

COMP416: Computer Networks

Project 2 Report

Transport Layer Protocols' Experiments with Wireshark

Çisem Özden, 69707

Part 1.1 UDP Experiment

- 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.

No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
1	0.000000	10.4.2.44	10.4.96.100	UDP	70	2428 → 2424 Len=28	
2	0.000215	10.4.2.44	10.4.96.100	UDP	60	2428 → 2424 Len=18	
3	0.009132	10.4.96.100	10.4.2.44	UDP	1424	2424 → 2428 Len=1382	
4	0.0822104	10.4.2.44	10.4.96.100	UDP	54	2428 → 2424 Len=12	
5	0.082437	10.4.96.100	10.4.2.44	UDP	70	2424 → 2428 Len=28	
6	0.096025	10.4.2.44	10.4.96.100	UDP	70	2428 → 2424 Len=28	
7	0.097183	10.4.96.100	10.4.2.44	UDP	199	2424 → 2428 Len=157	
8	0.097324	10.4.96.100	10.4.2.44	UDP	157	2424 → 2428 Len=115	
9	0.097333	10.4.96.100	10.4.2.44	UDP	184	2424 → 2428 Len=142	
10	0.097342	10.4.96.100	10.4.2.44	UDP	210	2424 → 2428 Len=168	
11	0.097353	10.4.96.100	10.4.2.44	UDP	60	2424 → 2428 Len=16	
12	0.097519	10.4.96.100	10.4.2.44	UDP	157	2424 → 2428 Len=115	
13	0.097528	10.4.96.100	10.4.2.44	UDP	184	2424 → 2428 Len=142	

```

> Frame 7: 199 bytes on wire (1592 bits), 199 bytes captured (1592 bits)
> Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:0e:00:00:00:00)
  Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
    Version: 4
    ....0100 .... = Version: 4
    ....0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
    Total Length: 185
    Identification: 0x000a (10)
  > 000. .... = Flags: 0x08
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
  Protocol: UDP (17)
  Header Checksum: 0x036b [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 10.4.96.100

```

No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
0000	00 00 e2 66 de b2 00 90	8f 05 70 90 08 00 45 28	...f.... p..E(
0010	00 b9 00 8a 00 00 40 11	03 6b 0a 04 60 64 0a 04@.. -k..d..				
0020	02 2c 09 78 09 7c 00 a5	98 88 15 19 00 99 00 00	..,X ..				
0030	00 00 00 00 01 38 ff ff	ff ff 00 00 00 02 00 008.....				
0040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				
0050	00 1d 00 00 00 1d 00 00	06 8a 74 72 6d 70 63 38				
0060	32 36 30 2e 63 00 53 61	6c 6c 5c 52 63 76 42 75	260.c Sa 11 RcvBu				
0070	66 66 65 72 73 41 75 64	69 74 20 41 6c 6f 63	ffersAud it Alloc				
0080	61 74 65 41 6e 64 52 75	6e 54 69 6d 65 72 20 66	ateAndRu nTimer f				
0090	61 69 6c 65 64 2e 28 72	63 28 3d 28 31 38 20 5b	ailed. r c = 10 [
00a0	46 69 6c 65 3a 74 72 6d	70 63 38 32 36 38 20 63	File:trm_pc8260.c				
00b0	20 4c 69 6e 65 3a 31 36	37 34 5d 31 64 3a 30 68	Line:16 74]1d:0h				
00c0	3a 30 6d 3a 31 73 28		:0m:1s				

Figure 1. Time to Live value

- 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.

No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
1	0.000000	10.4.2.44	10.4.96.100	UDP	70	2428 → 2424 Len=28	
2	0.000215	10.4.2.44	10.4.96.100	UDP	60	2428 → 2424 Len=18	
3	0.009132	10.4.96.100	10.4.2.44	UDP	1424	2424 → 2428 Len=1382	
4	0.0822104	10.4.2.44	10.4.96.100	UDP	54	2428 → 2424 Len=12	
5	0.082437	10.4.96.100	10.4.2.44	UDP	70	2424 → 2428 Len=28	
6	0.096025	10.4.2.44	10.4.96.100	UDP	70	2428 → 2424 Len=28	
7	0.097183	10.4.96.100	10.4.2.44	UDP	199	2424 → 2428 Len=157	
8	0.097324	10.4.96.100	10.4.2.44	UDP	157	2424 → 2428 Len=115	
9	0.097333	10.4.96.100	10.4.2.44	UDP	184	2424 → 2428 Len=142	
10	0.097342	10.4.96.100	10.4.2.44	UDP	210	2424 → 2428 Len=168	
11	0.097353	10.4.96.100	10.4.2.44	UDP	60	2424 → 2428 Len=16	
12	0.097519	10.4.96.100	10.4.2.44	UDP	157	2424 → 2428 Len=115	
13	0.097528	10.4.96.100	10.4.2.44	UDP	184	2424 → 2428 Len=142	

```

> Frame 7: 199 bytes on wire (1592 bits), 199 bytes captured (1592 bits)
> Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:0e:00:00:00:00)
  Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
    Version: 4
    ....0100 .... = Version: 4
    ....0101 = Header Length: 20 bytes (5)
  > User Datagram Protocol, Src Port: 2424, Dst Port: 2428
    Source Port: 2424
    Destination Port: 2428
    Length: 165
    Checksum: 0x9888 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
    > [Timestamps]
      UDP payload (157 bytes)
    > Data (157 bytes)

