Sai Valluru

Professor Bo Sheng

CS446

11/10/2023

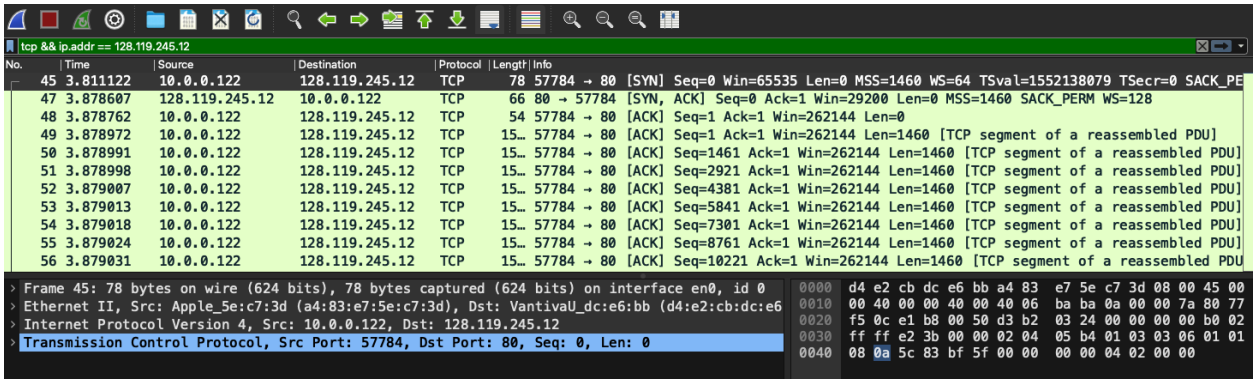
Wireshark Lab: TCP

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?**

   The IP address and TCP port number used by the client computer (source) is:

   IP address = 10.0.0.122

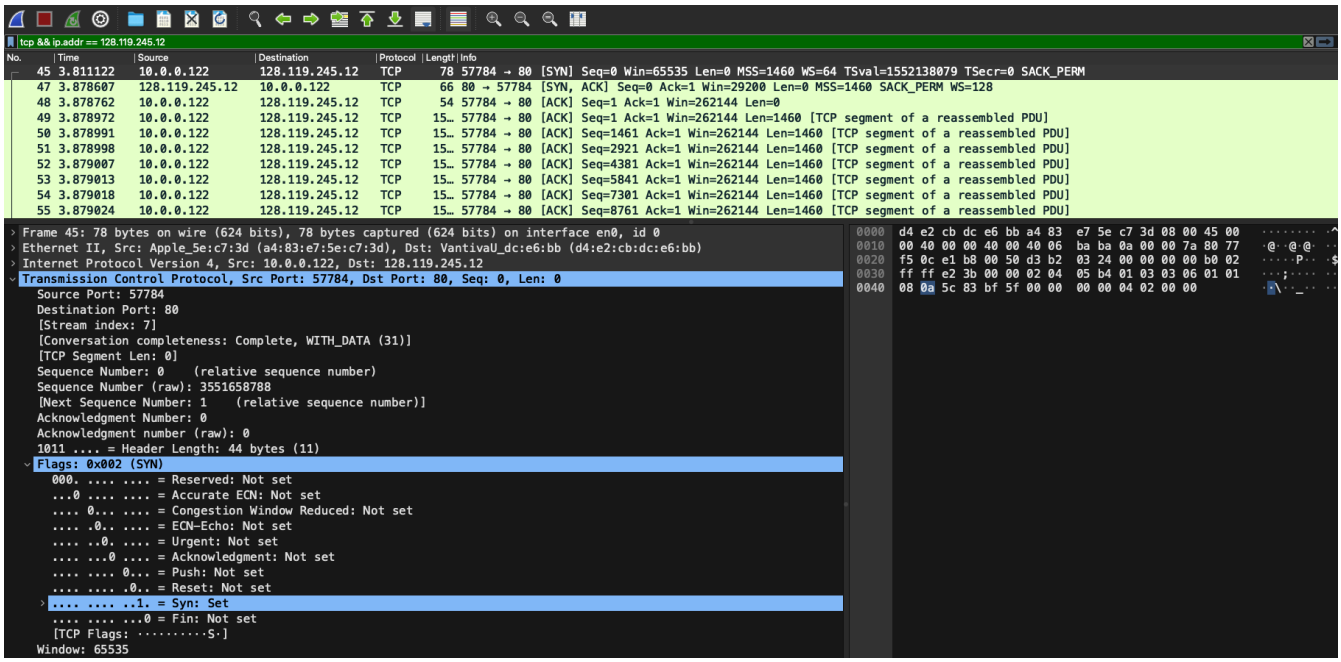
   TCP port number = 57784

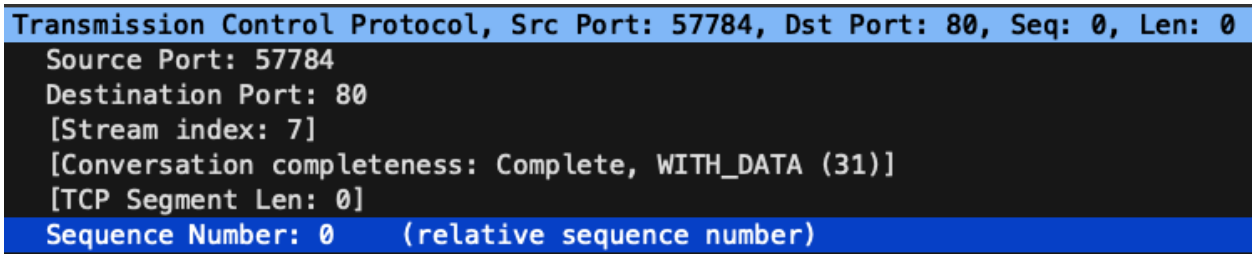


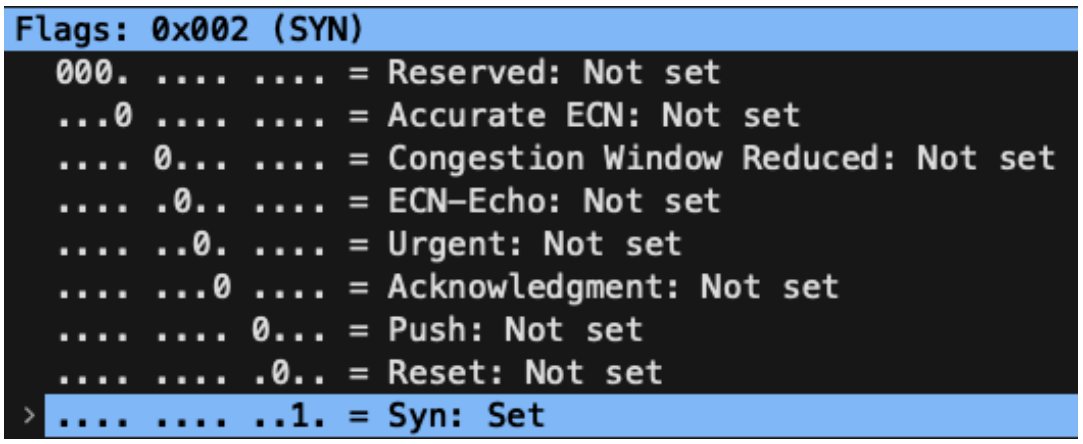
2. **What is the sequence number of the TCP SYN segment used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?**

The sequence number of the TCP SYN segment used to initiate the TCP connection between the client computer and gaia.cs.umass.edu is 0.

