Student Information

Full Name : Mehmet Rüçhan Yavuzdemir
Id Number: 2522159

1 A first look at the captured trace

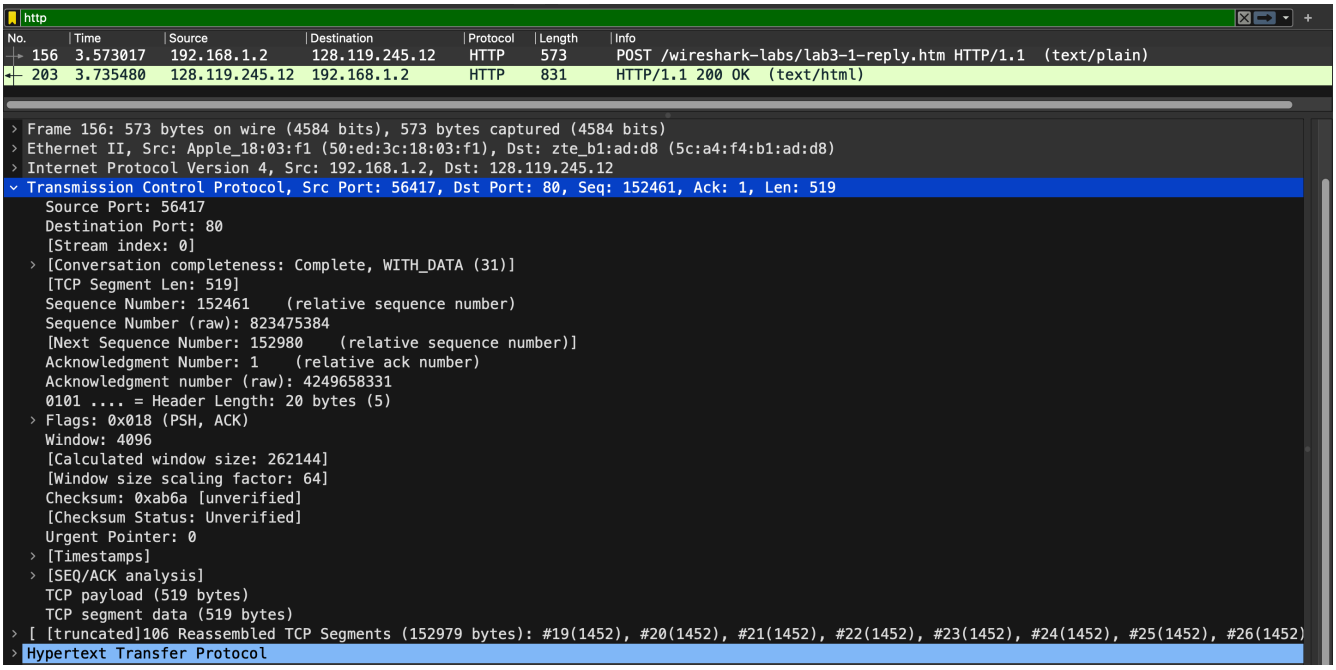


Figure 1: Answer 1-2

Answer 1.1

The source IP address is `192.168.1.2`, as shown in the figure, under the `Source` column in the packet listing window.

Answer 1.2

The source port number is `56417`, as shown in the figure under the TCP `Source Port` section.

Answer 2.1

The destination IP address is `128.119.245.12`, as shown in the figure, under the `Destination` column in the packet listing window.

Answer 2.2

The destination port number is `80`, as shown in the figure under the TCP `Destination Port` section. This is the port number from where the server is sending and receiving TCP segments for this connection.