```

No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
0000	00 00 e2 66 de b2 00 90	8f 05 70 90 08 00 45 28	...f.... p..E(
0010	00 b9 00 8a 00 00 40 11	03 6b 0a 04 60 64 0a 04@.. -k..d..				
0020	02 2c 09 78 09 7c 00 a5	98 88 15 19 00 99 00 00	..,X ..				
0030	00 00 00 00 01 38 ff ff	ff ff 00 00 00 02 00 008.....				
0040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00				
0050	00 1d 00 00 00 1d 00 00	06 8a 74 72 6d 70 63 38				
0060	32 36 30 2e 63 00 53 61	6c 6c 5c 52 63 76 42 75	260.c Sa 11 RcvBu				
0070	66 66 65 72 73 41 75 64	69 74 20 41 6c 6f 63	ffersAud it Alloc				
0080	61 74 65 41 6e 64 52 75	6e 54 69 6d 65 72 20 66	ateAndRu nTimer f				
0090	61 69 6c 65 64 2e 28 72	63 28 3d 28 31 38 20 5b	ailed. r c = 10 [
00a0	46 69 6c 65 3a 74 72 6d	70 63 38 32 36 38 20 63	File:trm_pc8260.c				
00b0	20 4c 69 6e 65 3a 31 36	37 34 5d 31 64 3a 30 68	Line:16 74]1d:0h				
00c0	3a 30 6d 3a 31 73 28		:0m:1s				

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.

1 0.000000 10.4.2.44 10.4.96.100 UDP 70 2428 → 2424 Len=28	0000 00 00 e2 66 de b2 00 90 8f 05 70 90 08 00 45 28	...f... p...E(
2 0.000215 10.4.2.44 10.4.96.100 UDP 60 2428 → 2424 Len=18	0010 00 b9 00 0a 00 00 40 11 03 6b 0a 04 60 64 0a 04	...@... -k...d...
3 0.009132 10.4.2.44 10.4.96.100 UDP 1424 2424 → 2428 Len=1382	0020 02 2c 09 78 09 7c 00 a5 98 88 15 19 00 99 00 00	,x8
4 0.082104 10.4.2.44 10.4.96.100 UDP 54 2428 → 2424 Len=12	0030 00 00 00 00 01 38 ff ff ff ff 00 00 00 02 00 00
5 0.082437 10.4.2.44 10.4.96.100 UDP 70 2424 → 2428 Len=28	0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6 0.096025 10.4.2.44 10.4.96.100 UDP 70 2428 → 2424 Len=28	0050 00 1d 00 00 00 00 1d 00 00 06 8a 74 72 6d 70 63 38	trmpc8
7 0.097183 10.4.2.44 10.4.96.100 UDP 199 2424 → 2428 Len=157	0060 32 36 30 2e 63 00 53 61 6c 6c 5c 52 63 76 42 75	260.c Sa 11 RcvBu
8 0.097324 10.4.2.44 10.4.96.100 UDP 157 2424 → 2428 Len=115	0070 66 66 65 72 73 41 75 64 69 74 20 41 6c 6c 6f 63	ffersAud it Alloc
9 0.097333 10.4.2.44 10.4.96.100 UDP 184 2424 → 2428 Len=142	0080 61 74 65 41 64 64 52 75 6e 54 69 6d 65 72 20 66	ateAndRu nTimer f
10 0.097342 10.4.2.44 10.4.96.100 UDP 210 2424 → 2428 Len=168	0090 61 69 6c 65 64 6e 2e 72 72 63 20 3d 28 31 30 20 5b	ailed. r c = 10 [
11 0.097353 10.4.2.44 10.4.96.100 UDP 60 2424 → 2428 Len=16	00a0 46 69 6c 65 3a 74 72 6d 70 63 38 32 36 30 2e 63	File:trmp pc8260.c
12 0.097519 10.4.2.44 10.4.96.100 UDP 157 2424 → 2428 Len=115	00b0 20 4c 69 6e 65 3a 31 36 37 34 5d 31 64 3a 30 68	Line:16 74]1d:0h
13 0.097528 10.4.2.44 10.4.96.100 UDP 184 2424 → 2428 Len=142	00c0 3a 30 6d 3a 31 73 20	:0m:1s

> Frame 7: 199 bytes on wire (1592 bits), 199 bytes captured (1592 bits)
> Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:00:
> Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
> User Datagram Protocol, Src Port: 2424, Dst Port: 2428
Source Port: 2424
Destination Port: 2428
Length: 165
Checksum: 0x9888 [Unverified]
[Checksum Status: Unverified]
[Stream index: 0]
> [Timestamps]
UDP payload (157 bytes)
Data (157 bytes)

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.