The SYN Flag is in the segment and determines if it is a SYN segment. The SYN Flag is 1, showcasing that this segment is a SYN segment.




Transmission Control Protocol, Src Port: 57784, Dst Port: 80, Seq: 0, Len: 0
Source Port: 57784
Destination Port: 80
[Stream index: 7]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)


Flags: 0x002 (SYN)
000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
.... 0... .... = Congestion Window Reduced: Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....0 .... = Acknowledgment: Not set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....1. = Syn: Set

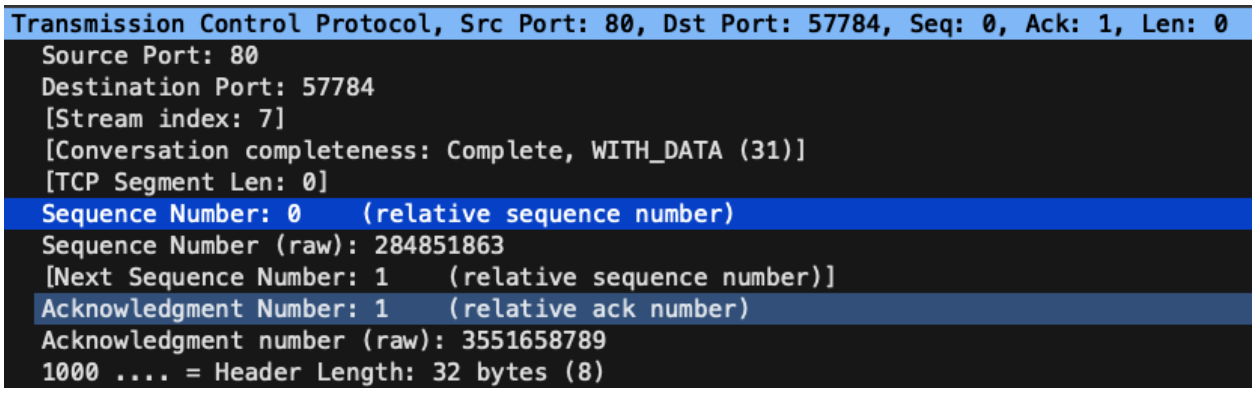
3. **What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgment field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a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

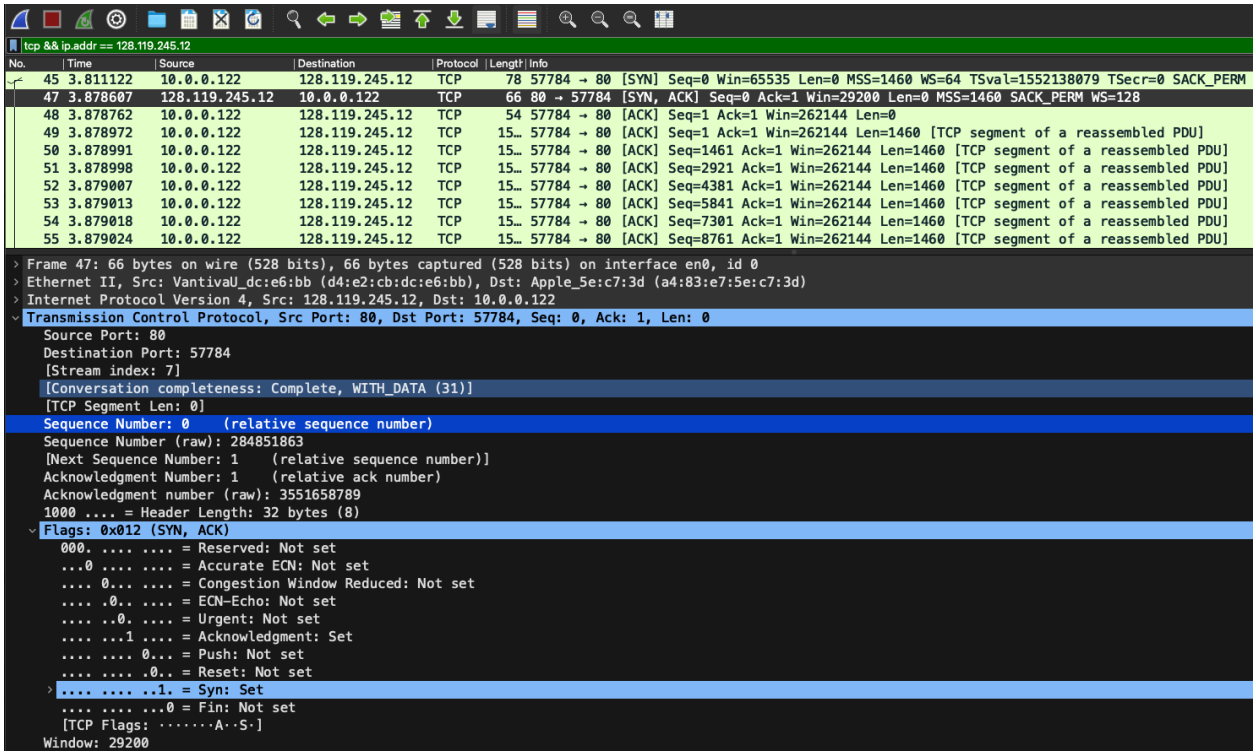
The value of the Acknowledgment field in the SYNACK segment is 1.

In terms of how gaia.cs.umass.edu determined the value, it calculates the ACK number for the SYN-ACK response by adding 1 to the initial sequence number, which the client computer used in its SYN segment. Since the client's sequence number is 0, adding 1 will result in the server's ACK number to be 1.

The SYN Flag along with the Acknowledgement is in the segment and determines if it is a SYNACK segment. The SYN Flag and the ACK Flag are 1, showcasing that this segment is a SYN segment.



