COMP416: Computer Networks
Project 2 Report
Transport Layer Protocols' Experiments with Wireshark

Çisem Özden, 69707

Part 1.1 UDP Experiment

1. *What is the segment's time to live value, numerically? What does that mean? What may happen if TTL reaches to zero before the segment arrives its destination?*

Time to live value of my segment is 64. TTL means remaining life time of a packet when it is floating on a network. In other words, it is the amount of time or "hops" that a packet is set to exist inside a network before being discarded by a router. [1] If TTl reaches to zero before the segment arrives its destination, the packet can't arrive to the destination, and it is discarded.

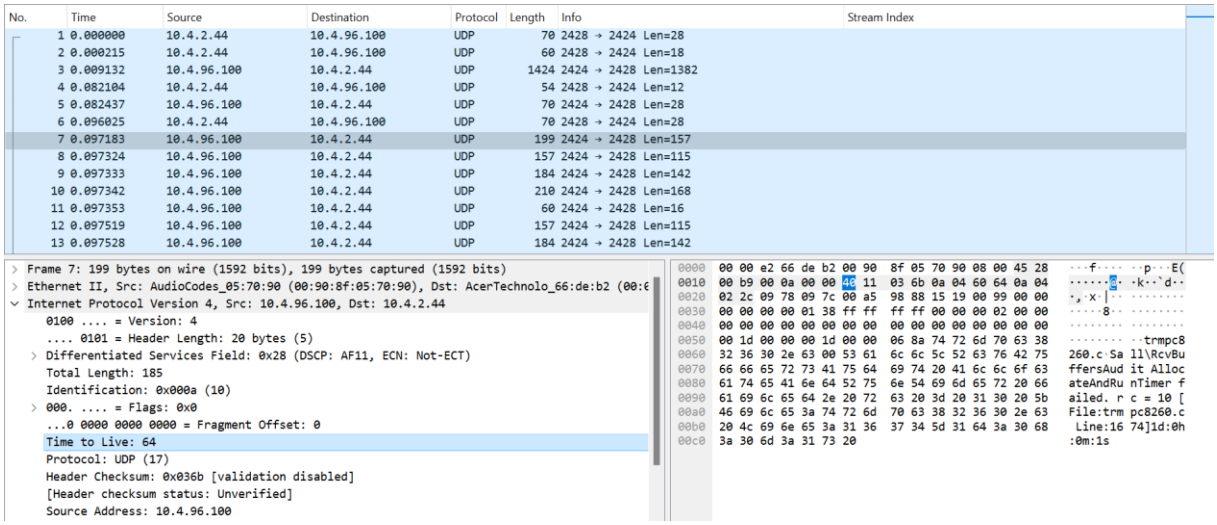


Figure 1. Time to Live value

2. *What does the stream index for a UDP segment signify? What is your segment's stream index?*

In UDP, there is no concept of a "stream" that we see in TCP. UDP is a connectionless protocol and it simply sends data in the form of datagrams, without guaranteeing delivery or order. Thus, UDP segments don't have a stream index as they do in TCP. Therefore, the Stream Index value for UDP segments is 0 as seen also for my package below.

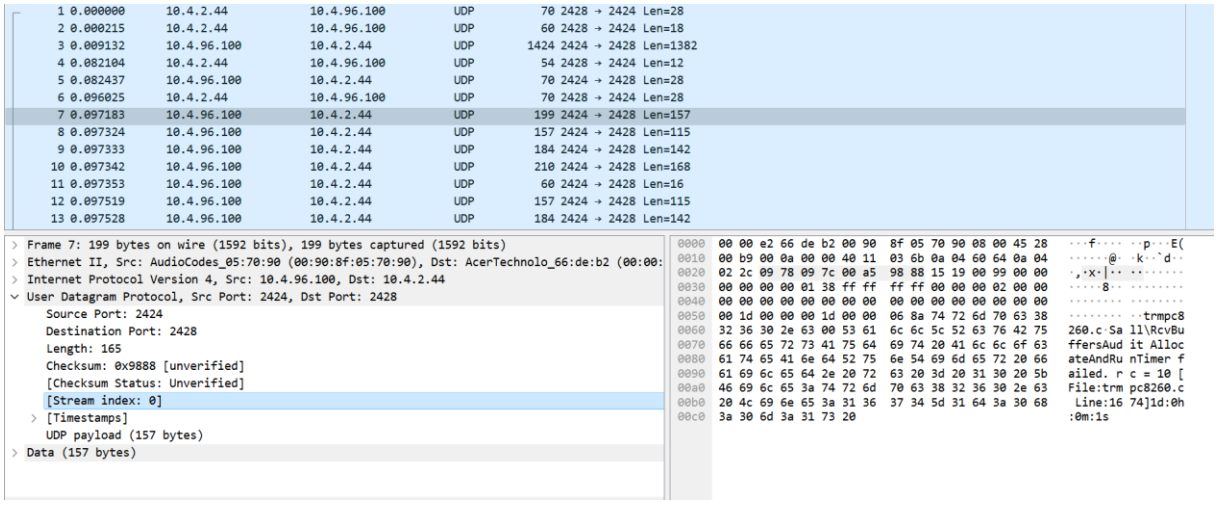


Figure 2. Stream index

3. *What is the checksum value of the segment? What would happen if the value was different than what you have observed? Is there a similar application for the same functionality in TCP? If so, what are the differences in between?*

The checksum value of my segment is 0x9888 as seen below. If the value was different than what I observe, if it was 0 more specifically that would mean a transmission error has occurred. In TCP, there is the same method just like what we have for UDP. In terms of calculation there is no difference between them (Data is divided into 16 bits chunks and they are all added up then the complement of the result becomes the checksum. This checksum is sent along with the data to the receiver. Receiver sums up all of them to check if the result is equal to all 1s in 16 bit.). However, checksums are mandatory with TCP unlike UDP (they are optional with UDP). Furthermore, their usage is mandatory for both the sending and receiving systems.

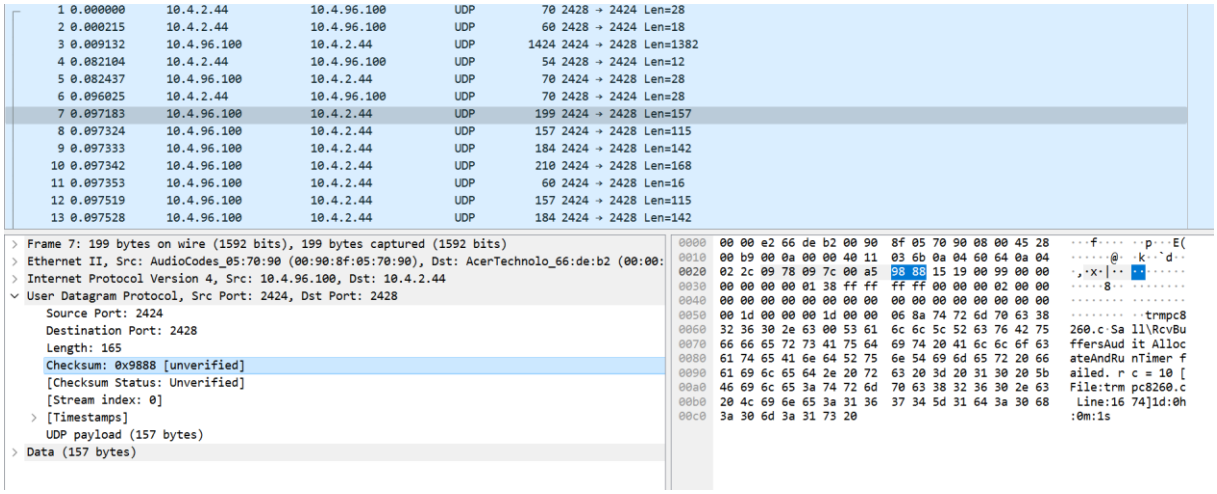


Figure 3. Checksum value

4. *Observe the flags under the IP v4 header. What is the purpose of the Reserved bit flag? What is the value of your segment's reserved bit flag?*

This bit is reserved for future use. It should be set to zero when sending a packet and ignored upon reception. The purpose of that flag is that, if needed in the future, this bit could be used for specific functionality without requiring a fundamental change to the IPv4 protocol. The value of my segment's reserved bit is 0 (not set) as well, as seen below.