1 0.000000 10.4.2.44 10.4.96.100 UDP 70 2428 → 2424 Len=28	0000 00 00 e2 66 de b2 00 90 8f 05 70 90 08 00 45 28	...f... p...E(
2 0.000215 10.4.2.44 10.4.96.100 UDP 60 2428 → 2424 Len=18	0010 00 b9 00 0a 00 00 40 11 03 6b 0a 04 60 64 0a 04	...@... -k...d...
3 0.009132 10.4.2.44 10.4.96.100 UDP 1424 2424 → 2428 Len=1382	0020 02 2c 09 78 09 7c 00 a5 98 88 15 19 00 99 00 00	,x8
4 0.082104 10.4.2.44 10.4.96.100 UDP 54 2428 → 2424 Len=12	0030 00 00 00 00 01 38 ff ff ff ff 00 00 00 02 00 00
5 0.082437 10.4.2.44 10.4.96.100 UDP 70 2424 → 2428 Len=28	0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
6 0.096025 10.4.2.44 10.4.96.100 UDP 70 2428 → 2424 Len=28	0050 00 1d 00 00 00 00 1d 00 00 06 8a 74 72 6d 70 63 38	trmpc8
7 0.097183 10.4.2.44 10.4.96.100 UDP 199 2424 → 2428 Len=157	0060 32 36 30 2e 63 00 53 61 6c 6c 5c 52 63 76 42 75	260.c Sa 11 RcvBu
8 0.097324 10.4.2.44 10.4.96.100 UDP 157 2424 → 2428 Len=115	0070 66 66 65 72 73 41 75 64 69 74 20 41 6c 6c 6f 63	ffersAud it Alloc
9 0.097333 10.4.2.44 10.4.96.100 UDP 184 2424 → 2428 Len=142	0080 61 74 65 41 64 64 52 75 6e 54 69 6d 65 72 20 66	ateAndRu nTimer f
10 0.097342 10.4.2.44 10.4.96.100 UDP 210 2424 → 2428 Len=168	0090 61 69 6c 65 64 6e 2e 72 72 63 20 3d 28 31 30 20 5b	ailed. r c = 10 [
11 0.097353 10.4.2.44 10.4.96.100 UDP 60 2424 → 2428 Len=16	00a0 46 69 6c 65 3a 74 72 6d 70 63 38 32 36 30 2e 63	File:trmp pc8260.c
12 0.097519 10.4.2.44 10.4.96.100 UDP 157 2424 → 2428 Len=115	00b0 20 4c 69 6e 65 3a 31 36 37 34 5d 31 64 3a 30 68	Line:16 74]1d:0h
13 0.097528 10.4.2.44 10.4.96.100 UDP 184 2424 → 2428 Len=142	00c0 3a 30 6d 3a 31 73 20	:0m:1s

> Frame 7: 199 bytes on wire (1592 bits), 199 bytes captured (1592 bits)
> Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:00:
> Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
> User Datagram Protocol, Src Port: 2424, Dst Port: 2428
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
Total Length: 185
Identification: 0x000a (10)
< 000. = Flags: 0x0
0... = Reserved bit: Not set
..0. = Don't fragment: Not set
..0. = More fragments: Not set
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)

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.

http.request.method == "POST"						
No.	Time	Source	Destination	Protocol	Length	Info
3380	16.168812	172.21.53.98	128.119.245.12	HTTP	1352	POST /wireshark-labs/lab3-1-reply.htm HTTP/1...

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.

tcp.analysis.retransmission							
No.	Time	Source	Destination	Protocol	Length	Info	Stream Index
2978	14.509136	172.21.53.98	128.119.245.12	TCP	66	[TCP Retransmission] 53841 > 80 [SYN] Seq=0 W...	2
3119	15.234684	172.21.53.98	128.119.245.12	TCP	1514	[TCP Spurious Retransmission] 53842 > 80 [ACK...]	3
3120	15.234684	172.21.53.98	128.119.245.12	TCP	1514	[TCP Spurious Retransmission] 53842 > 80 [ACK...]	3
3121	15.234753	172.21.53.98	128.119.245.12	TCP	1514	[TCP Spurious Retransmission] 53842 > 80 [ACK...]	3
3122	15.234753	172.21.53.98	128.119.245.12	TCP	690	[TCP Spurious Retransmission] 53842 > 80 [ACK...]	3
3123	15.234784	172.21.53.98	128.119.245.12	TCP	878	[TCP Retransmission] 53842 > 80 [ACK] Seq=108...	3
3124	15.234818	172.21.53.98	128.119.245.12	TCP	690	[TCP Retransmission] 53842 > 80 [ACK] Seq=116...	3
3186	15.523892	172.21.53.98	128.119.245.12	TCP	1514	[TCP Retransmission] 53842 > 80 [ACK] Seq=239...	3
3449	16.405478	172.21.53.98	128.119.245.12	TCP	1514	[TCP Retransmission] 53842 > 80 [ACK] Seq=107...	3
3521	16.707744	172.21.53.98	128.119.245.12	TCP	1514	[TCP Retransmission] 53842 > 80 [ACK] Seq=107...	3