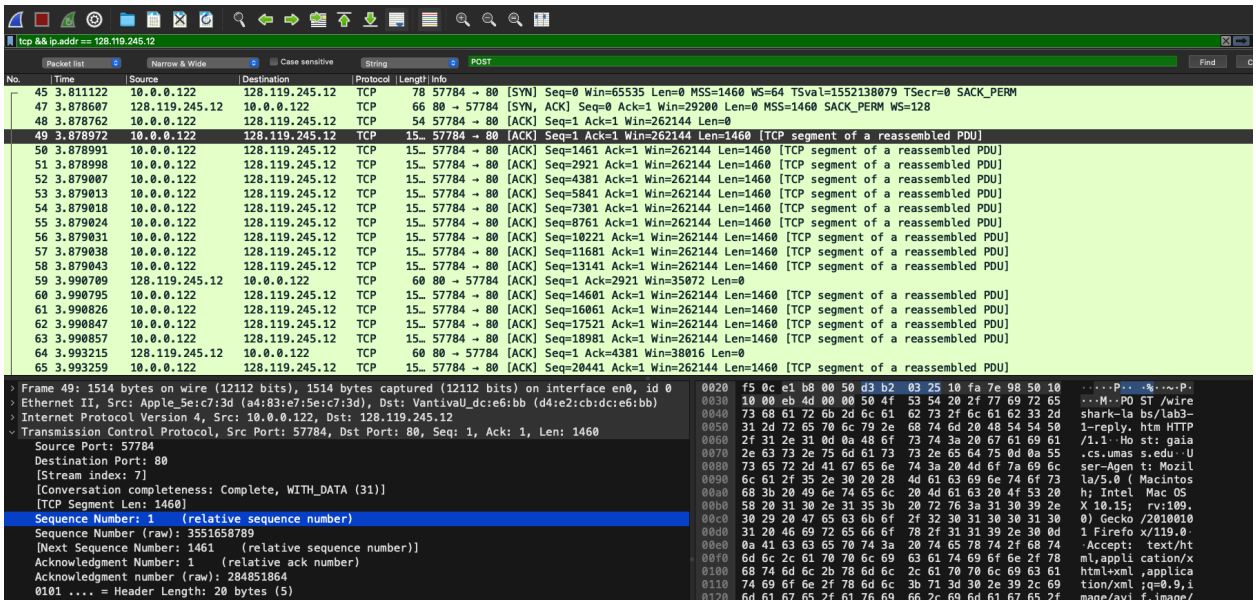
```
Flags: 0x012 (SYN, ACK)
    000. .... .... = Reserved: Not set
    ...0 .... .... = Accurate ECN: Not set
    .... 0... .... = Congestion Window Reduced: Not set
    .... .0.. .... = ECN-Echo: Not set
    .... ..0. .... = Urgent: Not set
    .... ...1 .... = Acknowledgment: Set
    .... .... 0... = Push: Not set
    .... .... .0.. = Reset: Not set
  > .... .... ..1. = Syn: Set
    .... .... ...0 = Fin: Not set
    [TCP Flags: ·······A··S·]
```

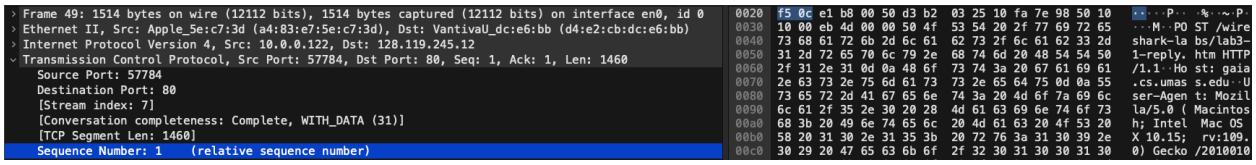


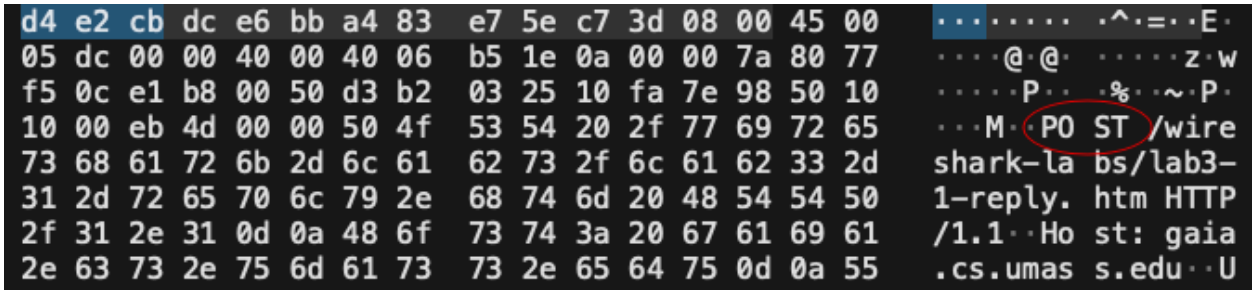
4. **What is the sequence number of the TCP segment containing the HTTP POST command?**

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

Looking into the packet content field in the bottom right, we can see the segment where it says “POST” in its data field. To find the HTTP POST command within a TCP segment, you would need to look in the packet data itself.






d4 e2 cb dc e6 bb a4 83 e7 5e c7 3d 08 00 45 00 ···· ·^·=··E·
05 dc 00 00 40 00 40 06 b5 1e 0a 00 00 7a 80 77 ····@·@· ·····z·w
f5 0c e1 b8 00 50 d3 b2 03 25 10 fa 7e 98 50 10 ·····P·· ·%··~·P·
10 00 eb 4d 00 00 50 4f 53 54 20 2f 77 69 72 65 ···M··PO ST /wire
73 68 61 72 6b 2d 6c 61 62 73 2f 6c 61 62 33 2d shark-la bs/lab3-
31 2d 72 65 70 6c 79 2e 68 74 6d 20 48 54 54 50 1-reply. htm HTTP
2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 67 61 69 61 /1.1··Ho st: gaia
2e 63 73 2e 75 6d 61 73 73 2e 65 64 75 0d 0a 55 .cs.umas s.edu··U

5. **Identify the 3 TCP ACKs from the server, and find their corresponding TCP data segments from the client. Explain how you found these 3 TCP data segments. At what time was each TCP data segment sent by the client? When was each TCP ACK received? Given the difference, what are the RTT values for these 3 TCP segments?**

   (*Note:* Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. You can use it to compare with your own calculation. 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.)*

   3 TCP ACKs from the server are frames No. 59, 64, and 67.

```
59 3.990709    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=2921 Win=35072 Len=0

64 3.993215    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=4381 Win=38016 Len=0

67 3.994257    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=5841 Win=40960 Len=0
```

   Corresponding TCP data segments from the client are frames No. 50, 51, and 52.

```
50 3.878991    10.0.0.122       128.119.245.12   TCP    15… 57784 → 80 [ACK] Seq=1461 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
51 3.878998    10.0.0.122       128.119.245.12   TCP    15… 57784 → 80 [ACK] Seq=2921 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
52 3.879007    10.0.0.122       128.119.245.12   TCP    15… 57784 → 80 [ACK] Seq=4381 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
```

   Explanation of how I found the 3 TCP data Segments.