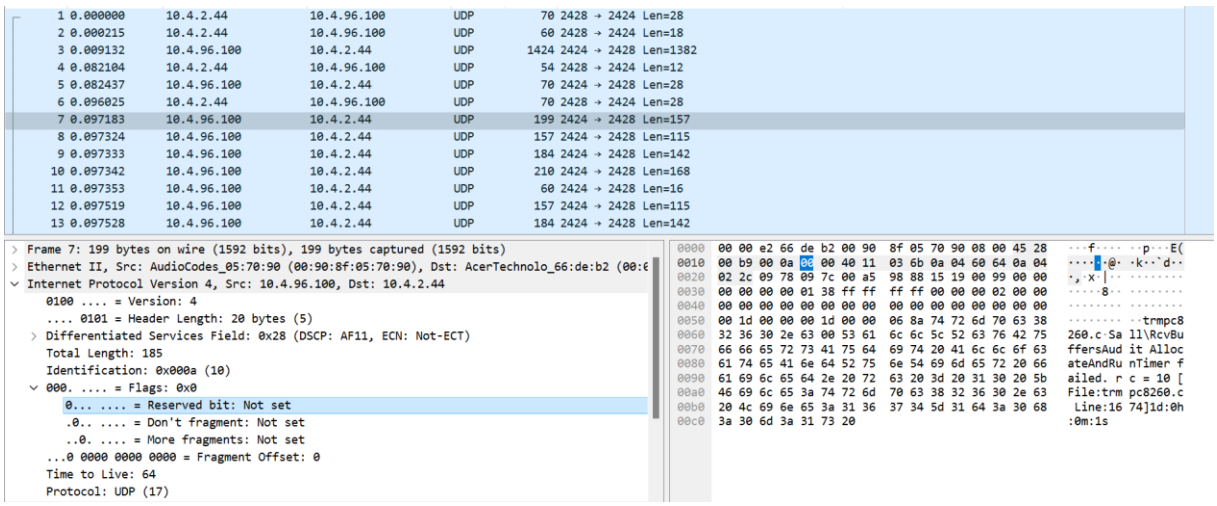


Figure 4. Reserved Bit

5. *Explain the Length field. What does this value signify? Provide your packet's length.*

The length value determines the size of the whole packet including the header, and the data (payload) that has been sent on that packet. Packet lengths is useful and it provide regarding the size of packets being transmitted. Unusual or unexpected values in this field could point out problems related to packet size or potential issues within the network. My packet's total length is 185, and the length of data is 157 (in bytes).

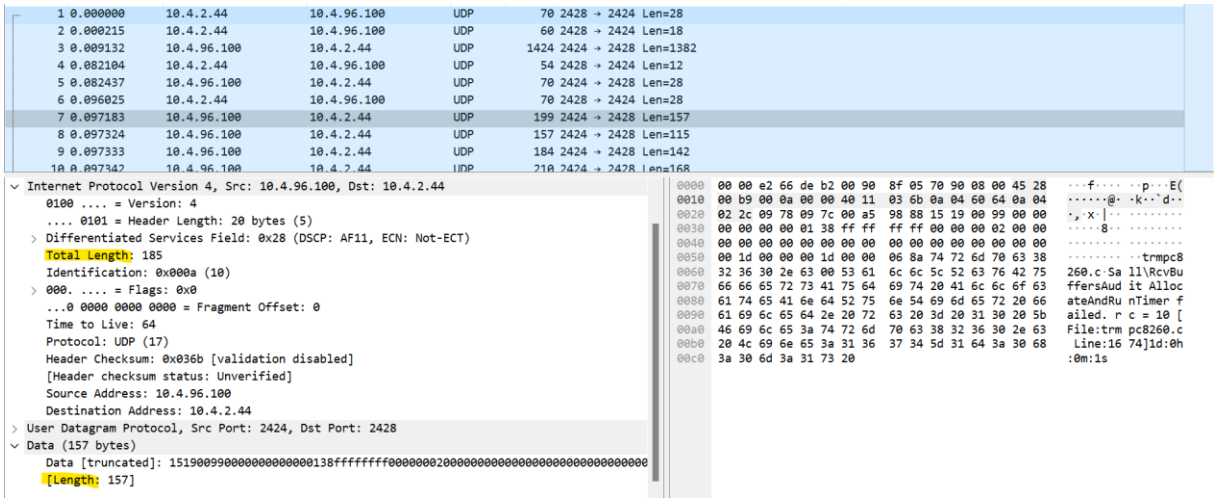


Figure 5. Packet Length

Part 1.2 TCP Experiment

Before answering the questions, it can be seen below that the desired post operation is done.

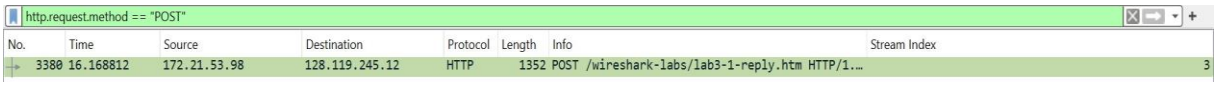


Figure 6. Post Operation

1. *Can you identify any segments marked as retransmissions? Are there indications of segment loss in the TCP flow? (Hint: Use post-filters.)*

There are packets marked as retransmissions as seen from the screenshot below.

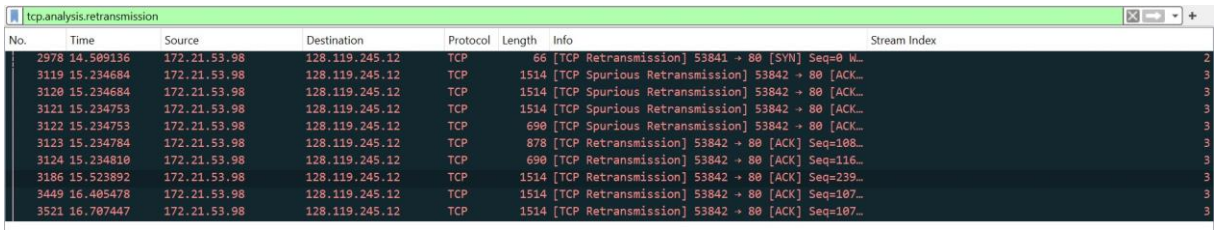


Figure 7. Retransmissions

When there are more than two "TCP dup ack" packets received, it is a strong indication of packet loss over the network. Result of tcp.analysis.duplicate_ack post filtter can be seen below.

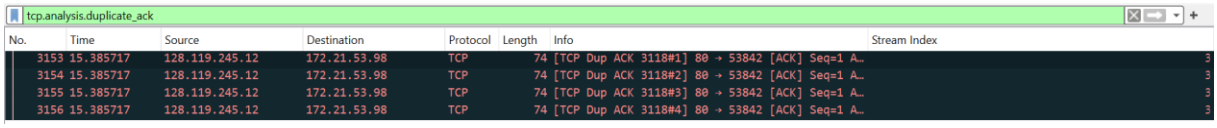


Figure 8. Possible Segment Losses

2. *What is the round-trip time for a specific TCP connection? Are there variations in RTT over the course of the communication? (Hint: Use the "Statistics" menu.)*

After obtaining the RTT graph for the communication, and selecting a specific TCP, for example I selected the segment with number 3068 which has a RTT value of approximately 445 ms. Over the course of the communication RTT shows some fluctuations as seen from the graph below. At some certain points there exists dramatic increases. However, overall it can be said that RTT varies between 50-200 ms fort his particular communication.

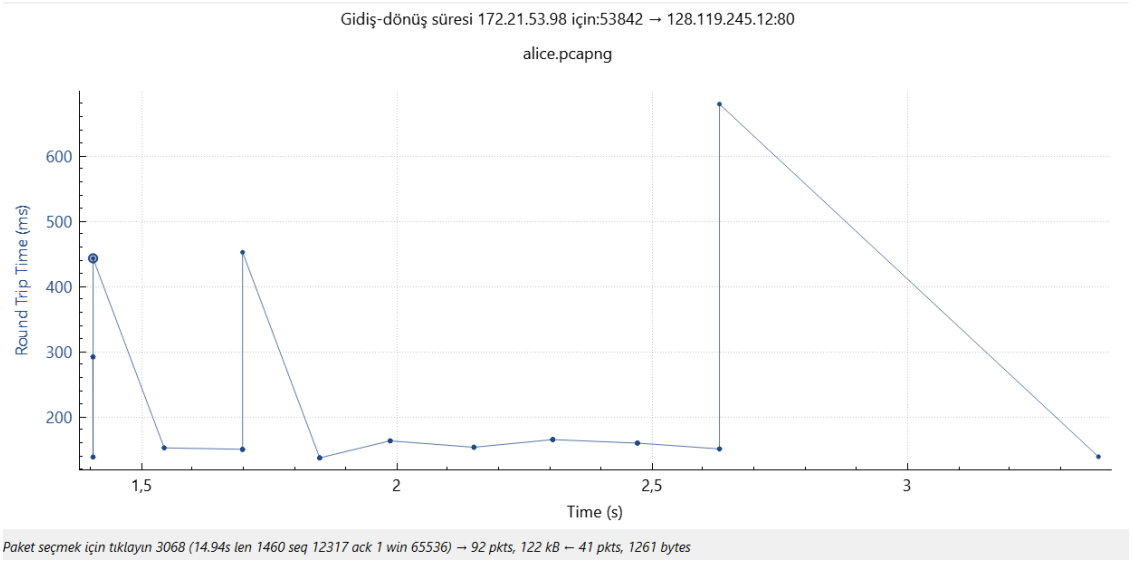


Figure 9. RTT Graph

3. *Identify the information that was present in the TCP but not in UDP. What do they signify? What is their purpose?*

In TCP, one can observe sequence numbers and acknowledgment numbers. These values help in ensuring reliable and ordered delivery of data. Sequence numbers are ued to indicate the order of the data in the stream, and acknowledgment numbers are used to confirm that the data is received. In addition, SYN (synchronize), ACK (acknowledge), FIN (finish), and RST (reset) flags are used in TCP in the scenarious like connection establishment, termination. However, we can't see such flags in UDP.