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 filter can be seen below.

tcp.analysis.duplicate_ack						
No.	Time	Source	Destination	Protocol	Length	Info
3155	15.385717	128.119.245.12	172.21.53.98	TCP	74	[TCP Dup ACK 3118#1] 80 → 53842 [ACK] Seq=1 A..
3154	15.385717	128.119.245.12	172.21.53.98	TCP	74	[TCP Dup ACK 3118#2] 80 → 53842 [ACK] Seq=1 A..
3155	15.385717	128.119.245.12	172.21.53.98	TCP	74	[TCP Dup ACK 3118#3] 80 → 53842 [ACK] Seq=1 A..
3156	15.385717	128.119.245.12	172.21.53.98	TCP	74	[TCP Dup ACK 3118#4] 80 → 53842 [ACK] Seq=1 A..

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 for his particular communication.

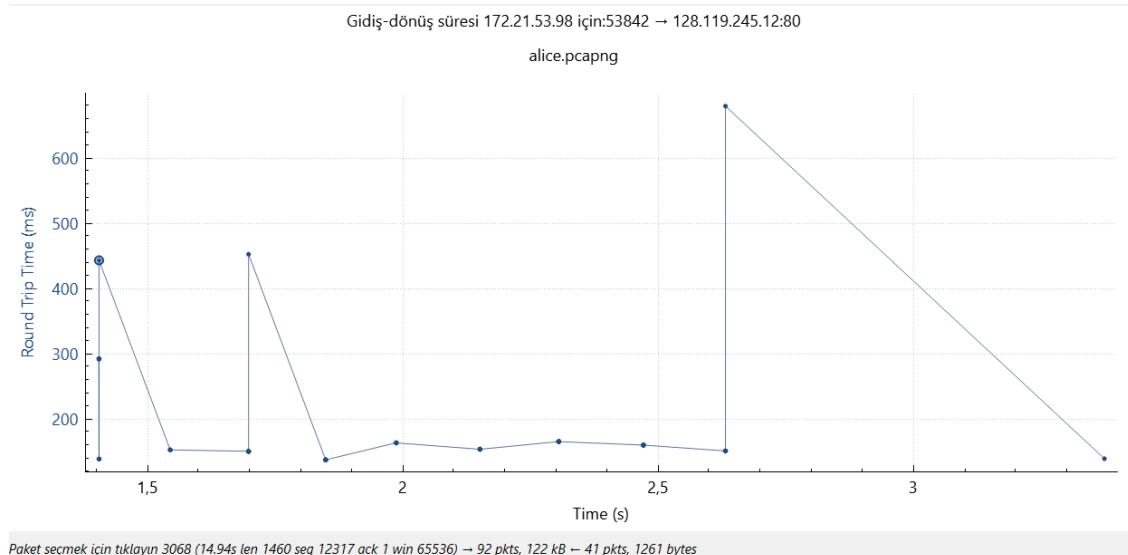


Figure 9. RTT Graph

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 used 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 scenarios like connection establishment, termination. However, we can't see such flags in UDP.

3363 16.082736	172.21.48.174	224.0.0.251	MDNS	587 Standard query 0x0000 PTR 70AED55048FB@..
3364 16.083482	172.21.48.174	224.0.0.251	MDNS	1473 Standard query 0x0000 PTR _airplay._tcp..
3365 16.083666	172.21.48.174	224.0.0.251	MDNS	1120 Standard query 0x0000 PTR Yaren MacBook..
3366 16.084412	172.21.53.98	224.0.0.251	MDNS	901 Standard query response 0x0000 PTR tild..
3367 16.105207	172.21.53.98	20.189.173.18	TLSv1.2	145 Application Data
3368 16.105548	172.21.53.98	20.189.173.18	TLSv1.2	100 Application Data
3369 16.105617	172.21.53.98	20.189.173.18	TLSv1.2	1394 Application Data
3370 16.168741	128.119.245.12	172.21.53.98	TCP	60 80 → 53842 [ACK] Seq=1 Ack=83857 Win=19..
3371 16.168741	128.119.245.12	172.21.53.98	TCP	60 80 → 53842 [ACK] Seq=1 Ack=88237 Win=20..
3372 16.168741	128.119.245.12	172.21.53.98	TCP	60 80 → 53842 [ACK] Seq=1 Ack=95537 Win=22..
3373 16.168741	128.119.245.12	172.21.53.98	TCP	60 80 → 53842 [ACK] Seq=1 Ack=99917 Win=23..
3374 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=99917 Ack=1 Win=65..
3375 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=101377 Ack=1 Win=6..
3376 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=102837 Ack=1 Win=6..
3377 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=104297 Ack=1 Win=6..
3378 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=105757 Ack=1 Win=6..
3379 16.168812	172.21.53.98	128.119.245.12	TCP	1514 53842 → 80 [ACK] Seq=107217 Ack=1 Win=6..
3380 16.168812	172.21.53.98	128.119.245.12	HTTP	1352 POST /wireshark-labs/lab3-1-reply.htm H..
3381 16.179592	172.21.41.45	239.255.255.250	SSDP	167 M-SEARCH * HTTP/1.1
3382 16.179725	172.21.48.58	224.0.0.251	MDNS	432 Standard query response 0x0000 PTR Süle..