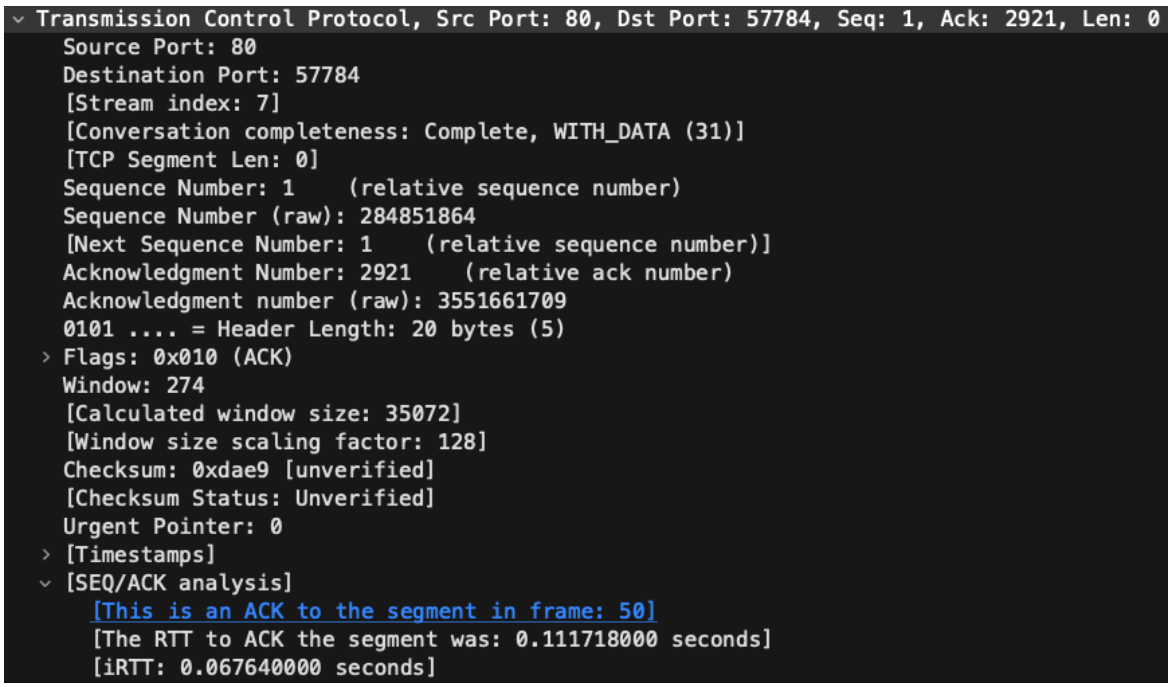
For example, frame 59 TCP ACK from the server has an acknowledgment number of 2921. Doing the calculation, 2921 - 1 = 2920. This means it is confirming receipt of all data up to and including the byte 2920 and is now expecting the next byte, which would be byte 2921. When we look at frame 50, it has a sequence number of 1461 and carries 1460 bytes of data. This means it contains bytes numbered 1461 to 2920 (1461 + 1460 - 1) of the data stream. So frame

59 is directly acknowledging the data sent in segment frame 50. Similar calculations go for Frame 64 and 67.

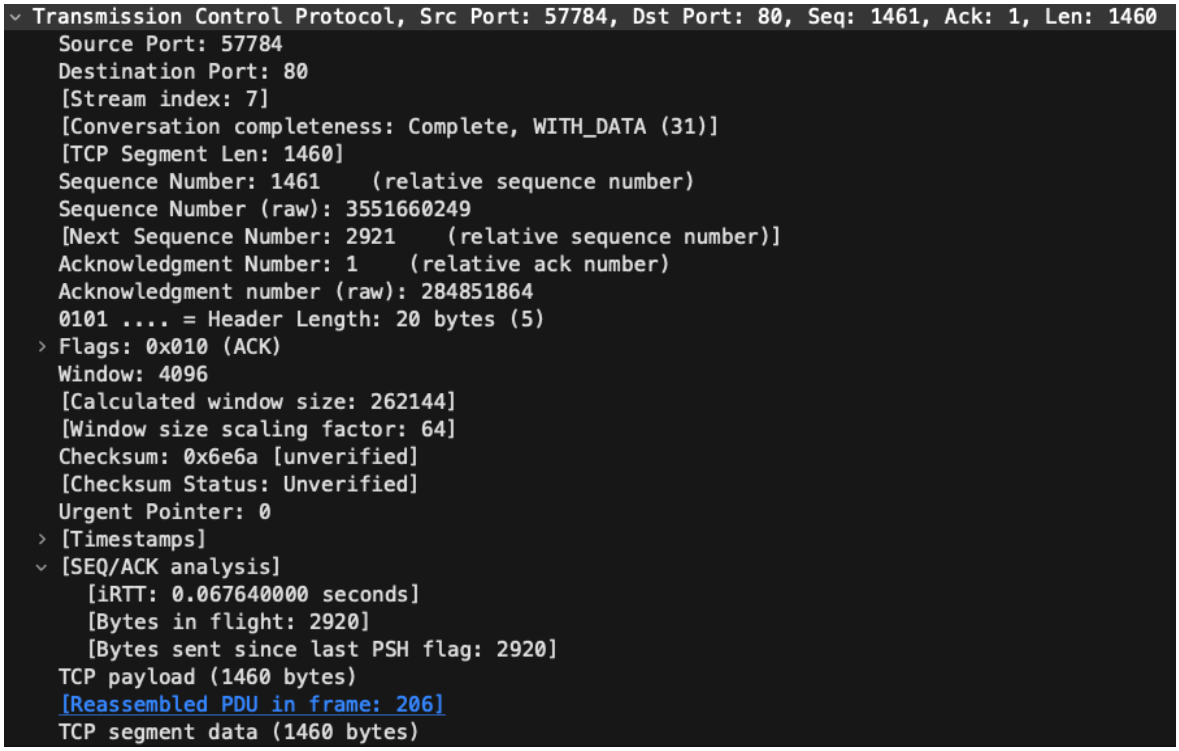
For Frame 64, it has an acknowledgment number of 4381. It confirms the receipt of all data up to and including byte 4380, and it's now expected byte would be 4381. If you look at the previous segment sent from the client that was not yet acknowledged, Frame 51, you would expect to find that Frame 51 sent data up to byte 4380.

For Frame 67, it has an acknowledgment number of 5841. It confirms the receipt of all data up to and including byte 5840, and it's now expected byte would be 5841. Again, looking back at the previously unacknowledged data sent from the client, Frame 52, you would expect Frame 52 to have sent data up to byte 5840.