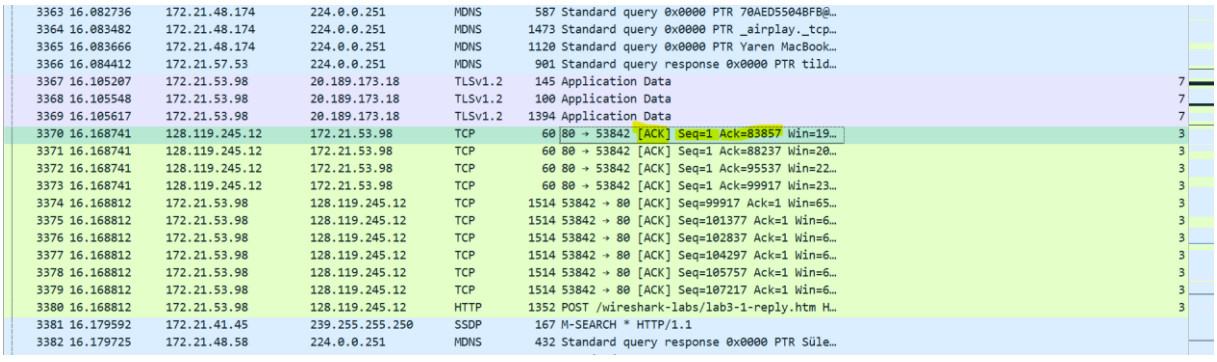


Figure 10. TCP specific fields

4. *Observe the TCP and UDP segments' flow graphs from the Statistics menu. Explain what is being sent as data and their meanings. Underline the differences.*

In the flow graph of UDP we see the requests and responses one under the other between the two ports. Also, we see that the packets are not sent necessarily in order. We can understand this by having more than one request or response from one side to the other. Conversely, in TCP, we see that the packets are transferred in order. We observe that when a request arrives to the port then the receiver port acknowledges it then make the data transfer possible, follows by other acknowledgements so on an so forth. In addition, in the flow graph of TCP, sequence numbers and acknowledgment numbers present unlike in the flow graph of UDP.

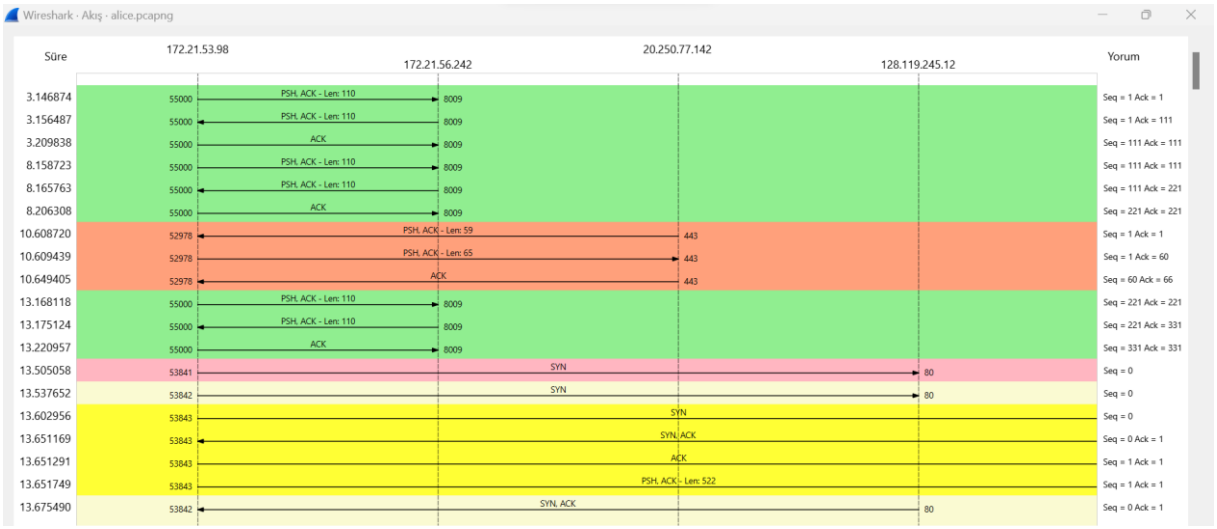


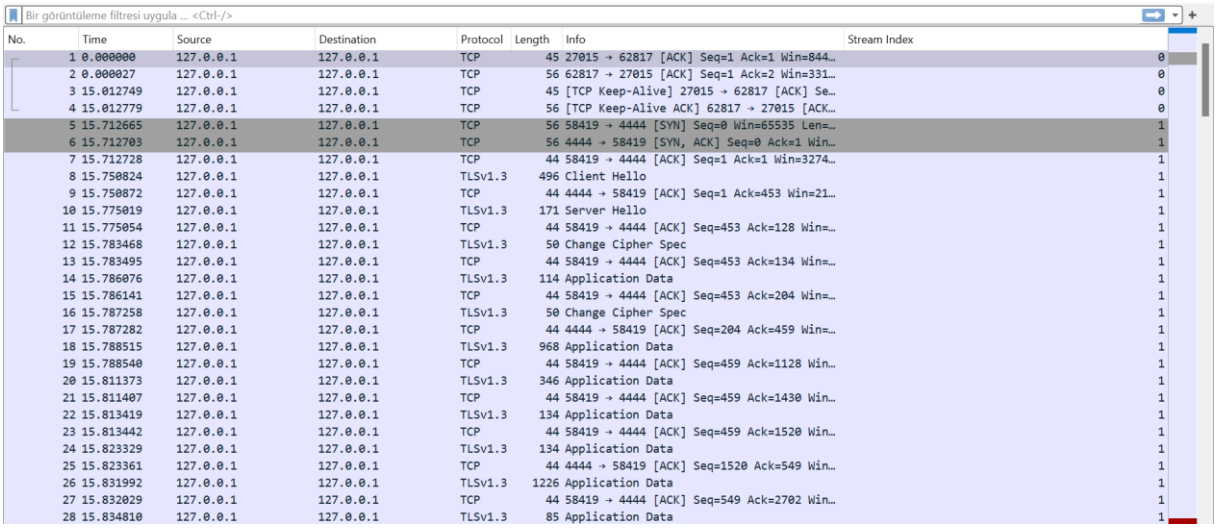
Figure 11. TCP Flow Graph

Figure 12. UDP Flow Graph

Part 2.1 TCP vs SSL Experiments

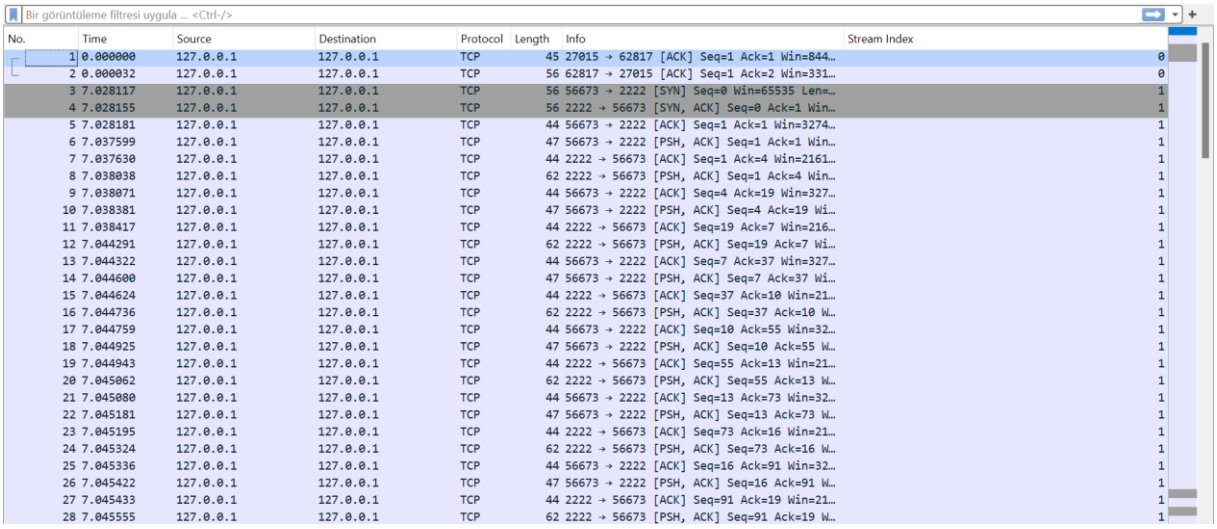
1. *What are the used protocol(s) in the captured packets?*

In the secure connection, TCP and TLS are used as seen from the some captured packets below. In particular, TLSv1.3 is used as TLS protocol. Conversely, in the insecure connection only TCP protocol is used as seen from the captured packets below.