2 TCP Basics

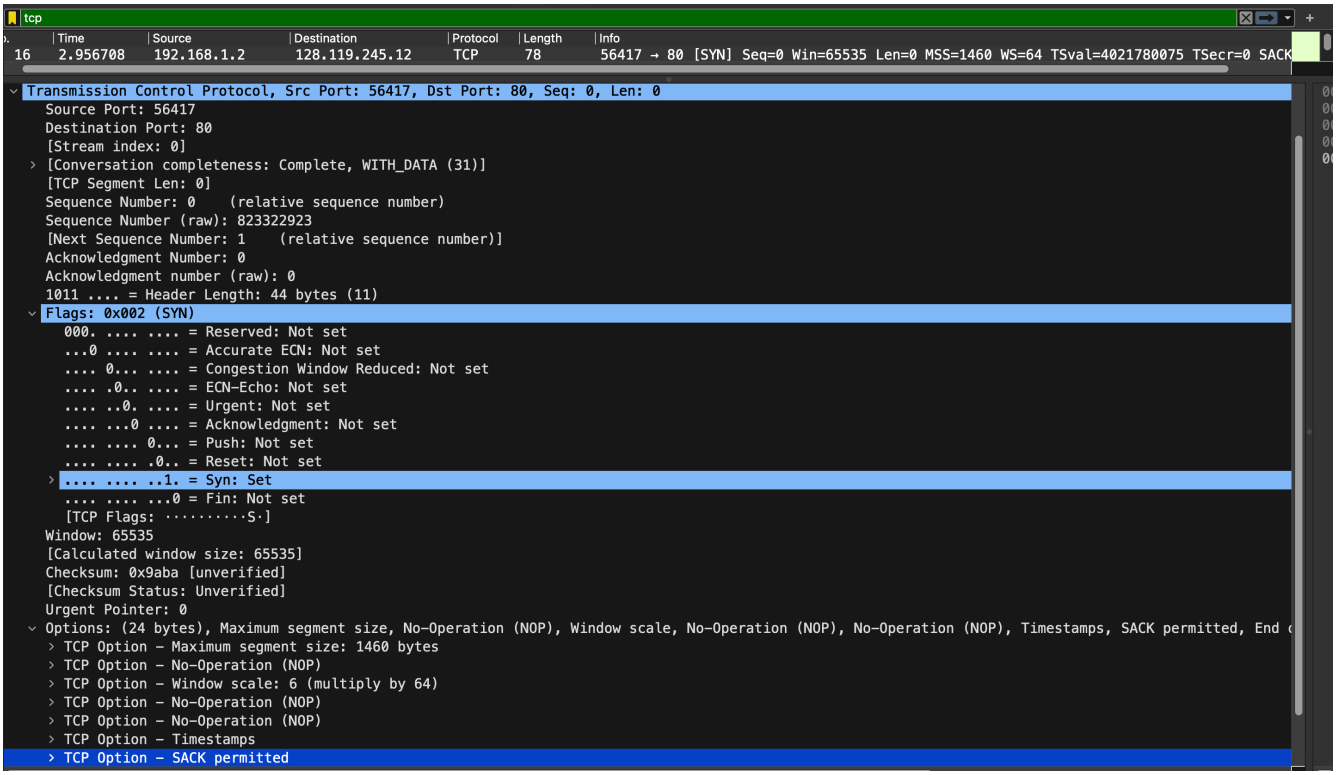


Figure 2: Answer 3

Answer 3.1

Under the TCP section, we have a field called `Sequence Number (raw)`. This is exactly what we are looking for, not a relative sequence number, not a packet number. The sequence number of the TCP `SYN` segment initiating the TCP connection between client and server is `823322923`.

Answer 3.2

For this purpose, the TCP `Flags` section is used. As highlighted in the figure, `Syn` bit is set to 1, indicating that this segment is a `SYN` segment.

Answer 3.3

In the TCP `Options` section, there is an option called `SACK permitted`. Hence, the TCP receiver will be able to use Selective Acknowledgments, so-called `SACK`s.