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 and 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

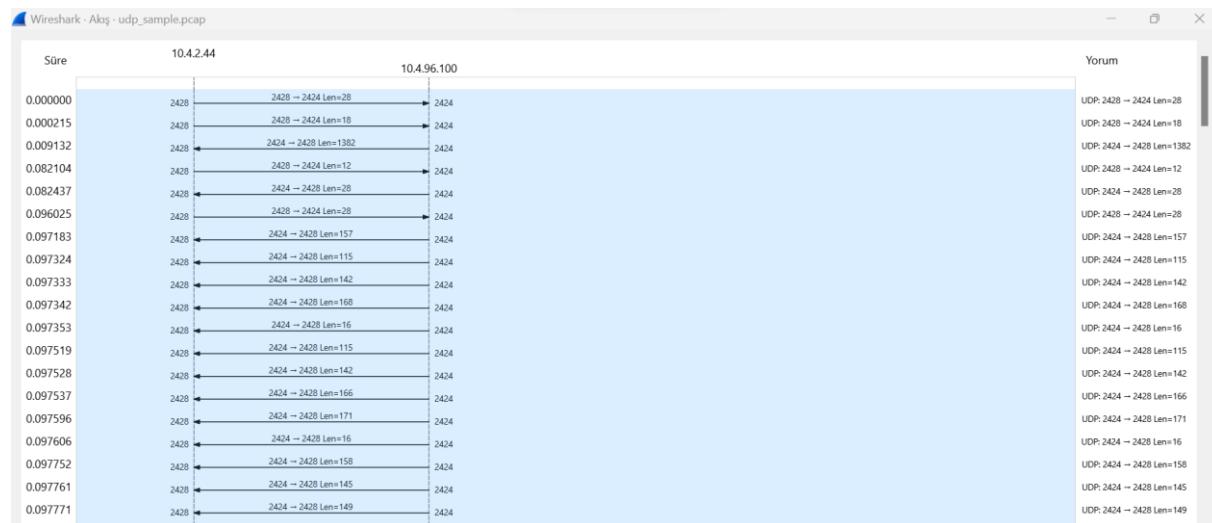


Figure 12. UDP Flow Graph

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] Seq=1 Ack=2 Win=331..	0
4	15.012779	127.0.0.1	127.0.0.1	TCP	56	[TCP Keep-Alive ACK] 62817 → 27015 [ACK] Seq=1 Ack=2 Win=331..	0
5	15.712665	127.0.0.1	127.0.0.1	TCP	56	58419 → 4444 [SYN] Seq=0 Win=65535 Len=1024..	1
6	15.712703	127.0.0.1	127.0.0.1	TCP	56	4444 → 58419 [SYN, ACK] Seq=0 Ack=1 Win=1..	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..	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..	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..	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..	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..	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..	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..	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..	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..	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=1024..	1
4	7.028155	127.0.0.1	127.0.0.1	TCP	56	2222 → 56673 [SYN, ACK] Seq=0 Ack=1 Win=1..	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..	1
7	7.037630	127.0.0.1	127.0.0.1	TCP	44	2222 → 56673 [ACK] Seq=1 Ack=Win=2161..	1
8	7.038388	127.0.0.1	127.0.0.1	TCP	62	2222 → 56673 [PSH, ACK] Seq=1 Ack=4 Win=1..	1
9	7.038871	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=4 Ack=1 Win=327..	1
10	7.038381	127.0.0.1	127.0.0.1	TCP	47	56673 → 2222 [PSH, ACK] Seq=4 Ack=19 Win=1..	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 Win=1..	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 Win=1..	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 Win=1..	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 Win=1..	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 Win=1..	1
21	7.045088	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 Win=1..	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 Win=1..	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 Win=1..	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 Win=1..	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.