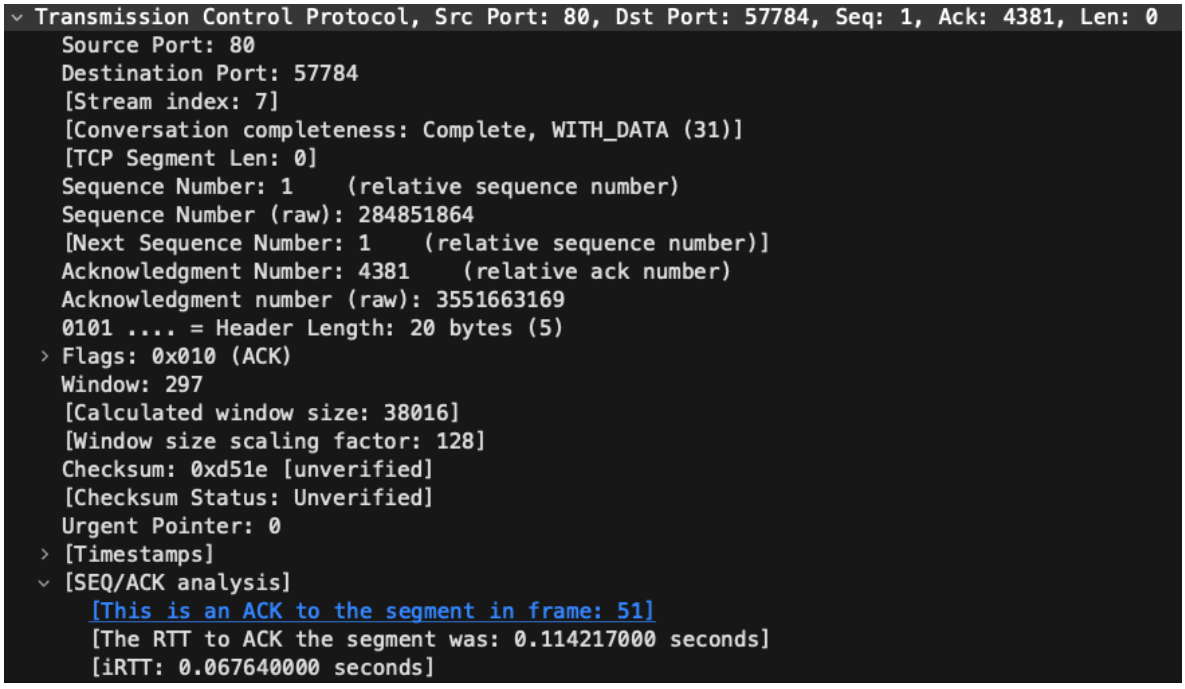
Frame 59:



Frame 50:



Frame 64:



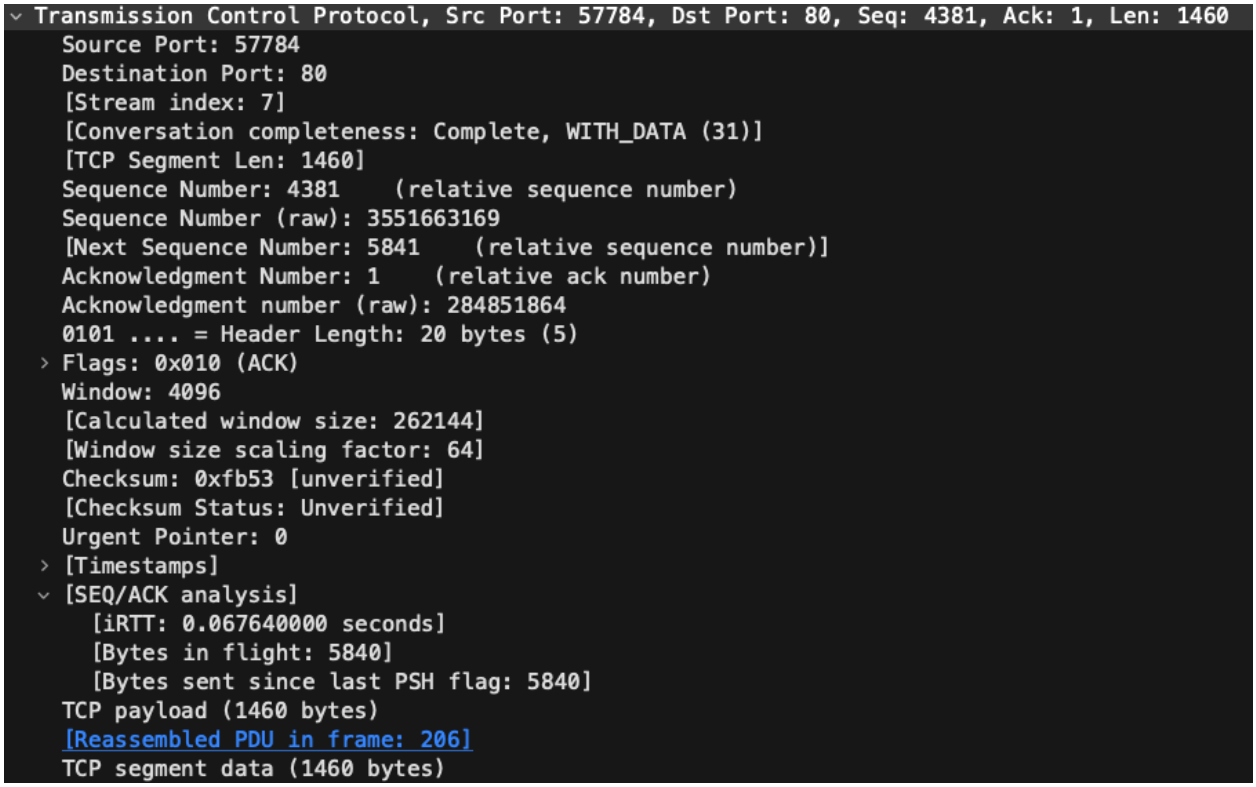
Frame 51:

```
Transmission Control Protocol, Src Port: 57784, Dst Port: 80, Seq: 2921, Ack: 1, Len: 1460
    Source Port: 57784
    Destination Port: 80
    [Stream index: 7]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 1460]
    Sequence Number: 2921    (relative sequence number)
    Sequence Number (raw): 3551661709
    [Next Sequence Number: 4381    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 284851864
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 4096
    [Calculated window size: 262144]
    [Window size scaling factor: 64]
    Checksum: 0x85ca [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  v [SEQ/ACK analysis]
      [iRTT: 0.067640000 seconds]
      [Bytes in flight: 4380]
      [Bytes sent since last PSH flag: 4380]
    TCP payload (1460 bytes)
    [Reassembled PDU in frame: 206]
    TCP segment data (1460 bytes)
```

Frame 67:

```
Transmission Control Protocol, Src Port: 80, Dst Port: 57784, Seq: 1, Ack: 5841, Len: 0
    Source Port: 80
    Destination Port: 57784
    [Stream index: 7]
    [Conversation completeness: Complete, WITH_DATA (31)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 284851864
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 5841    (relative ack number)
    Acknowledgment number (raw): 3551664629
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x010 (ACK)
    Window: 320
    [Calculated window size: 40960]
    [Window size scaling factor: 128]
    Checksum: 0xcf53 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > [Timestamps]
  v [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 52]
      [The RTT to ACK the segment was: 0.115250000 seconds]
      [iRTT: 0.067640000 seconds]
```

Frame 50:



The statement, "This is an ACK to the segment in Frame: No." also gives a direct reference to the frame that the acknowledgment is for.