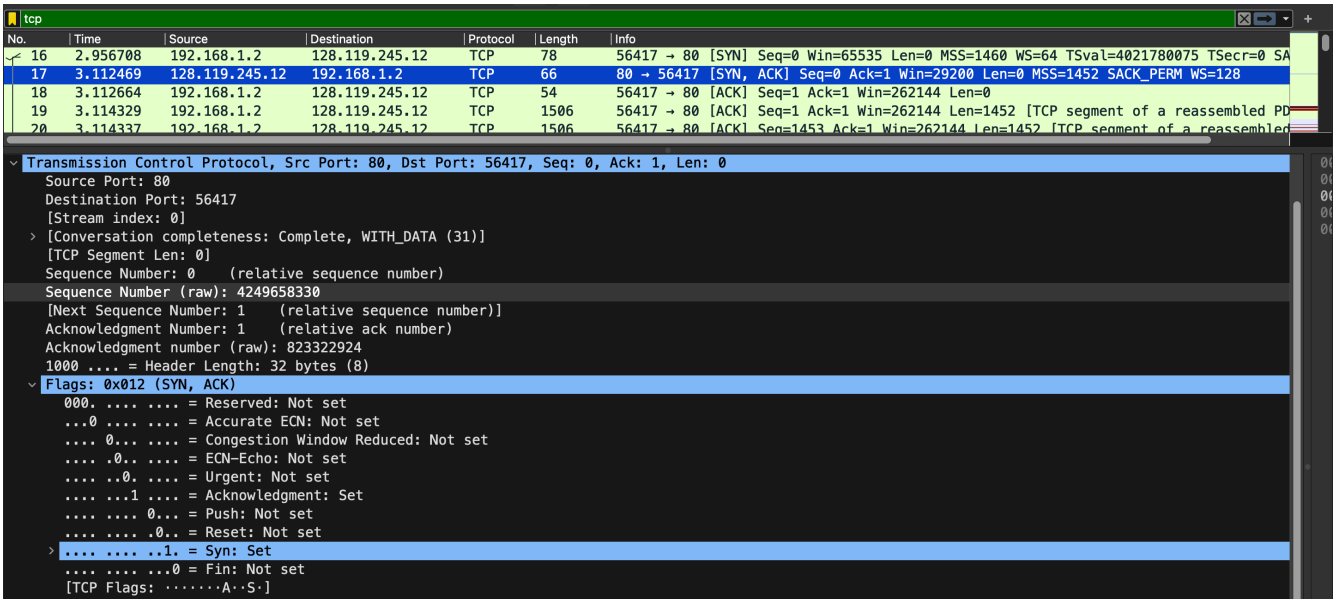


Figure 3: Answer 4

Answer 4.1

Again, under the TCP section, we have a field called `Sequence Number (raw)`. This is exactly what we are looking for, not a relative sequence number, not a packet number. The sequence number of the TCP `SYN` sent by the server to the client is `4249658330`.

Answer 4.2

For this purpose, the TCP `Flags` section is used. In the figure above, `Acknowledgment` and `Syn` bit is set to 1, indicating that this segment is a `SYNACK` segment.

Answer 4.3

The value of `Acknowledgment` field in the `SYNACK` segment is `823322924`.

Answer 4.4

The value of the `Acknowledgment` field in the `SYNACK` segment is the sequence number of the next expected data to be received at the server from the client, and in this case, one higher than the sequence number in the initial `SYN` segment sent from the client to server.