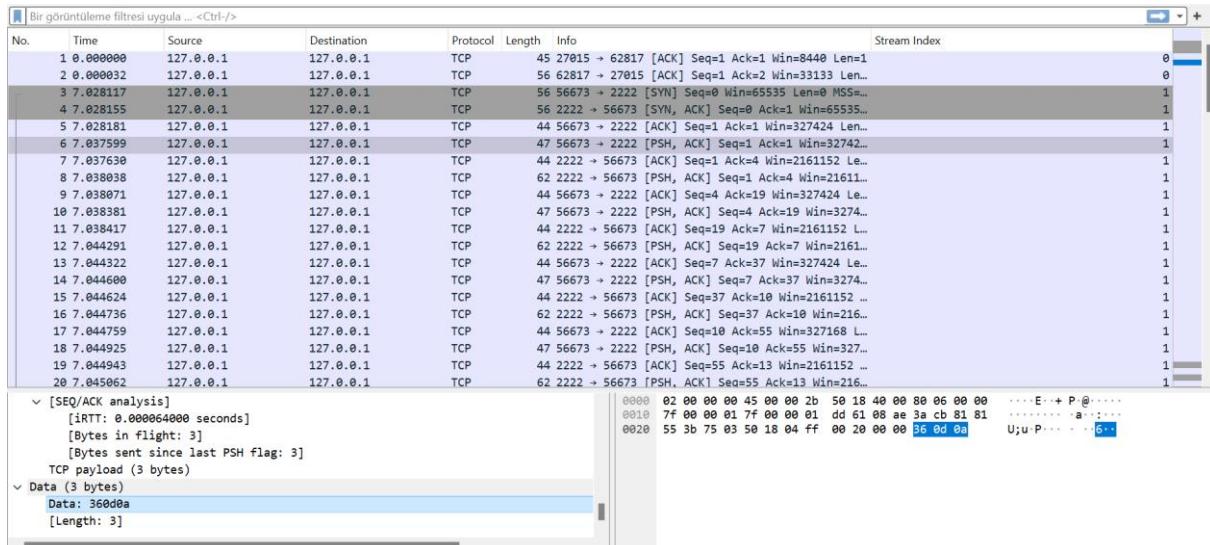


Figure 15. First data sent in TCP

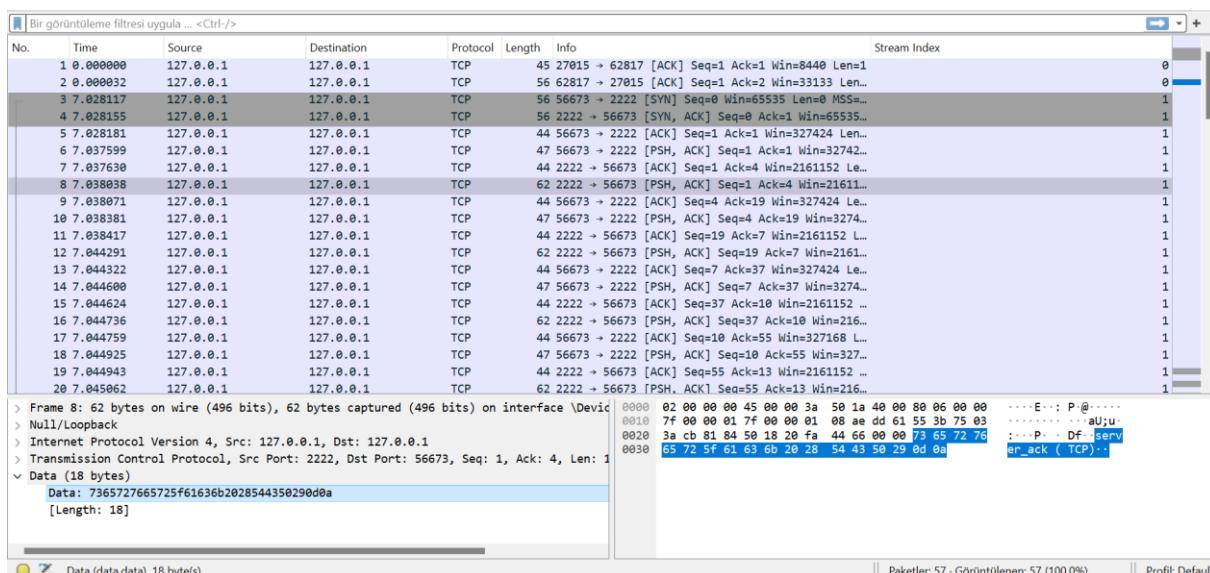


Figure 16. Response to the first data in TCP

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	I	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	7.038871	127.0.0.1	127.0.0.1	TCP	44	56673 → 2222 [ACK] Seq=4 Ack=19 Win=327...		1	
10	7.038881	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	7.045088	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	7.046738	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 be sent as in TCP case, we can't see any clue about the data, instead we see an information stating that the data is encrypted 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	

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.