At what time each TCP data segment was sent by the client, when was each TCP ACK received, and given the difference, the RTT values for the 3 TCP segments are all showcased below:

Segments & Frames	Time TCP data segment sent by the client	Time TCP ACK was received	RTT
Segment 1 [Frames 50, 59]	3.878991	3.990709	0.111718
Segment 2 [Frames 51, 64]	3.878998	3.993215	0.114217
Segment 3 [Frames 52, 67]	3.879007	3.994307	0.11525

How to Calculate RTT
RTT = ACK received time - Segment sent time
Seg 1 RTT = 3.990709 - 3.878991 = 0.111718 sec
Seg 2 RTT = 3.993215 - 3.878998 = 0.114217 sec
Seg 3 RTT = 3.994257 - 3.879007 = 0.115250 sec

```
50 3.878991    10.0.0.122       128.119.245.12   TCP     15… 57784 → 80 [ACK] Seq=1461 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
51 3.878998    10.0.0.122       128.119.245.12   TCP     15… 57784 → 80 [ACK] Seq=2921 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
52 3.879007    10.0.0.122       128.119.245.12   TCP     15… 57784 → 80 [ACK] Seq=4381 Ack=1 Win=262144 Len=1460 [TCP segment of a reassembled PDU]
```

```
59 3.990709    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=2921 Win=35072 Len=0
```

```
64 3.993215    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=4381 Win=38016 Len=0
```

```
67 3.994257    128.119.245.12   10.0.0.122       TCP      60 80 → 57784 [ACK] Seq=1 Ack=5841 Win=40960 Len=0
```

6. **Among the 3 TCP ACKs you identified in the previous question, 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?**

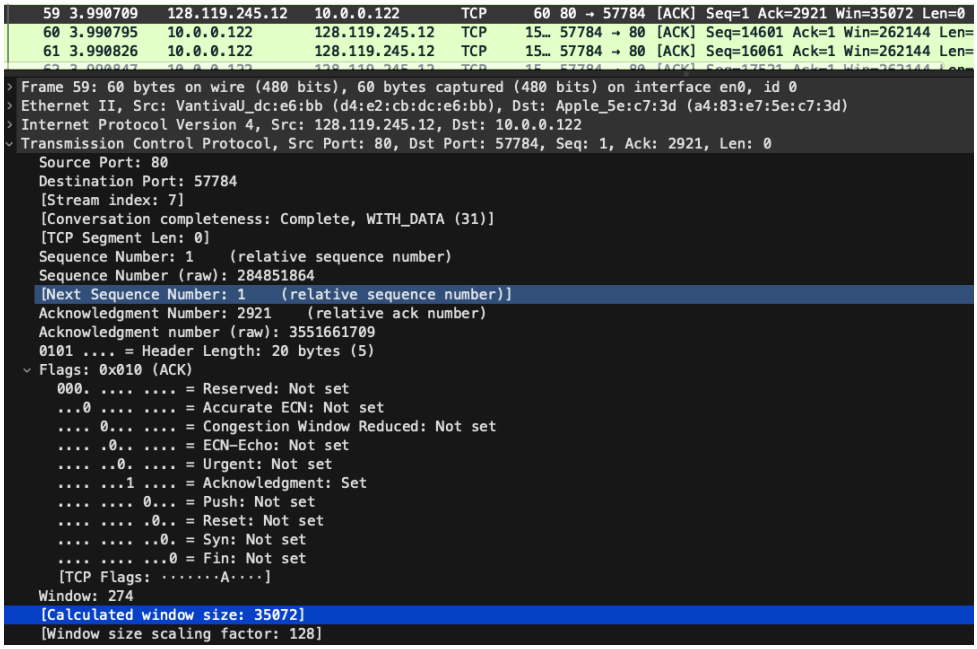
Among the 3 TCP ACKs, the minimum amount of available buffer space advertised at the received for the entire trace are:

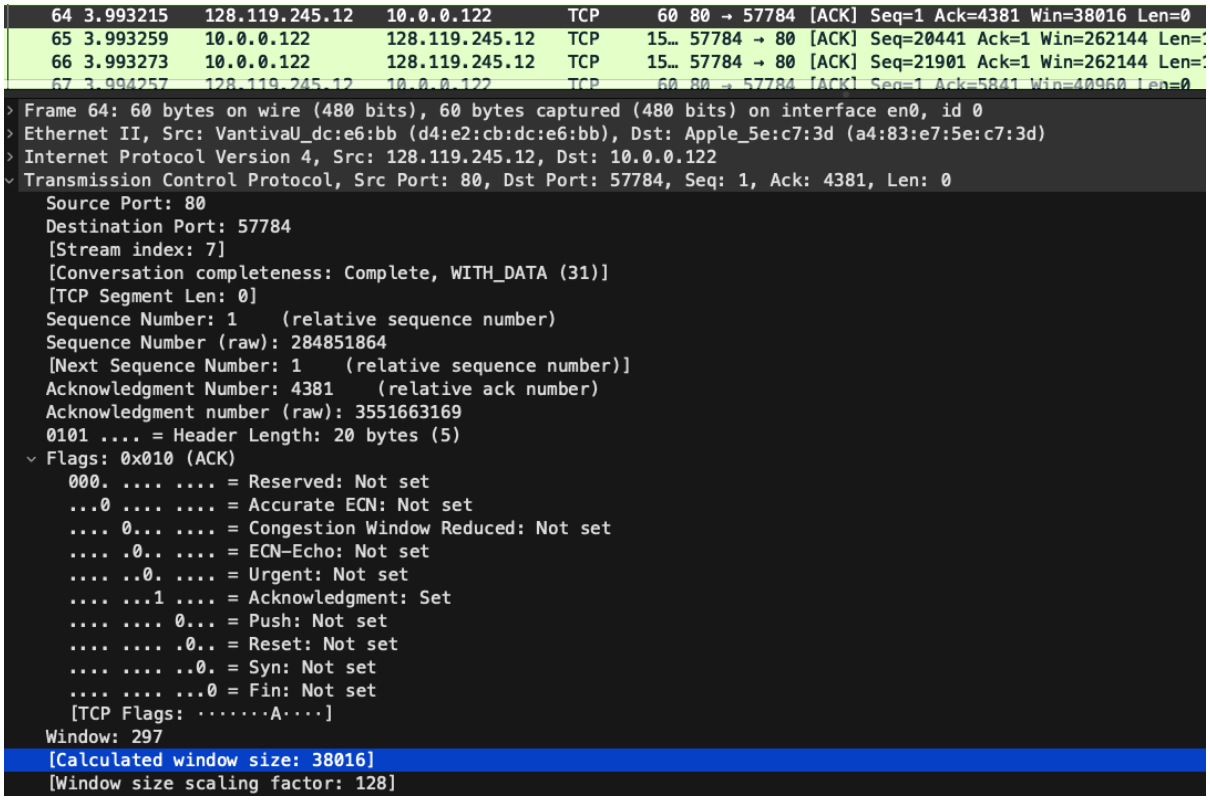
For 1st TCP ACK (Frame 59): 35072 bytes

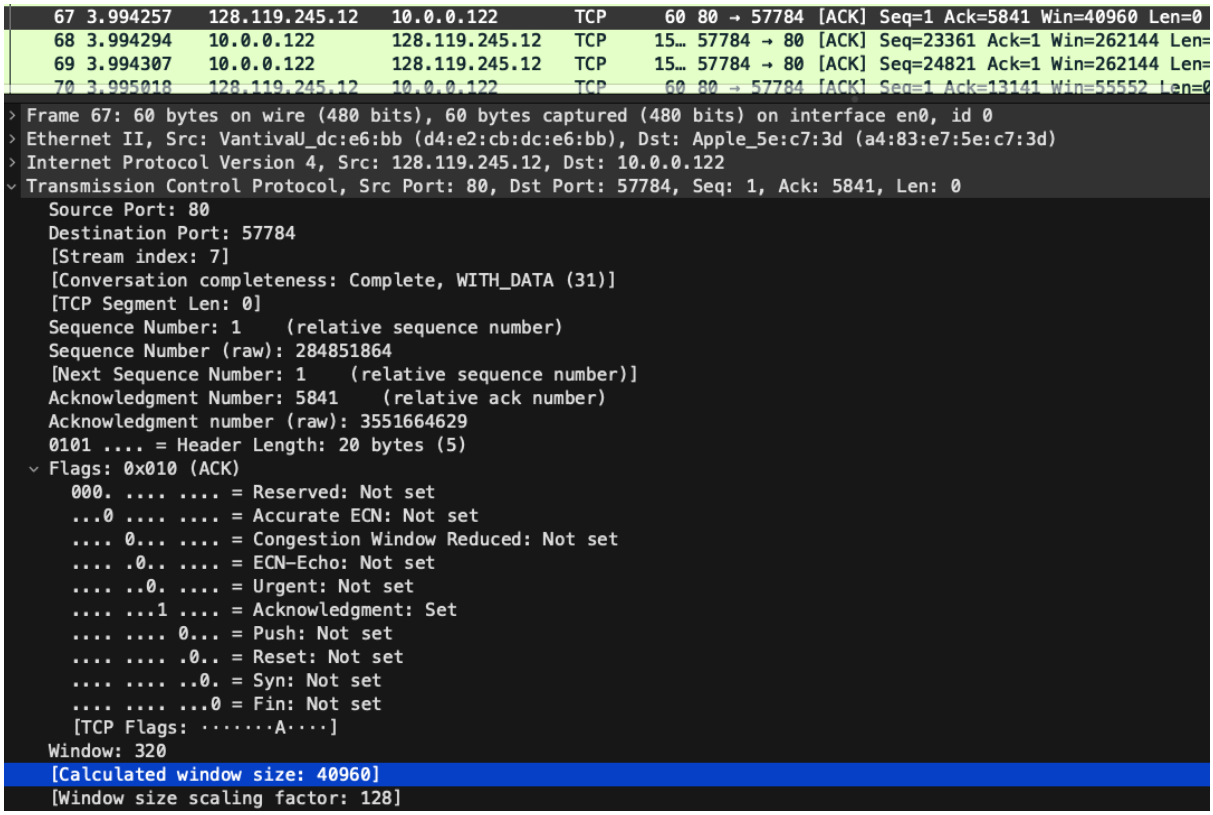
For 2nd TCP ACK (Frame 64): 38016 bytes

For 3rd TCP ACK (Frame 67): 40960 bytes

As for if the lack of receiver buffer space ever throttles the sender, based on the TCP ACK packets info, the receiver's window size is either the same or increasing until it reaches a certain max buffer size. So it's an indication that the sender will not be throttled due to a lack of receiver buffer space.



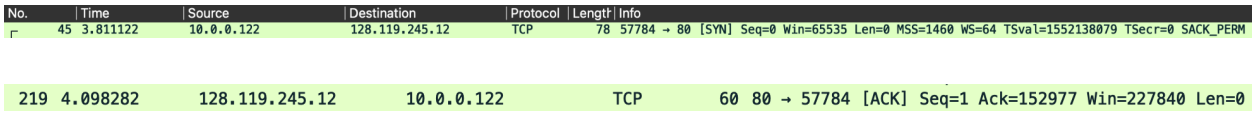

64 3.993215 128.119.245.12 10.0.0.122 TCP 60 80 → 57784 [ACK] Seq=1 Ack=4381 Win=38016 Len=0
65 3.993259 10.0.0.122 128.119.245.12 TCP 15… 57784 → 80 [ACK] Seq=20441 Ack=1 Win=262144 Len=
66 3.993273 10.0.0.122 128.119.245.12 TCP 15… 57784 → 80 [ACK] Seq=21901 Ack=1 Win=262144 Len=
67 3.994257 128.119.245.12 10.0.0.122 TCP 60 80 → 57784 [ACK] Seq=1 Ack=5841 Win=40960 Len=0
Frame 64: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface en0, id 0
Ethernet II, Src: VantivaU_dc:e6:bb (d4:e2:cb:dc:e6:bb), Dst: Apple_5e:c7:3d (a4:83:e7:5e:c7:3d)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.0.0.122
Transmission Control Protocol, Src Port: 80, Dst Port: 57784, Seq: 1, Ack: 4381, Len: 0
Source Port: 80
Destination Port: 57784
[Stream index: 7]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 284851864
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 4381 (relative ack number)
Acknowledgment number (raw): 3551663169
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
.... 0... .... = Congestion Window Reduced: Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....0 = Fin: Not set
[TCP Flags: ·····A····]
Window: 297
[Calculated window size: 38016]
[Window size scaling factor: 128]


67 3.994257 128.119.245.12 10.0.0.122 TCP 60 80 → 57784 [ACK] Seq=1 Ack=5841 Win=40960 Len=0
68 3.994294 10.0.0.122 128.119.245.12 TCP 15… 57784 → 80 [ACK] Seq=23361 Ack=1 Win=262144 Len=
69 3.994307 10.0.0.122 128.119.245.12 TCP 15… 57784 → 80 [ACK] Seq=24821 Ack=1 Win=262144 Len=
70 3.995018 128.119.245.12 10.0.0.122 TCP 60 80 → 57784 [ACK] Seq=1 Ack=13141 Win=55552 Len=0
Frame 67: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface en0, id 0
Ethernet II, Src: VantivaU_dc:e6:bb (d4:e2:cb:dc:e6:bb), Dst: Apple_5e:c7:3d (a4:83:e7:5e:c7:3d)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 10.0.0.122
Transmission Control Protocol, Src Port: 80, Dst Port: 57784, Seq: 1, Ack: 5841, Len: 0
Source Port: 80
Destination Port: 57784
[Stream index: 7]
[Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 0]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 284851864
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 5841 (relative ack number)
Acknowledgment number (raw): 3551664629
0101 .... = Header Length: 20 bytes (5)
Flags: 0x010 (ACK)
000. .... = Reserved: Not set
...0 .... = Accurate ECN: Not set
.... 0... .... = Congestion Window Reduced: Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
....1 .... = Acknowledgment: Set
.... 0... = Push: Not set
.... .0.. = Reset: Not set
....0. = Syn: Not set
....0 = Fin: Not set
[TCP Flags: ·····A····]
Window: 320
[Calculated window size: 40960]
[Window size scaling factor: 128]