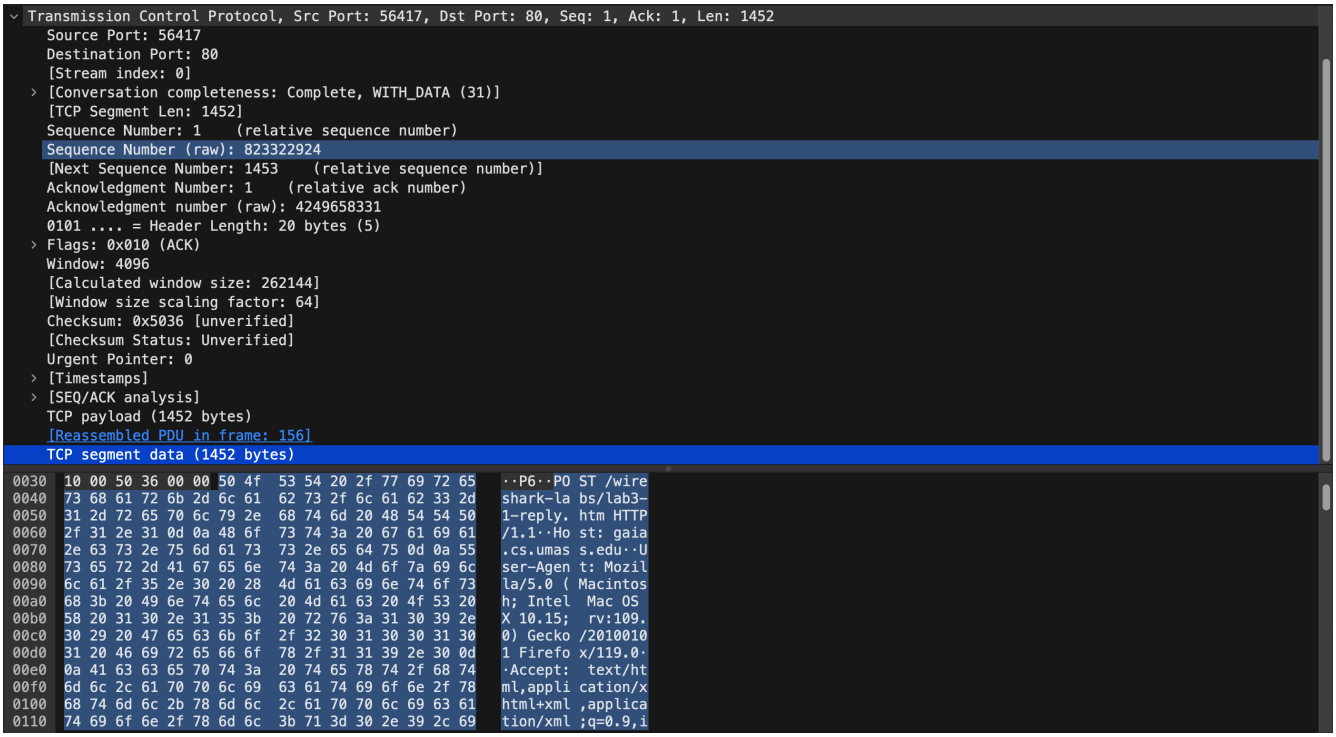


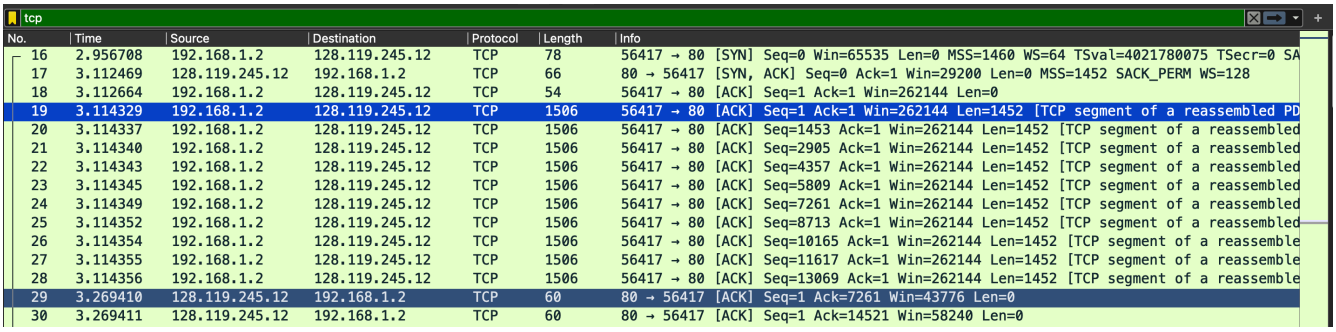
Figure 4: Answer 5

Answer 5.1

I found the TCP segment containing the header of the HTTP POST request, and the beginning of the `alice.txt` file (it did not fit into the screen though). In the packet content window, you can see the real HTTP POST header information in ASCII format. The sequence number can be retrieved from the highlighted TCP `Sequence Number (raw)` field, so the sequence number is `823322924`.

Answer 5.2

As highlighted in the figure above, the size of the payload(data) of this TCP segment is `1452` bytes. Of course, the whole `alice.txt` file did not fit into one segment, not even close! Remember from Figure 1 that there were 106 TCP segments, so there is a lot to transmit!



tcp

No.	Time	Source	Destination	Protocol	Length	Info
16	2.956708	192.168.1.2	128.119.245.12	TCP	78	56417 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=4021780075 TSecr=0 SA
17	3.112469	128.119.245.12	192.168.1.2	TCP	66	80 → 56417 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1452 SACK_PERM WS=128
18	3.112664	192.168.1.2	128.119.245.12	TCP	54	56417 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
19	3.114329	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled PD
20	3.114337	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=1453 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
21	3.114340	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=2905 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
22	3.114343	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=4357 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
23	3.114345	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=5809 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
24	3.114349	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=7261 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
25	3.114352	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=8713 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
26	3.114354	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=10165 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
27	3.114355	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=11617 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
28	3.114356	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=13069 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
29	3.269410	128.119.245.12	192.168.1.2	TCP	60	80 → 56417 [ACK] Seq=1 Ack=7261 Win=43776 Len=0
30	3.269411	128.119.245.12	192.168.1.2	TCP	60	80 → 56417 [ACK] Seq=1 Ack=14521 Win=58240 Len=0

Figure 5: Answer 6

Answer 6.1

As highlighted in the figure above, the first TCP segment containing the HTTP POST request in the data-transfer part of the TCP connection was sent `3.114329` seconds after the packet sniffing started.

Answer 6.2

Again, as highlighted above, the first ACK from server to client for the first data-containing segment arrived at `3.269410` seconds after the packet sniffing started.

Answer 6.3

Since we solved the two questions above, finding RTT for the first data-containing segment is easy. RTT is basically how much time has passed to wait for the ACK for the data-containing segment. Using the values from the previous two questions, the RTT for the first data-containing segment is `3.269410 - 3.114329 = 0.155081` seconds.

Answer 6.4

For the second data-carrying TCP segment, there is no ACK segment for the second segment specifically sent by the server. Instead, the server sent cumulative ACK for this purpose, which can be understood by examining the window sizes. Therefore, we can approximate the RTT from the values we currently have. The client sent the second data-carrying TCP segment at the time `3.114337`, and its corresponding ACK segment was received at the time `3.269410`. Hence, the RTT for the second data-carrying TCP segment is `3.269410 - 3.114337 = 0.155073` seconds.