No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
1	0.000000	127.0.0.1	127.0.0.1	TCP	45	27015 → 62817 [ACK] Seq=1 Ack=1 Win=844…	0
2	0.000027	127.0.0.1	127.0.0.1	TCP	56	62817 → 27015 [ACK] Seq=1 Ack=2 Win=331…	0
3	15.012749	127.0.0.1	127.0.0.1	TCP	45	[TCP Keep-Alive] 27015 → 62817 [ACK] Se…	0
4	15.012779	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 62817 → 27015 [ACK…	0
5	15.712665	127.0.0.1	127.0.0.1	TCP	56	58419 → 4444 [SYN] Seq=0 Win=65535 Len=…	1
6	15.712703	127.0.0.1	127.0.0.1	TCP	56	4444 → 58419 [SYN, ACK] Seq=0 Ack=1 Win…	1
7	15.712728	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=1 Ack=1 Win=3274…	1
8	15.750824	127.0.0.1	127.0.0.1	TLSv1.3	496	Client Hello	1
9	15.750872	127.0.0.1	127.0.0.1	TCP	44	4444 → 58419 [ACK] Seq=1 Ack=453 Win=21…	1
10	15.775019	127.0.0.1	127.0.0.1	TLSv1.3	171	Server Hello	1
11	15.775054	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=453 Ack=128 Win=…	1
12	15.783468	127.0.0.1	127.0.0.1	TLSv1.3	50	Change Cipher Spec	1
13	15.783495	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=453 Ack=134 Win=…	1
14	15.786076	127.0.0.1	127.0.0.1	TLSv1.3	114	Application Data	1
15	15.786141	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=453 Ack=204 Win=…	1
16	15.787258	127.0.0.1	127.0.0.1	TLSv1.3	50	Change Cipher Spec	1
17	15.787282	127.0.0.1	127.0.0.1	TCP	44	4444 → 58419 [ACK] Seq=204 Ack=459 Win=…	1
18	15.788515	127.0.0.1	127.0.0.1	TLSv1.3	968	Application Data	1
19	15.788540	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1128 Win…	1
20	15.811373	127.0.0.1	127.0.0.1	TLSv1.3	346	Application Data	1
21	15.811407	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1430 Win…	1
22	15.813419	127.0.0.1	127.0.0.1	TLSv1.3	134	Application Data	1
23	15.813442	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1520 Win…	1
24	15.823329	127.0.0.1	127.0.0.1	TLSv1.3	134	Application Data	1
25	15.823361	127.0.0.1	127.0.0.1	TCP	44	4444 → 58419 [ACK] Seq=1520 Ack=549 Win…	1
26	15.831992	127.0.0.1	127.0.0.1	TLSv1.3	1226	Application Data	1
27	15.832029	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=549 Ack=2702 Win…	1
28	15.834810	127.0.0.1	127.0.0.1	TLSv1.3	85	Application Data	1

Figure 13. Protocols in TLS



No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
1	0.000000	127.0.0.1	127.0.0.1	TCP	45	27015 → 62817 [ACK] Seq=1 Ack=1 Win=844…	0
2	0.000032	127.0.0.1	127.0.0.1	TCP	56	62817 → 27015 [ACK] Seq=1 Ack=2 Win=331…	0
3	7.028117	127.0.0.1	127.0.0.1	TCP	56	56673 → 2222 [SYN] Seq=0 Win=65535 Len=…	1
4	7.028155	127.0.0.1	127.0.0.1	TCP	56	2222 → 56673 [SYN, ACK] Seq=0 Ack=1 Win…	1
5	7.028181	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=1 Ack=1 Win=3274…	1
6	7.037599	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=1 Ack=1 Win…	1
7	7.037630	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=1 Ack=4 Win=2161…	1
8	7.038038	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=1 Ack=4 Win…	1
9	7.038071	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=4 Ack=19 Win=327…	1
10	7.038381	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=4 Ack=19 Wi…	1
11	7.038417	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=19 Ack=7 Win=216…	1
12	7.044291	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=19 Ack=7 Wi…	1
13	7.044322	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=7 Ack=37 Win=327…	1
14	7.044600	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=7 Ack=37 Wi…	1
15	7.044624	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=37 Ack=10 Win=21…	1
16	7.044736	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=37 Ack=10 W…	1
17	7.044759	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=10 Ack=55 Win=32…	1
18	7.044925	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=10 Ack=55 W…	1
19	7.044943	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=55 Ack=13 Win=21…	1
20	7.045062	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=55 Ack=13 W…	1
21	7.045080	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=13 Ack=73 Win=32…	1
22	7.045181	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=13 Ack=73 W…	1
23	7.045195	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=73 Ack=16 Win=21…	1
24	7.045324	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=73 Ack=16 W…	1
25	7.045336	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=16 Ack=91 Win=32…	1
26	7.045422	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=16 Ack=91 W…	1
27	7.045433	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=91 Ack=19 Win=21…	1
28	7.045555	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=91 Ack=19 W…	1

Figure 14. Protocols in TCP

2. *Can you find the sent data (the characters) within the sent packets? Show that as screenshots when possible. &*
3. *By looking at the received response in the packets, can you understand what is sent? Show that as screenshots when possible.*

Looking at the information belonging to the selected packet (6th packet), it is seen that the data sent first is 6 (First character of “69707COMP416”). Response of the server to that message can also be seen from the 8th packet.