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

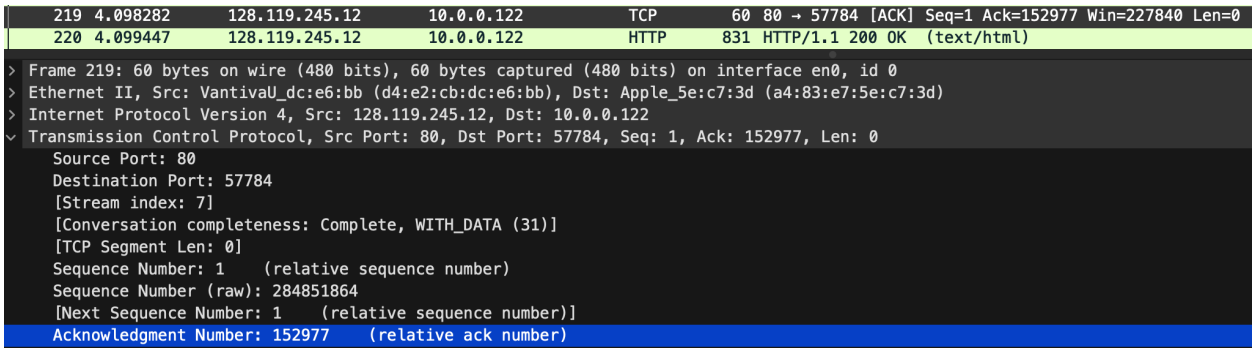
To calculate the throughput (bytes transferred per unit time) for the TCP connection, we need to determine the total amount of data transferred.

- Find the first packet in the file transfer, which is Frame 45
- Find the final Ack from the server, which is Frame 219



To figure out the total amount of data transferred, we can take a look at the Ack when the payload is Len = 0. So in this scenario, the Ack is equal to 152,977 Bytes. Since the Len = 0 when the Seq = 1 at the start of the session, we can simply see that the bytes transferred is 152,977 – 1 = 152,976 Bytes.

[The Total Num of Bytes = Final Ack - Initial Seq]



Now, we need to calculate the total transmission time.

- Finding the timestamp of the first TCP segment that contains data.
- Finding the timestamp of the last ACK segment.
- Then, calculate the total transmission time by taking the difference between these two timestamps.

Time(Frame 219) - Time(Frame 45) = Total Time Transmission

4.098282 - 3.811122 = 0.28716 seconds

Finally, to calculate the throughput, you need to divide the total amount of data transferred by the total transmission time. This will result in the TCP throughput using sequence numbers.

The total amount of data transferred = 152,976 bytes

The total transmission time = 0.28716 seconds

Throughput = Total amount of data transferred / Total transmission time

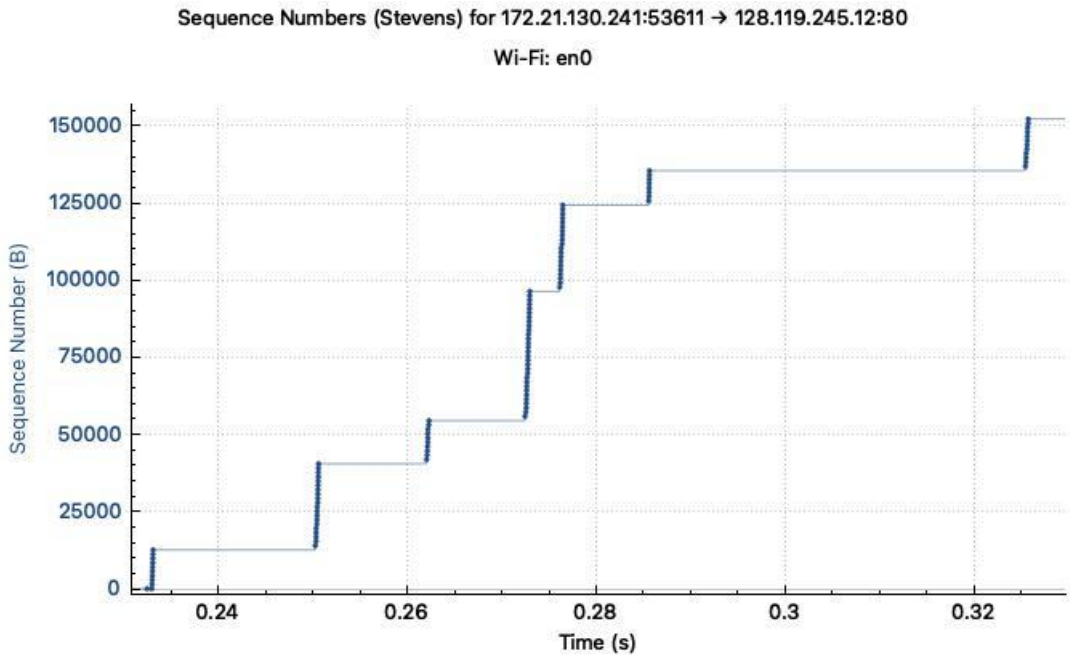
152,976 bytes / 0.28716 seconds = approximately 532,720 bytes/second

(Exactly 532,720.434601 bytes/seconds)

Conversion from Bps to KBps: 532,720 / 1024 = 520.23

So the throughput for this TCP connection is approximately 520 KBps

8. **Use the *Time-Sequence-Graph(Stevens)* plotting tool to view the sequence number versus the time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where is TCP's slow start phase and congestion avoidance phase? Can you identify any congestion?**



In order to identify where the TCP's slow start phase begins, we need to analyze the time plot to see the sequence number increasing exponentially, which is usually present at the beginning of the TCP connection. So we can make an inference that the slow start of the TCP approximately starts around 0.2325, because it is between 0.23 and 0.235, and ends approximately around 0.25, which is between 0.24 and 0.26.

As for Congestion avoidance, it usually starts after when the slow start threshold is reached. Instead of increasing exponentially, the increase in sequence numbers is more linear. We could guess that it starts at approximately around 0.2725 since it's between 0.27 and 0.275 because the amount of data being sent starts to reduce.