Answer 6.5

In this question, we are asked to find the `EstimatedRTT` after the ACK for the second data-carrying segment is received by the client. We assume that the initial value of the `EstimatedRTT` is equal to the measured RTT for the first segment, and its new value is calculated by the formula below, with $\alpha = 0.125$.

$$\texttt{EstimatedRTT} = (1\text{-}\alpha) \cdot \texttt{EstimatedRTT} + \alpha \cdot \texttt{SampleRTT}$$

If we plug the values we found from the answers 6.3 and 6.4 into the formula, where `EstimatedRTT` is the measured RTT for the first data-carrying segment, and `SampleRTT` is the measured RTT for the second data-carrying segment, we will find the `EstimatedRTT` after the client receives the ACK for the second data-carrying segment.

$$\texttt{EstimatedRTT} = (0.875) \cdot 0.155081 + (0.125) \cdot 0.155073 = 0.15508$$

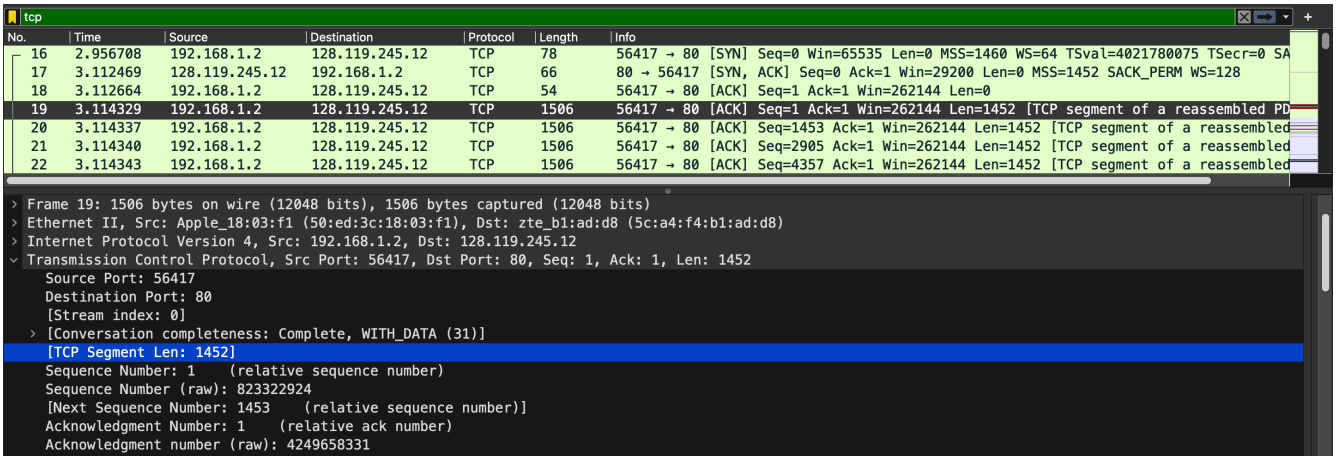


Figure 6: Answer 7

Answer 7

As highlighted in the figure above, the length of each of the first four data-carrying TCP segments is `1452` bytes.

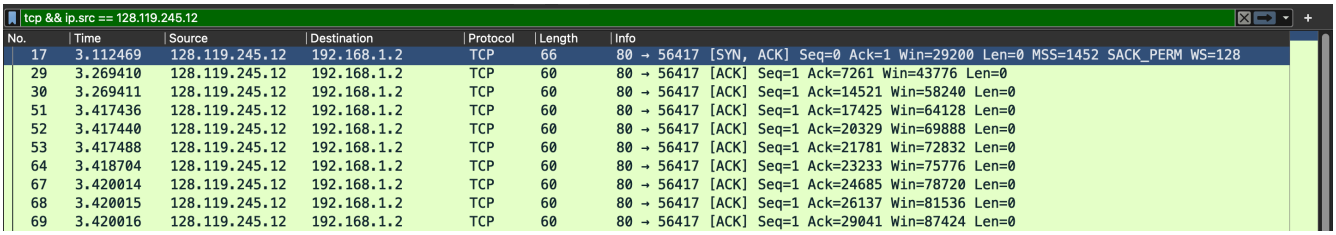


Figure 7: Answer 8

Answer 8.1

I filtered the TCP packets having the IP address of the server so that we can see how window size, which can also be used interchangeably with buffer space in this case, changes over time. As highlighted in the figure above, the initial and also minimum available buffer space provided by the server to the client is 29200 bytes. It increases over time. The window size indicated in the SYNACK segment is for the first data-carrying TCP segment, hence the value we are looking for is again 29200.

Answer 8.2

No, actually, the reality is the exact opposite. The receiver buffer space is large, hence it does not throttle the sender for the first four data-carrying segments. It can fit almost 20 TCP segments at a time, no overflow occurred, and everything is fine. Furthermore, remember that the window size is growing over time. Therefore, no throttling occurred.