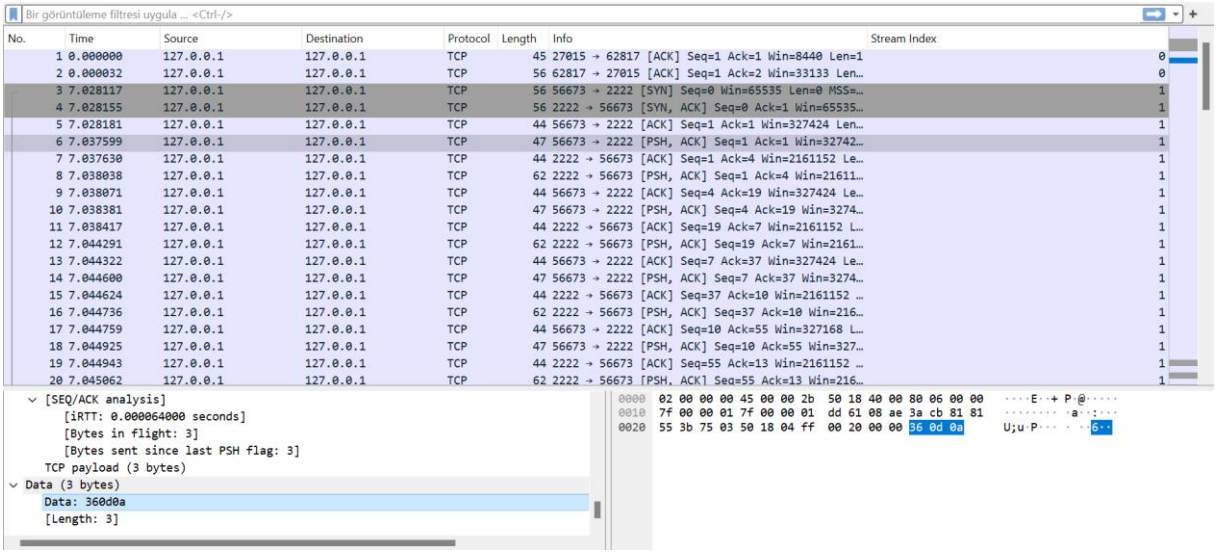


Figure 15. First data sent in TCP

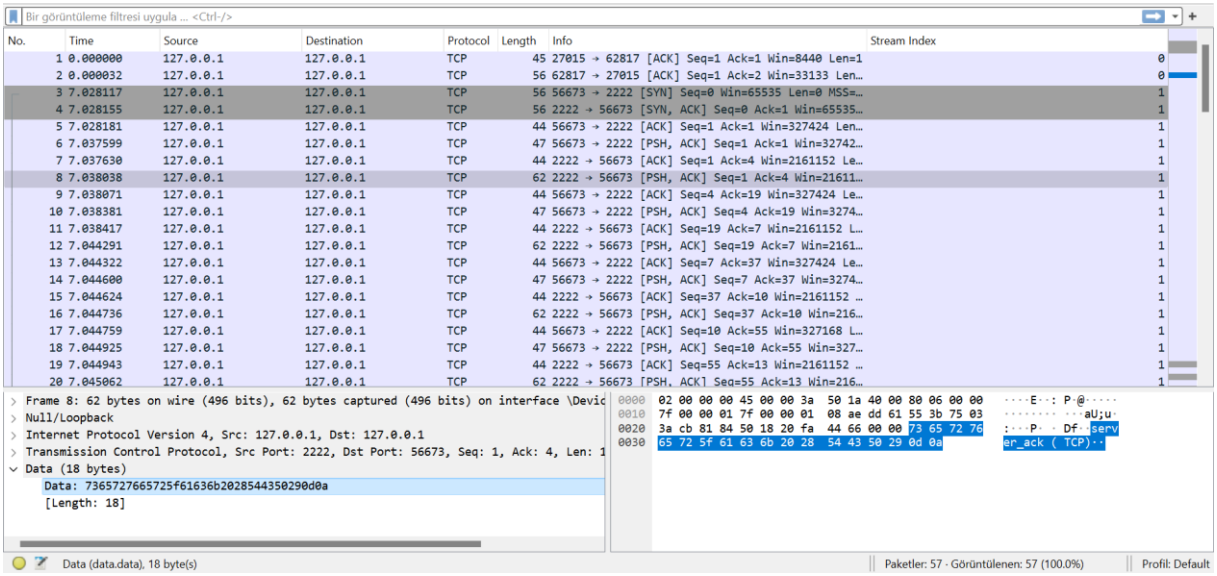
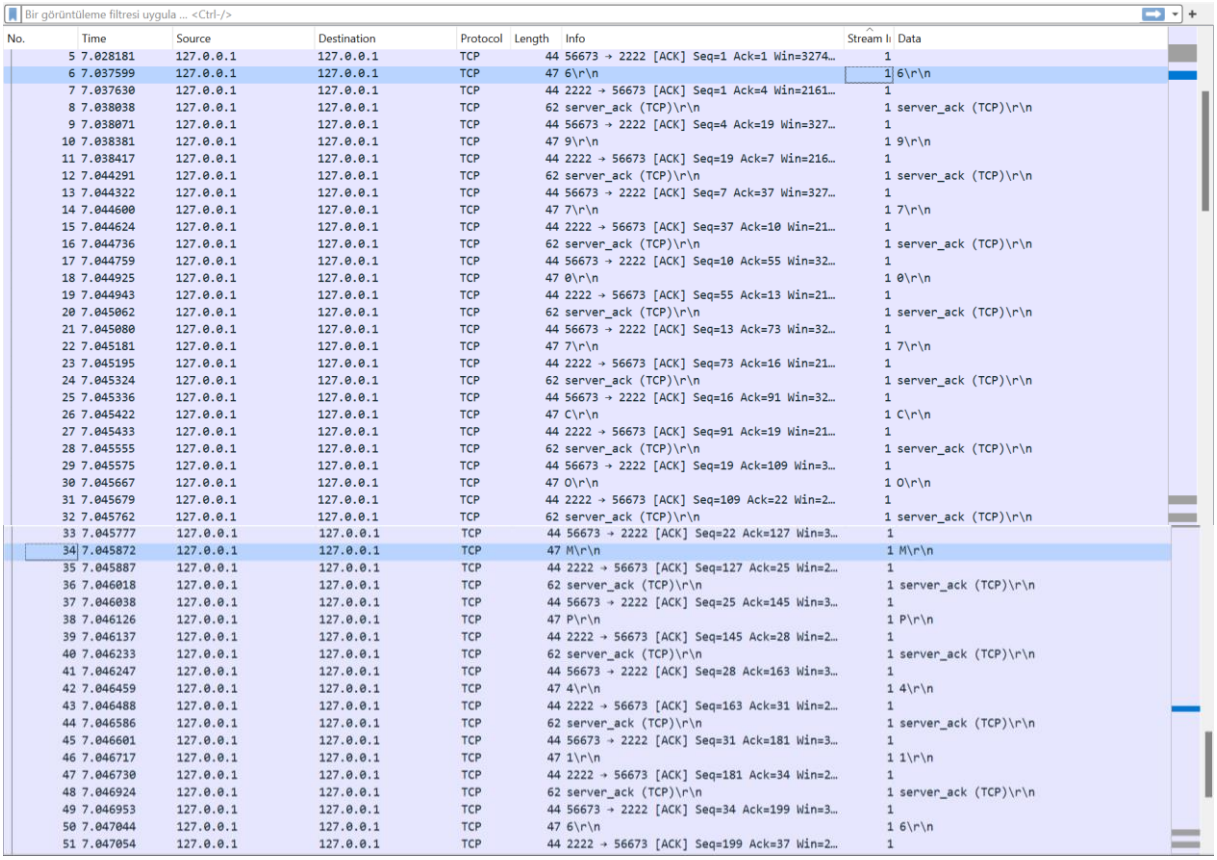


Figure 16. Response to the first data in TCP

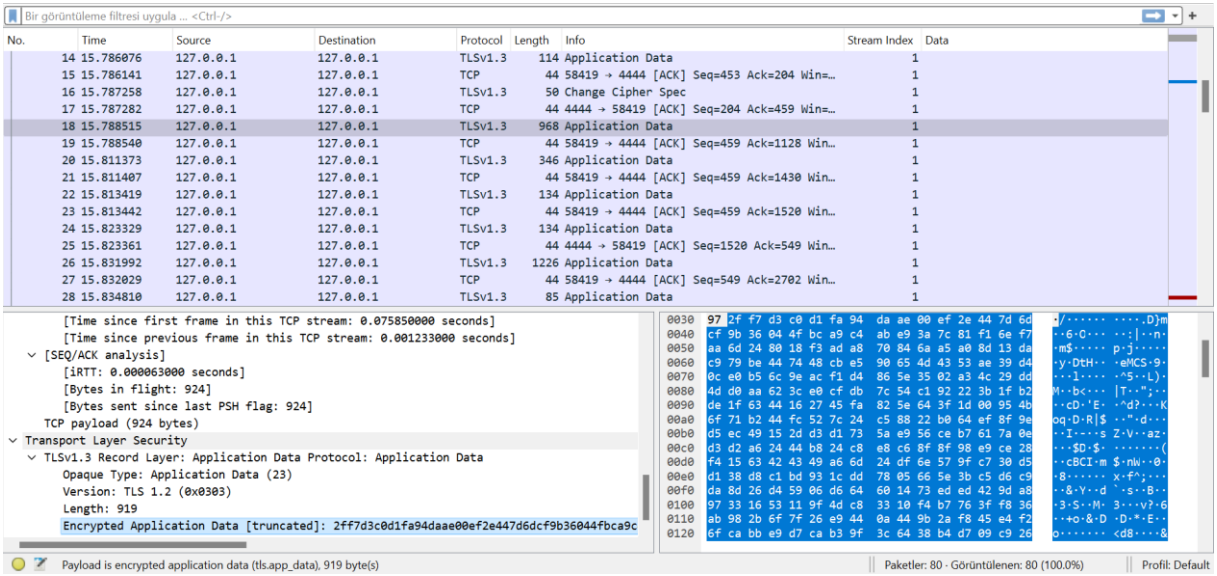
Similarly, other characters sent from the client can be seen from the packets with number 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50. Also corresponding server responses can be seen from the packets with number 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52. To make it easy to see, I made the data.text field visible and added it as column, and now the data being sent between the client and the server can be easily seen from the captured packets below.