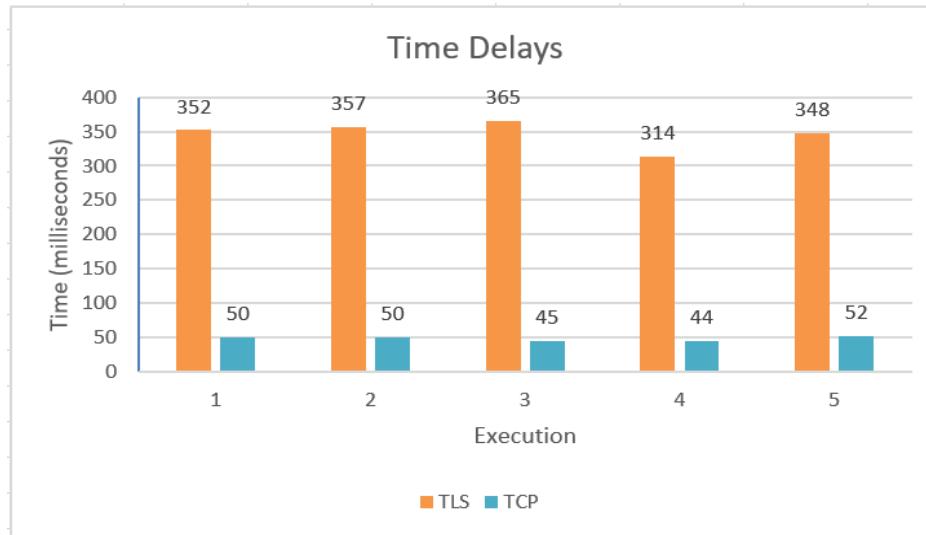


Figure 19. Time Delay Graph of TLS and TCP Executions

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).

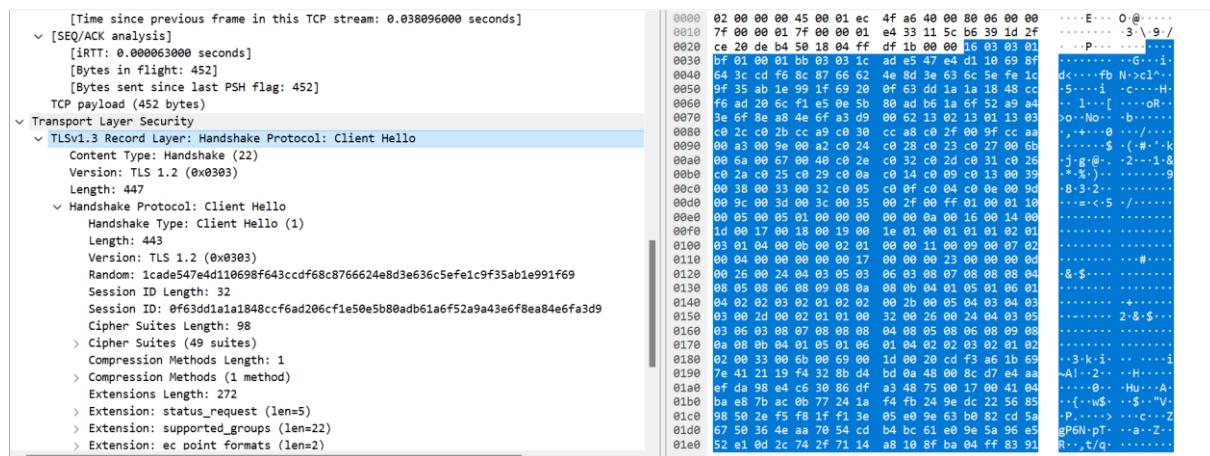


Figure 20. Client Hello Packet

```

> Frame 10: 171 bytes on wire (1368 bits), 171 bytes captured (1368 bits) on interface \Device\NPF_{...}
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 4444, Dst Port: 58419, Seq: 1, Ack: 453, Len: 127
  > Transport Layer Security
    > TLSv1.3 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 122
      > Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 118
        Version: TLS 1.2 (0x0303)
        Random: 9700c9e95edc3dc1fc466f9abef064ecd3bce8a115684527db3939fd099e099e
        Session ID Length: 32
        Session ID: 0f63dd1a1a1848ccf6ad206cf1e50e5b80adb1a6f52a9a43e6f8ea84e6fa3d9
        Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
        Compression Method: null (0)
        Extensions Length: 46
        > Extension: supported_versions (len=2) TLS 1.3
        > Extension: key_share (len=36) x25519
          [JA3S FullString: 771,4866,43-51]
          [JA3S: 15af977ce25de452b96affa2addb1036]

```

Hex dump of the Server Hello packet:

```

0000  02 00 00 00 45 00 00 a7 4f a8 40 00 80 06 00 00
0010  7f 00 00 b1 7f 00 00 b1 11 5c e4 33 ce 20 de b4
0020  b6 39 1e f3 50 18 20 f8 bd db 00 00 16 03 03 00
0030  7a 02 00 00 75 03 03 97 00 c9 e9 5e dc 3d c1 fc
0040  46 6f 9a be f0 64 ec d3 bc e8 a1 15 68 45 27 db
0050  39 39 fd 09 9e 99 9e 20 0f 63 dd 1a 1a 18 48 cc
0060  f6 ad 20 6c f1 e5 0e 5b 80 ad b6 1a 6f 52 a9 a4
0070  3e 6f 8e a8 4e 6f a3 d9 13 02 00 00 2e 00 2b 00
0080  02 03 04 00 33 00 24 00 1d 00 20 f4 4a 08 41 61
0090  88 0e ea b8 c3 7c 38 ce 92 43 83 fb 7d 77 95 42
00a0  9e b2 c2 be d5 b3 2d e3 98 a2 09