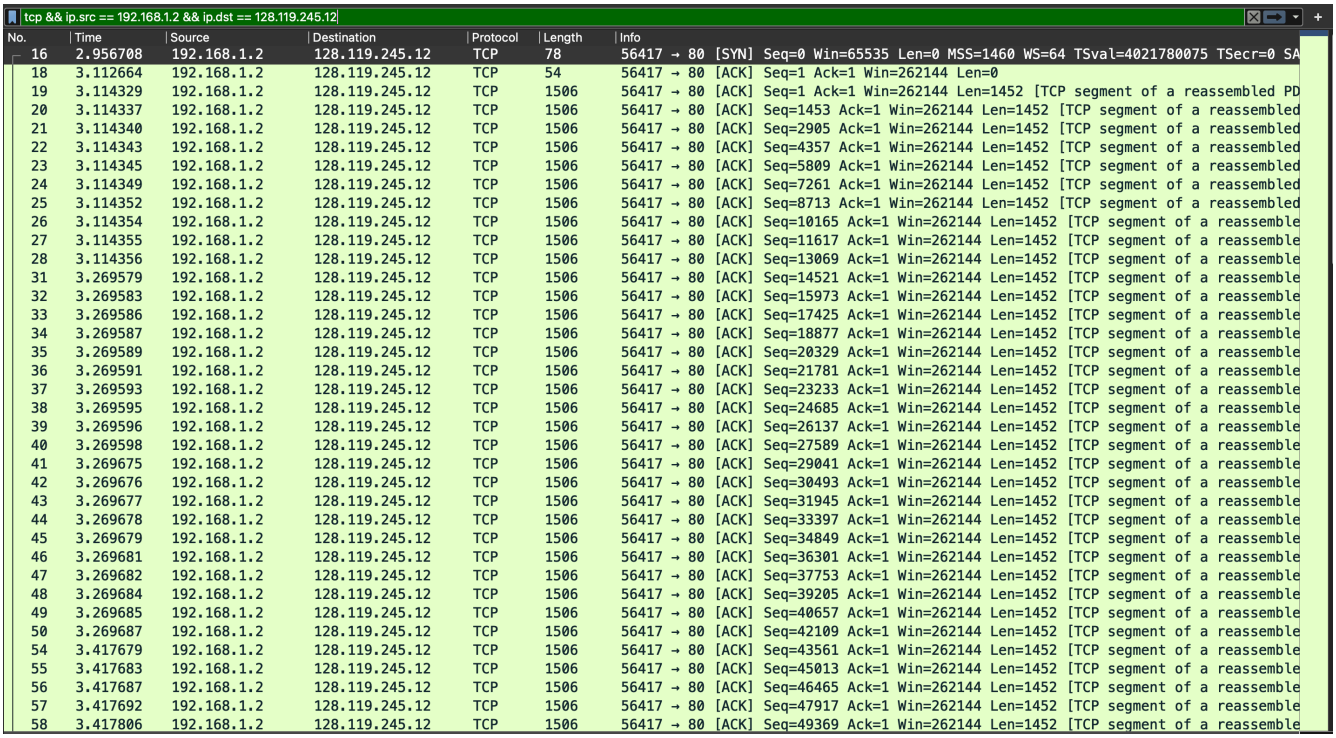


Figure 8: Answer 9

Answer 9.1

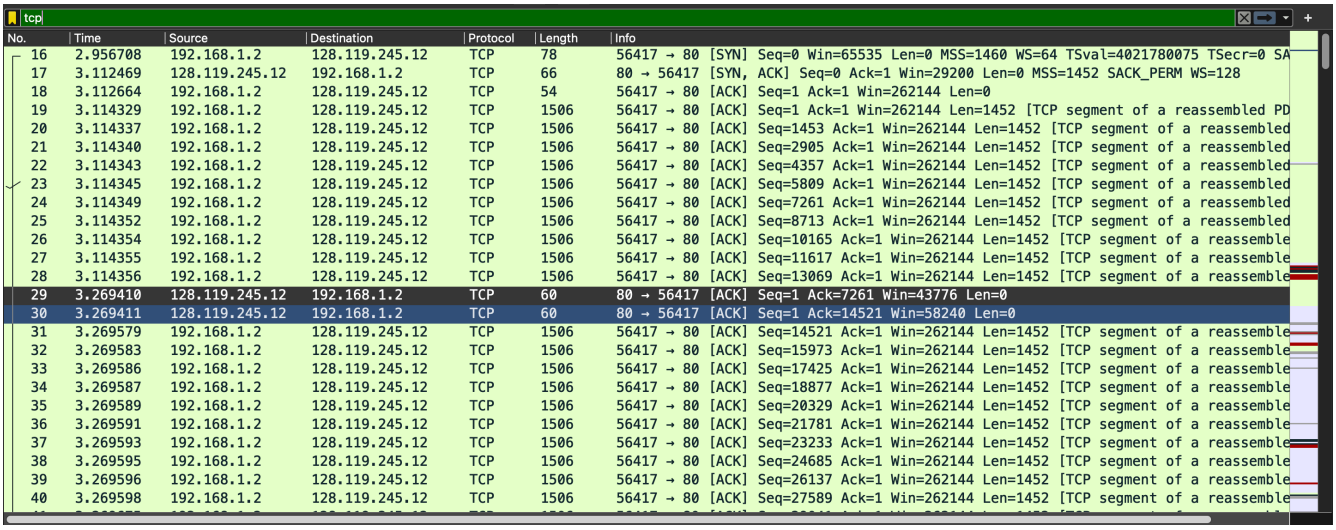
No, there are not any retransmitted segments in my trace file.