No.	Time	Source	Destination	Protocol	Length	Info	Stream Index	Data
5	7.028181	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=1 Ack=1 Win=3274…	1	
6	7.037599	127.0.0.1	127.0.0.1	TCP	47	6\r\n	1	6\r\n
7	7.037630	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=1 Ack=4 Win=2161…	1	
8	7.038038	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
9	7.038071	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=4 Ack=19 Win=327…	1	
10	7.038381	127.0.0.1	127.0.0.1	TCP	47	9\r\n	1	9\r\n
11	7.038417	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=19 Ack=7 Win=216…	1	
12	7.044291	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
13	7.044322	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=7 Ack=37 Win=327…	1	
14	7.044600	127.0.0.1	127.0.0.1	TCP	47	7\r\n	1	7\r\n
15	7.044624	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=37 Ack=10 Win=21…	1	
16	7.044736	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
17	7.044759	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=10 Ack=55 Win=32…	1	
18	7.044925	127.0.0.1	127.0.0.1	TCP	47	0\r\n	1	0\r\n
19	7.044943	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=55 Ack=13 Win=21…	1	
20	7.045062	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
21	7.045080	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=13 Ack=73 Win=32…	1	
22	7.045181	127.0.0.1	127.0.0.1	TCP	47	7\r\n	1	7\r\n
23	7.045195	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=73 Ack=16 Win=21…	1	
24	7.045324	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
25	7.045336	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=16 Ack=91 Win=32…	1	
26	7.045422	127.0.0.1	127.0.0.1	TCP	47	C\r\n	1	C\r\n
27	7.045433	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=91 Ack=19 Win=21…	1	
28	7.045555	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
29	7.045575	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=19 Ack=109 Win=3…	1	
30	7.045667	127.0.0.1	127.0.0.1	TCP	47	O\r\n	1	O\r\n
31	7.045679	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=109 Ack=22 Win=2…	1	
32	7.045762	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
33	7.045777	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=22 Ack=127 Win=3…	1	
34	7.045872	127.0.0.1	127.0.0.1	TCP	47	M\r\n	1	M\r\n
35	7.045887	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=127 Ack=25 Win=2…	1	
36	7.046018	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
37	7.046038	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=25 Ack=145 Win=3…	1	
38	7.046126	127.0.0.1	127.0.0.1	TCP	47	P\r\n	1	P\r\n
39	7.046137	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=145 Ack=28 Win=2…	1	
40	7.046233	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
41	7.046247	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=28 Ack=163 Win=3…	1	
42	7.046459	127.0.0.1	127.0.0.1	TCP	47	4\r\n	1	4\r\n
43	7.046488	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=163 Ack=31 Win=2…	1	
44	7.046586	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
45	7.046601	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=31 Ack=181 Win=3…	1	
46	7.046717	127.0.0.1	127.0.0.1	TCP	47	1\r\n	1	1\r\n
47	7.046730	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=181 Ack=34 Win=2…	1	
48	7.046924	127.0.0.1	127.0.0.1	TCP	62	server_ack (TCP)\r\n	1	server_ack (TCP)\r\n
49	7.046953	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=34 Ack=199 Win=3…	1	
50	7.047044	127.0.0.1	127.0.0.1	TCP	47	6\r\n	1	6\r\n
51	7.047054	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=199 Ack=37 Win=2…	1	

Figure 17. Data sent as characters and responses in TCP

Unlike TCP, we are not able to see either the data sent by the client or the response received from the server. After the handshake is done between the client and the server, in the places normally data would being sent as in TCP case, we can't see any clue about the data, instead we see an information stating that the data is encrpyted as seen below. That means the data being transferred between the server and the client is encrypted within the scope of TLS protocol. (Protocol used in these specific packets can be also seen as TLSv1.3 in the Protocol column). Therefore, packets including the data being sent from the client and the response received from the server are indicated as Application Data, and not visible as in TCP communication, instead, only the encrypted version of the data can be seen when inspected.



No.	Time	Source	Destination	Protocol	Length	Info	Stream Index	Data
14	15.786076	127.0.0.1	127.0.0.1	TLSv1.3	114	Application Data	1	
15	15.786141	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=453 Ack=204 Win=…	1	
16	15.787258	127.0.0.1	127.0.0.1	TLSv1.3	50	Change Cipher Spec	1	
17	15.787282	127.0.0.1	127.0.0.1	TCP	44	4444 → 58419 [ACK] Seq=204 Ack=459 Win=…	1	
18	15.788515	127.0.0.1	127.0.0.1	TLSv1.3	968	Application Data	1	
19	15.788540	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1128 Win…	1	
20	15.811373	127.0.0.1	127.0.0.1	TLSv1.3	346	Application Data	1	
21	15.811407	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1430 Win…	1	
22	15.813419	127.0.0.1	127.0.0.1	TLSv1.3	134	Application Data	1	
23	15.813442	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=459 Ack=1520 Win…	1	
24	15.823329	127.0.0.1	127.0.0.1	TLSv1.3	134	Application Data	1	
25	15.823361	127.0.0.1	127.0.0.1	TCP	44	4444 → 58419 [ACK] Seq=1520 Ack=549 Win…	1	
26	15.831992	127.0.0.1	127.0.0.1	TLSv1.3	1226	Application Data	1	
27	15.832029	127.0.0.1	127.0.0.1	TCP	44	58419 → 4444 [ACK] Seq=549 Ack=2702 Win…	1	
28	15.834810	127.0.0.1	127.0.0.1	TLSv1.3	85	Application Data	1	

```
    [Time since first frame in this TCP stream: 0.075850000 seconds]
    [Time since previous frame in this TCP stream: 0.001233000 seconds]
  [SEQ/ACK analysis]
    [iRTT: 0.000063000 seconds]
    [Bytes in flight: 924]
    [Bytes sent since last PSH flag: 924]
  TCP payload (924 bytes)
Transport Layer Security
  TLSv1.3 Record Layer: Application Data Protocol: Application Data
    Opaque Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 919
    Encrypted Application Data [truncated]: 2ff7d3c0d1fa94daae00ef2e447d6dcf9b36044fbca9c
```

```
0030  97 2f f7 d3 c0 d1 fa 94  da ae 00 ef 2e 44 7d 6d   ·/····· ·····D}m
0040  cf 9b 36 04 4f bc a9 c4  ab e9 3a 7c 81 f1 6e f7   ··6·O··· ··:|··n·
0050  aa 6d 24 80 18 f3 ad a8  70 84 6a a5 a0 8d 13 da   ·m$····· p·j·····
0060  c9 79 be 44 74 48 cb e5  90 65 4d 43 53 ae 39 d4   ·y·DtH·· ·eMCS·9·
0070  0c e0 b5 6c 9e ac f1 d4  86 5e 35 02 a3 4c 29 dd   ···l···· ·^5··L)·
0080  4d d0 aa 62 3c e0 cf db  7c 54 c1 92 22 3b 1f b2   M··b<··· |T··";··
0090  de 1f 63 44 16 27 45 fa  82 5e 64 3f 1d 00 95 4b   ··cD·'E· ·^d?···K
00a0  6f 71 b2 44 fc 52 7c 24  c5 88 22 b0 64 ef 8f 9e   oq·D·R|$ ··"·d···
00b0  d5 ec 49 15 2d d3 d1 73  5a e9 56 ce b7 61 7a 0e   ··I·-··s Z·V··az·
00c0  d3 d2 a6 24 44 b8 24 c8  e8 c6 8f 8f 98 e9 ce 28   ···$D·$· ·······(
00d0  f4 15 63 42 43 49 a6 6d  24 df 6e 57 9f c7 30 d5   ··cBCI·m $·nW··0·
00e0  d1 38 d8 c1 bd 93 1c dd  78 05 66 5e 3b c5 d6 c9   ·8····· x·f^;···
00f0  da 8d 26 d4 59 06 d6 64  60 14 73 ed ed 42 9d a8   ··&·Y··d `·s··B··
0100  97 33 16 53 11 9f 4d c8  33 10 f4 b7 76 3f f8 36   ·3·S··M· 3···v?·6
0110  ab 98 2b 6f 7f 26 e9 44  0a 44 9b 2a f8 45 e4 f2   ··+o·&·D ·D·*·E··
0120  6f ca bb e9 d7 ca b3 9f  3c 64 38 b4 d7 09 c9 26   o····· <d8····&
```

Payload is encrypted application data (tls.app_data), 919 byte(s) | Paketler: 80 · Görüntülenen: 80 (100.0%) | Profil: Default

Figure 18. Encrypted data in TLS

4. *Report delays for 5 different executions and present the measurements as a single graph. Briefly describe the reasons for the results you have obtained. Measure the delays in code (Java).*

To measure the time delays I obtained the time before the connection is established and after all the data is sent, and took the difference. Results obtained from 5 different executions can be seen in the graph below. According to the graph, it is easily seen that the time delay belonging to the TLS execution is much longer compared to TCP. The time delay for TLS is around 350 ms, whereas the time delay for TCP is approximately 50 ms. The reason why TLS execution is slower is that it utilizes encryption and decryption during the communication, as well as the time it takes to establish the connection is longer due to the extra operations done during TLS handshake.

