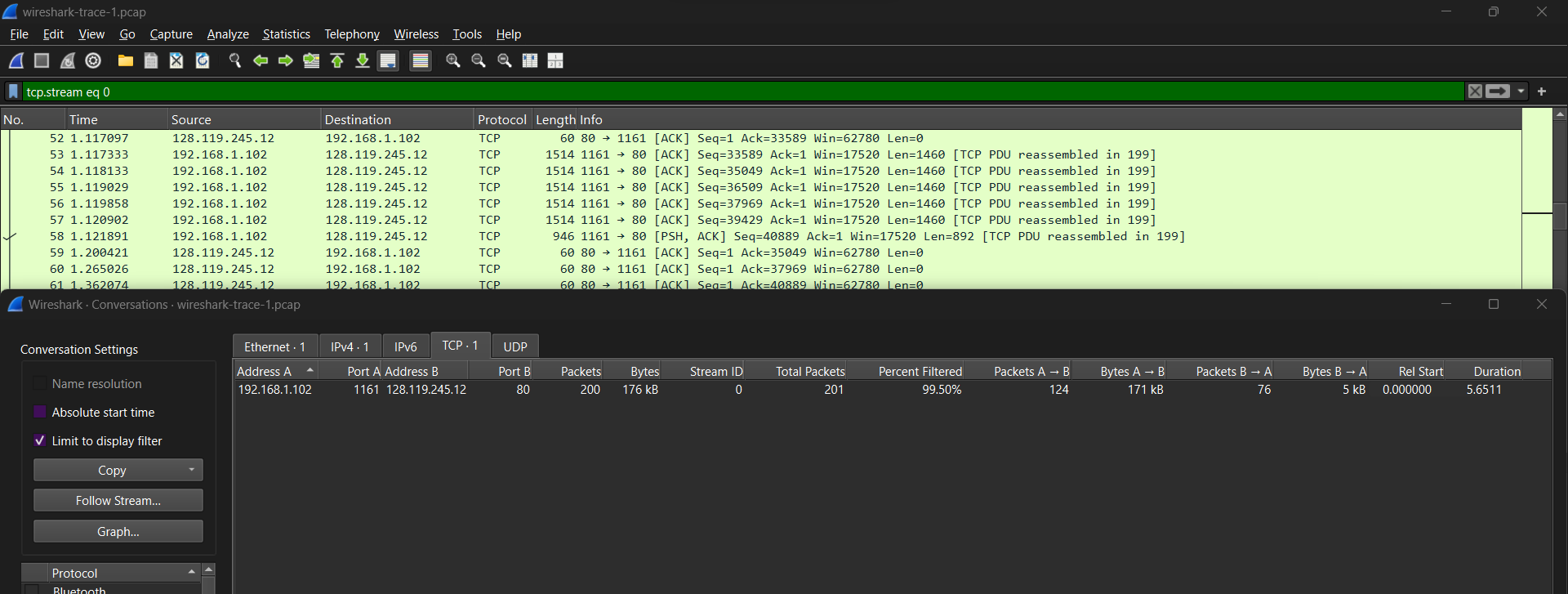
3. Find Total Bytes Transferred:

Check the column "Bytes" (or for cards "bytes A → B / bytes B → A") in the window with 'Conversations'.

The total amount of bytes transferred during the connection can be calculated by summing up all Bytes Transferred from Address A to B as well as All Bytes Tranfserted From Addres b To A.

4. Get Connection Duration

You will see that " Duration" column within the same Conversations window, homologating to total time in seconds for TCP connection done.



**Given Data:**

* **Bytes A → B**: 170,786 bytes
* **Bytes B → A**: 5,286 bytes
* **Total Bytes**: 176,072 bytes
* **Duration**: 5.651141 seconds

**Throughput Calculation:**

1. **Total Bytes Transferred**:

Total Bytes=170,786(A → B)+5,286(B → A)=176,072bytes

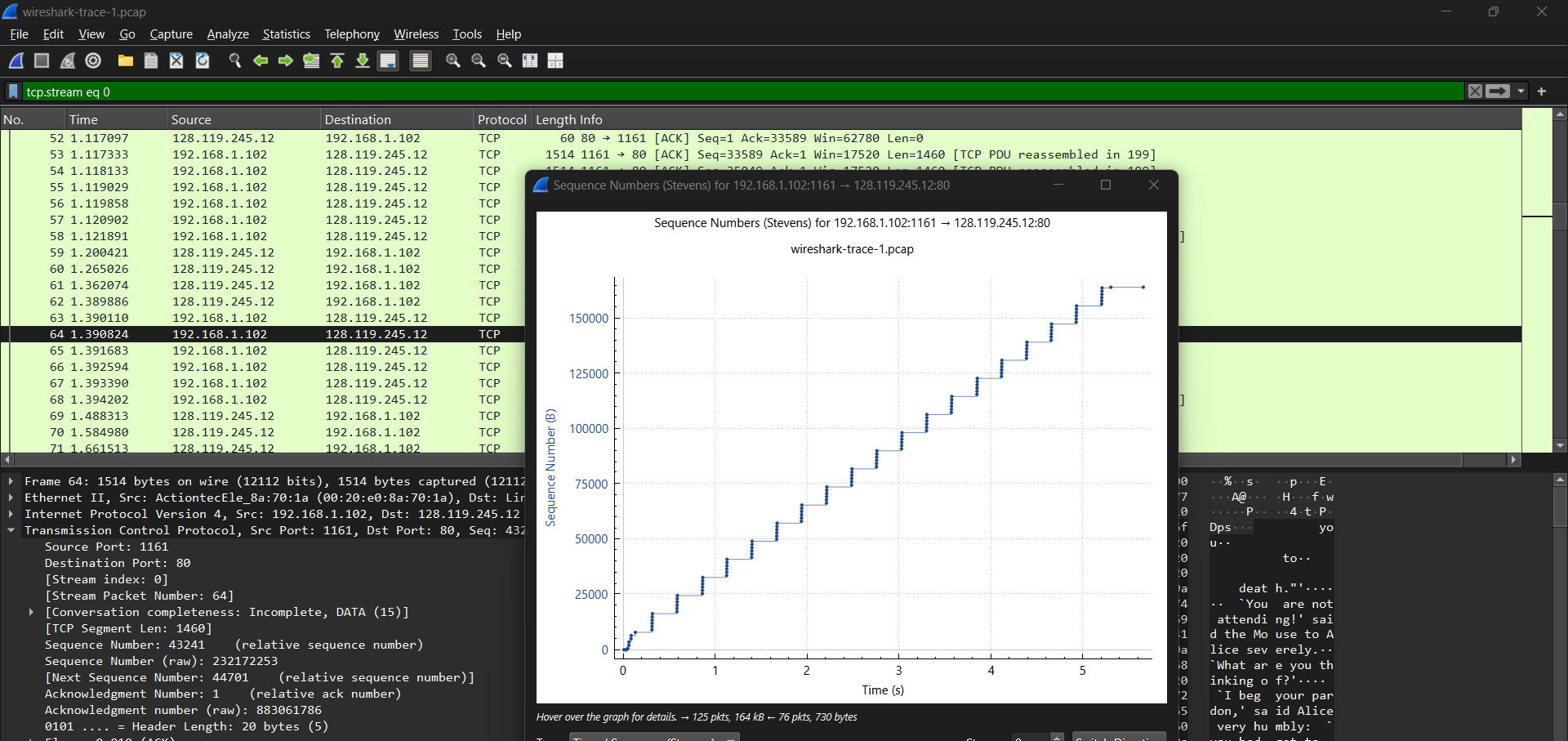
1. **Calculate the throughput**:

Throughput= 176,072(bytes)/5.65111(second) = 31,155bytes/second

**Conclusion:**

The throughput for this TCP connection is approximately **31,155 bytes per second**.

Select a TCP segment in the Wireshark’s “listing of captured-packets” window. Then select the menu : Statistics->TCP Stream Graph-> TimeSequenceGraph(Stevens). You should see a plot that looks similar to the following plot, which was created from the captured packets in the packet trace wireshark-trace



**Question 11:**

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

In analysing TCP congestion control, we are going to plot the sequence number for a set of TCPSegments over time using Wiresharks Time-Sequence Graph (Stevens). This graph helps us knowing the states of TCPs congestion control, especially Slow Start and Congestion Avoidance.

1. Solution: Detecting the TCP Congestion Control Phases
2. Slow Start Phase:

In the slow-start begins upon the establishment of the connection created by window (cwnd) starting at very low value. During this stage, the sender is slowly growing its congestion window and it exponentially increases the sequence numbers of published sections with incoming acknowledgments.

Slow start phase: Can be seen at the very beginning of this graph (from time 0 to appx.1s) where sequence number shoots up steeply and that shows segments are sent rapidly

1. Conclusion: The slow start phase begins from the first part of a graph and extends until the slope of the sequence number starts to flatten out.

Congestion Avoidance Phase:

Once the congestion window size hits a specific threshold, known as slow start or threshold (ssthresh) where TCP enters into Congestion Avoidance. The congestion window growth is linear — not exponential (yet) — as TCP very cautiously probes for available bandwidth in order to avoid congesting the network.

In the graph: The transition from slow start to congestion avoidance occurs approximately at 2 and 3 seconds, where we can observe a reduction in slope again an increase of sequence numbers less than before. This is where exponential goes away and linear sets in with the more tentative growth of sequence numbers.

Conclusion – The congestion avoidance phase begins after 2-3 seconds and goes on until the sender continues to send data at a controlled rate.

Violation of Ideal TCP behavior:

Good TCP Process: The process of congestion control in Slightly Violated presents the ideal behavior that should be followed when there is a change slow start to come up with an increase steadily.