Answer 9.2

To be able to analyze the trace file, I used this filter query:

```
tcp && ip.src == 192.168.1.2 && ip.dst == 128.119.245.12
```

Since the sequence numbers are strictly increasing, there is no retransmission happened.



tcp

No.	Time	Source	Destination	Protocol	Length	Info
16	2.956708	192.168.1.2	128.119.245.12	TCP	78	56417 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=4021780075 TSecr=0 SA
17	3.112469	128.119.245.12	192.168.1.2	TCP	66	80 → 56417 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1452 SACK_PERM WS=128
18	3.112664	192.168.1.2	128.119.245.12	TCP	54	56417 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
19	3.114329	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled PD
20	3.114337	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=1453 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
21	3.114340	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=2905 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
22	3.114343	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=4357 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
23	3.114345	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=5809 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
24	3.114349	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=7261 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
25	3.114352	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=8713 Ack=1 Win=262144 Len=1452 [TCP segment of a reassembled
26	3.114354	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=10165 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
27	3.114355	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=11617 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
28	3.114356	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=13069 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
29	3.269410	128.119.245.12	192.168.1.2	TCP	60	80 → 56417 [ACK] Seq=1 Ack=7261 Win=43776 Len=0
30	3.269411	128.119.245.12	192.168.1.2	TCP	60	80 → 56417 [ACK] Seq=1 Ack=14521 Win=58240 Len=0
31	3.269579	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=14521 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
32	3.269583	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=15973 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
33	3.269586	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=17425 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
34	3.269587	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=18877 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
35	3.269589	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=20329 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
36	3.269591	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=21781 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
37	3.269593	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=23233 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
38	3.269595	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=24685 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
39	3.269596	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=26137 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble
40	3.269598	192.168.1.2	128.119.245.12	TCP	1506	56417 → 80 [ACK] Seq=27589 Ack=1 Win=262144 Len=1452 [TCP segment of a reassemble

Figure 9: Answer 10

Answer 10.1

The receiver typically acknowledges 5 TCP segments of data at a time. For each of the 5 data-containing segments, it accumulates them and sends a cumulative ACK segment. As a result, the receiver typically acknowledges 7260 bytes of data at a time.

Answer 10.2

Yes! After the first ACK every remaining ACK acknowledges 5 segments' worth of payload data.