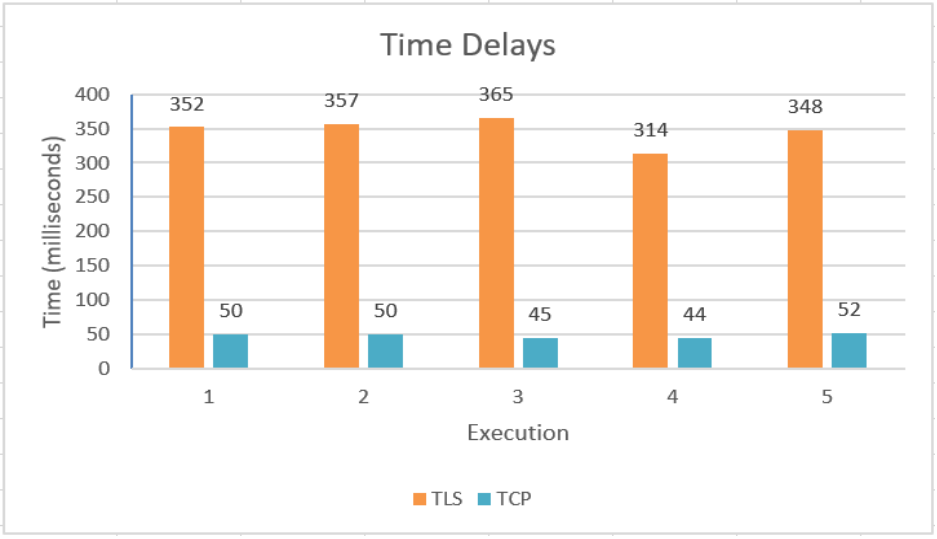


Figure 19. Time Delay Graph of TLS and TCP Executions

Answer the following questions considering only the secure channel (SSL) execution:

5. *In the observed packets, you can see "Client Hello" and "Server Hello ..." packets. Briefly explain what those packets are and write your own observations.*

Client Hello and Server Hello packets are simply the handshake protocols as can be seen from the figures below (in the Transport Layer Security part).

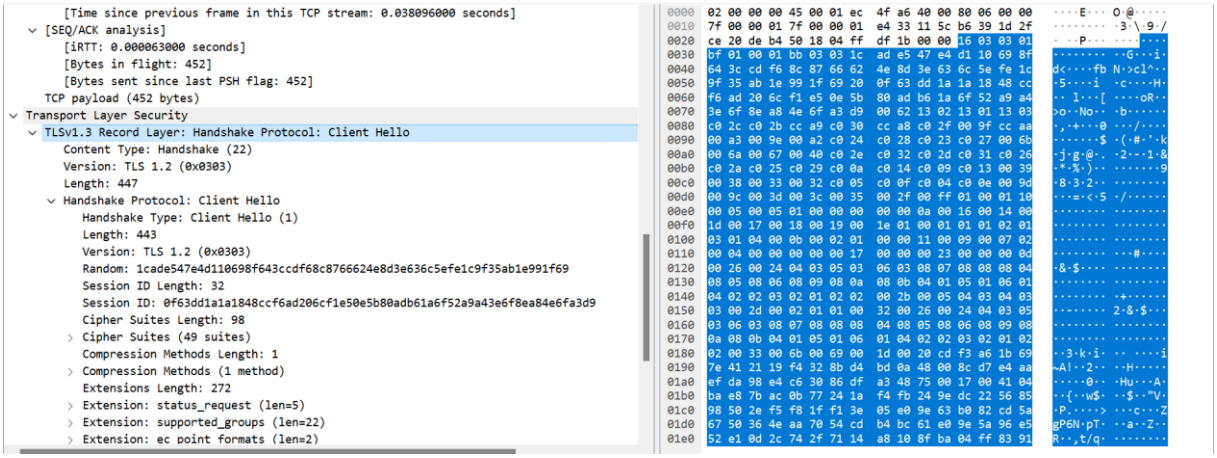


Figure 20. Client Hello Packet

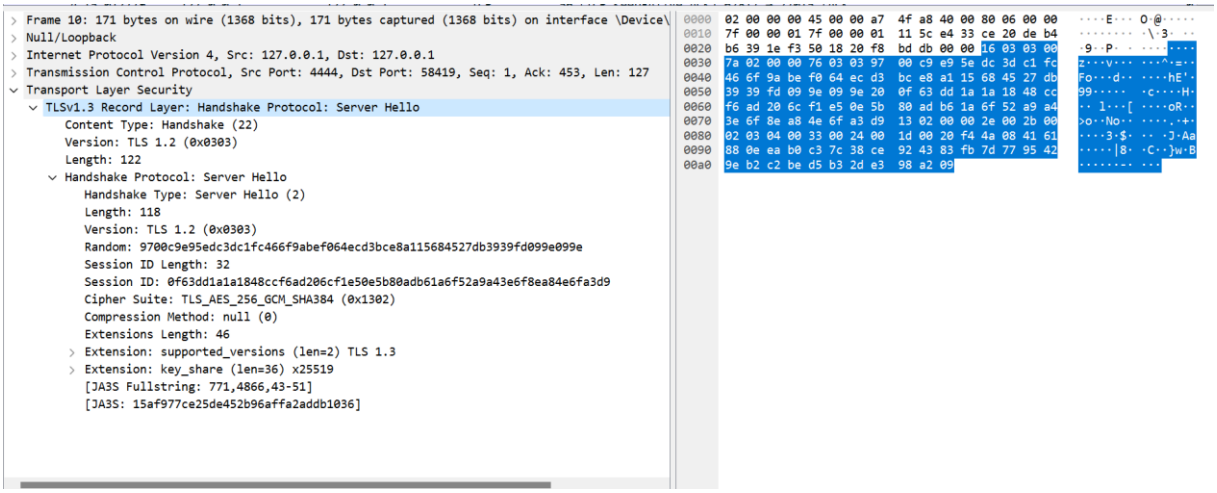


Figure 21. Server Hello Packet

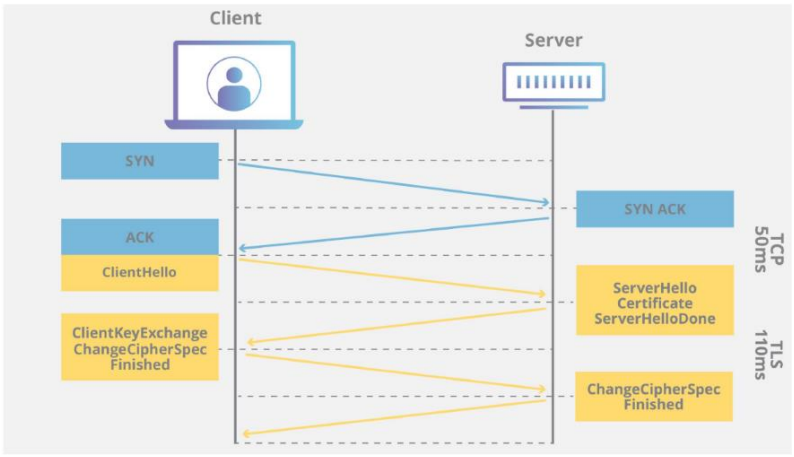


Figure 22. TLS Handshake

Above figure taken from the SSL related practical content shows the steps of TLS handshake. As seen from this figure, ClientHello and ServerHello protocols are used at the beginning. According to my observations, in the content of ClientHello, mainly, which TLS version the client supports, the cipher suites client supports, a string of random bytes, as well as some other specifications are included as seen in the figure below. The TLS Handshake is initiated by the ClientHello message. After that, the server sends ServerHello message as a response to the ClientHello. In the content of ServerHello, mainly the cipher suit (which determines what kind of algorithms to be used in ecryption, decryption etc.) chosen by the server, TLS version chosen by the server, a string of random bytes, and some other specifications are included as seen in the figure below. Mentioned cipher suit and randoms are used for creating session keys and encrypt information.