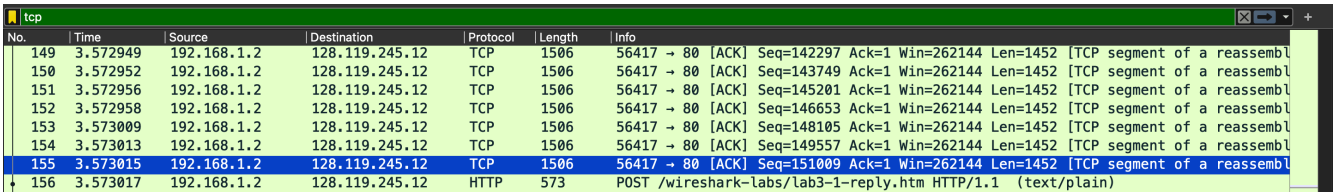


Figure 10: Answer 11-1

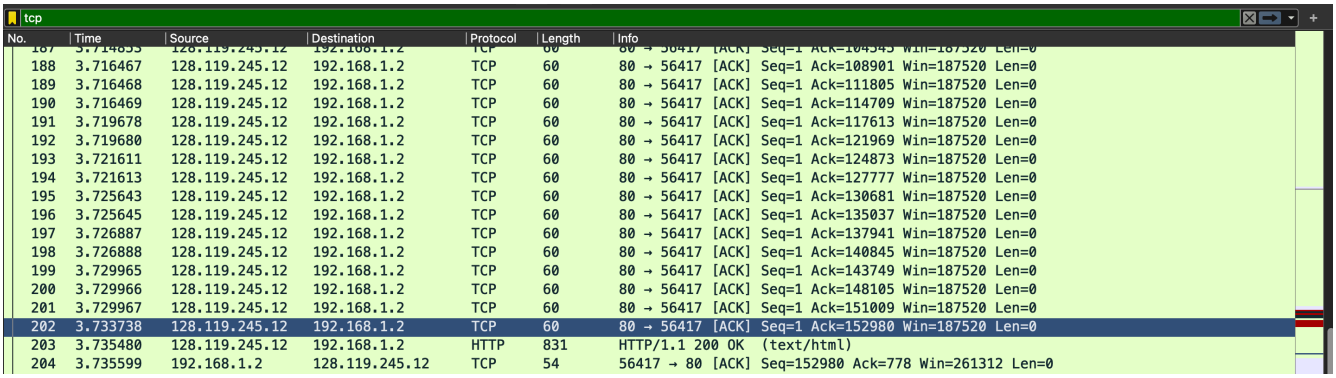


Figure 11: Answer 11-2

Answer 11.1

The throughput for the TCP connection is `333514 bytes per second`.

Answer 11.2

Recall from Answer 6.1 that the first segment containing the first bytes of data was sent at the time 3.114329. From Figure 10, we can observe when the last segment containing the last bytes of data was sent, which is the time 3.573017 in terms of seconds. From Figure 11, we can observe how many bytes were transferred from the client to the server, which is the last ACK value being sent from the server to the last data-containing TCP segment sent by the client, `152980 - 2 = 152978`. Throughput is how many bytes were transferred during the amount of time between when the client sent the first segment containing the first bytes of data in alice.txt and when the last segment in the connection containing the last bytes of data in alice.txt was sent.

The calculation of the throughput is as follows:

```
152978 / (3.573017 - 3.114329) = 333514 bytes per second
```

3 TCP congestion control in action

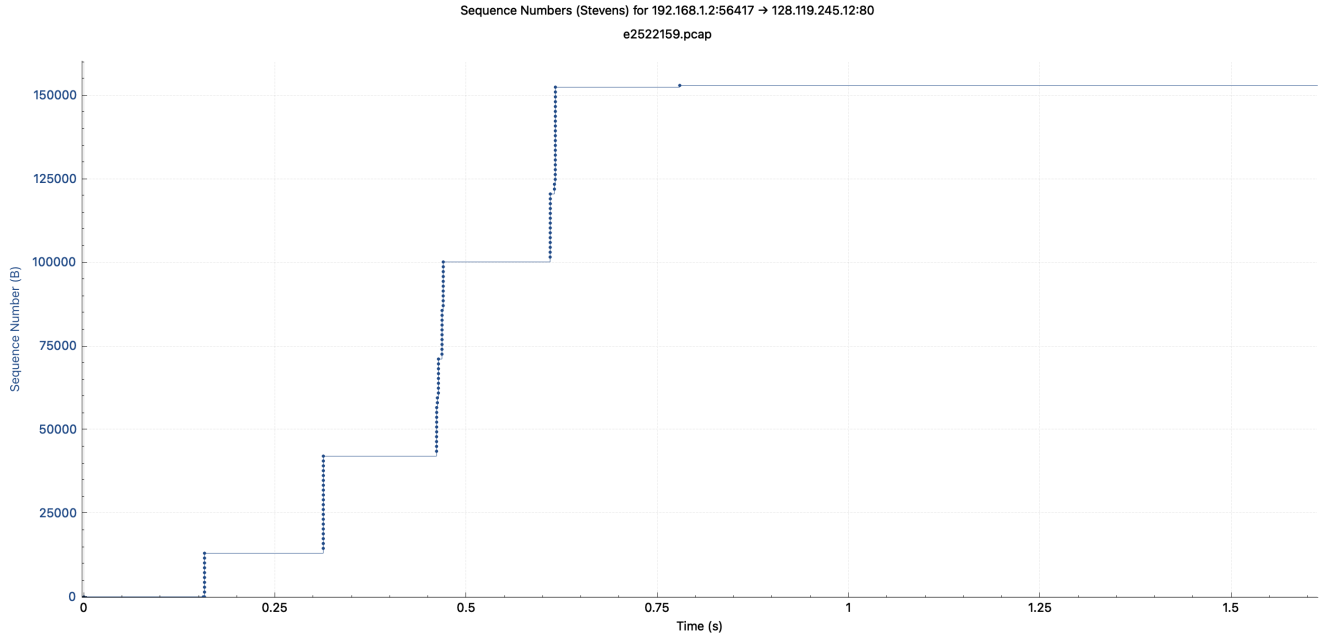


Figure 12: Answer 12

Answer 12.1

The number of packets being sent in each fleet of packets doubles, i.e. exponentially increases. Therefore, it is obvious that TCP is in its slow start phase.

Answer 12.2

It seems that the graph has a period of roughly 0.16 seconds.