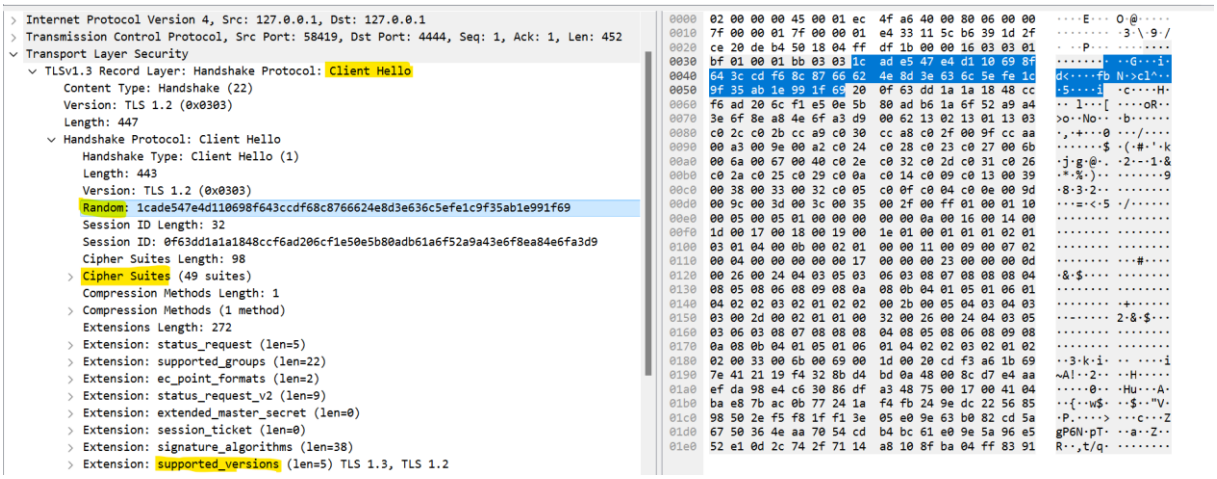


Figure 23. Content of Client Hello

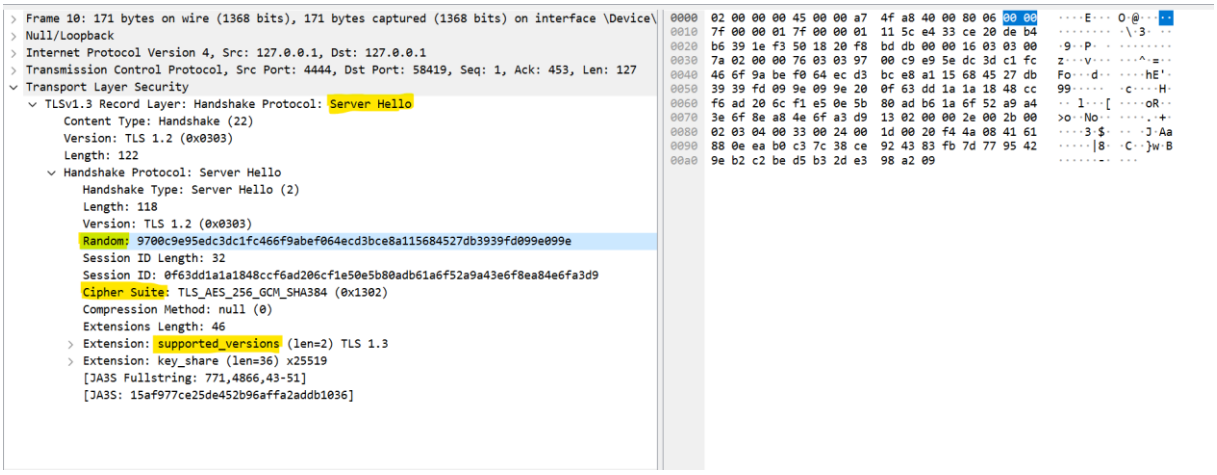


Figure 24. Content of Server Hello

6. *How does the protocol avoid sending the certificate for each connection? Show the used solution from the captured packets.*

To avoid sending the certificate for each connection and optimize performance, "session resumption" is used. Session resumption allows a client and server to reuse previously established session parameters without needing a full handshake process again, including the certificate. In subsequent connections with the same client, the client can include the session identifier associated with the previous connection. If the server recognizes the session identifier, it can quickly resume the session without establishing a full handshake. Below figure shows that the session parameters like session id and session length are included. Note that the purpose of keeping session id is indicated also in the bottom bar as “Identifies the SSL session, allowing later resumption”.

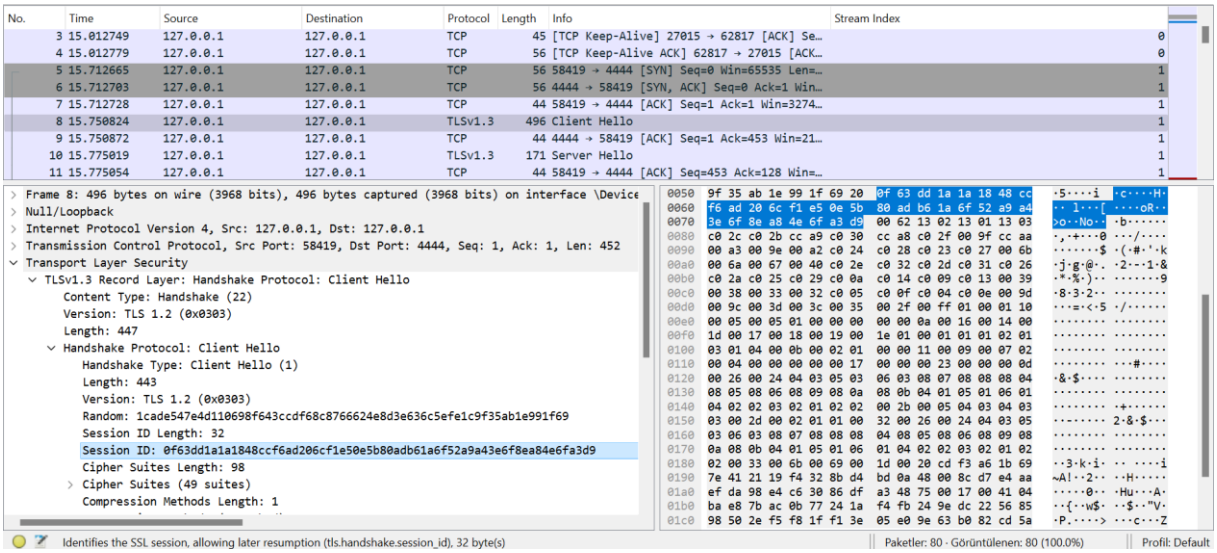


Figure 25. Session Parameters for later Resumption

Part 2.2 Creating Keystore and Trust Store

After following the insructions given in https://docs.oracle.com/cd/E19509-01/820-3503/ggsxx/index.html, the information belonging to the certificate I created can be found below. The password set when creating the keystore and truststore is cozden18.

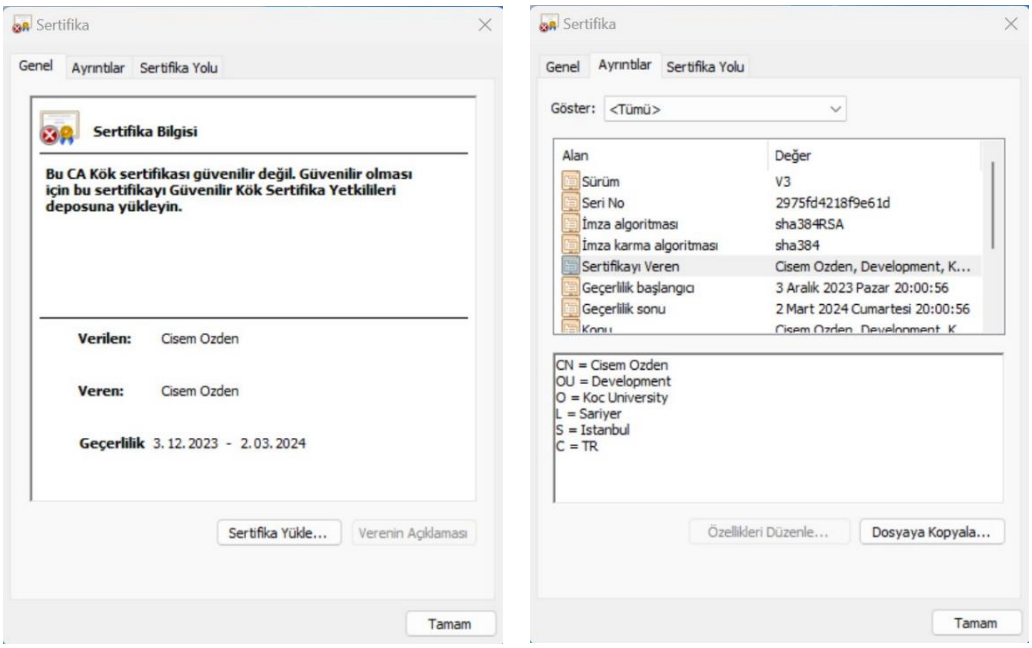


Figure 26. Certificate

References

1. https://www.cloudflare.com/learning/
2. https://osqa-ask.wireshark.org/questions/