```

Figure 21. Server Hello Packet

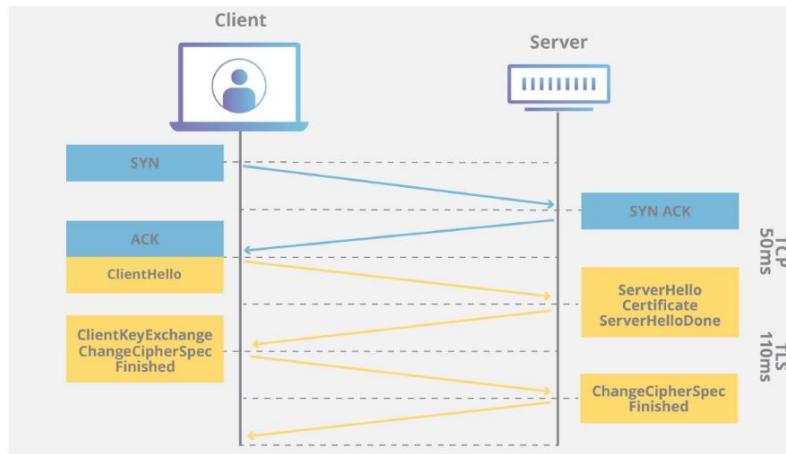


Figure 22. TLS Handshake

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 encryption, 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.

```

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 58419, Dst Port: 4444, Seq: 1, Ack: 1, Len: 452
< Transport Layer Security
  < TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 447
  < Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 443
    Version: TLS 1.2 (0x0303)
    Random: 1eade547ed4d110698f643ccdf68c8766624e8d3e6365afe1c9f35ab1e991f69
    Session ID Length: 32
    Session ID: 0f63dd11a1848ccf6ad206cf1e50e5b80adb61a6f52a9a43e6f8ea84e6fa3d9
    Cipher Suites Length: 98
    > Cipher Suites (49 suites)
    Compression Methods Length: 1
    > Compression Methods (1 method)
    Extensions Length: 272
    > Extension: status_request (len=5)
    > Extension: supported_groups (len=22)
    > Extension: ec_point_formats (len=2)
    > Extension: status_request_v2 (len=9)
    > Extension: extended_master_secret (len=0)
    > Extension: session_ticket (len=0)
    > Extension: signature_algorithms (len=38)
    > Extension: supported_versions (len=5) TLS 1.3, TLS 1.2

```

0000 02 00 00 00 45 00 01 ec 4f a6 40 00 80 06 00 00E.....O@....
0010 7f 00 00 01 7f 00 00 01 e4 33 11 5c b3 39 1d 2f 3\.....P.....
0020 c0 2e 00 b4 50 18 04 ff e4 8d 3e 63 5c fe 1c ..G:i.....
0030 bf 01 00 01 bb 03 03 1c ad e5 47 e4 d1 10 69 8f d...fb N>c1..
0040 64 3c cd f6 8c 87 66 62 4e 8d 3e 63 5c fe 1c 5.....i.....
0050 97 35 ab 99 1f 69 20 0f 63 dd 1a 1a 18 48 cc 1.....c-H.....
0060 f6 ad 20 6c f1 e5 0e 5b 80 ad b6 1a 6f 52 a9 a4 ..1...[...-OR..
0070 3e 6f 8e a8 4e 6f a3 d9 0e 62 13 02 13 01 13 03 >>No...b.....
0080 c8 2c c8 2b cc a9 c8 30 cc a8 c0 2f 00 9f cc aa ,+...0.../.
0090 00 a3 00 00 a2 c8 24 c8 28 c3 c8 27 00 6b ,...\$-(#-k.....
00a0 00 6a 00 67 00 40 c8 2e c8 32 02 d8 c8 31 c8 26 jg@...-2--1&.....
00b0 c8 2a c8 25 c8 29 c8 0a c8 14 c8 09 c8 13 00 39 .%"...)......9
00c0 00 38 00 33 00 32 c8 05 c8 0f c8 04 c8 0e 00 9d 8-3-2.....
00d0 00 9c 00 3d 00 3c 00 35 0e 2f 00 ff 01 00 01 10 =-<5 /.....
00e0 00 05 00 05 01 00 00 00 00 00 00 00 00 00 00 00<5 /.....
00f0 1d 00 17 00 18 00 19 00 1e 01 00 01 01 02 01#.....
0100 03 01 04 00 0b 00 02 01 00 11 00 09 00 07 02&-\$.....
0110 00 04 00 00 00 00 00 17 00 00 23 00 00 00 0d+.....
0120 00 26 00 24 04 03 05 03 06 03 08 07 08 08 08 042-8\$.....
0130 08 05 08 05 08 09 08 08 08 0b 04 01 05 01 06 01+.....
0140 04 02 02 03 02 01 02 02 00 2b 00 05 04 03 04 03+.....
0150 03 00 2d 00 02 01 01 00 32 00 26 00 24 04 03 052-8\$.....
0160 03 06 03 00 07 08 08 08 08 08 05 08 06 08 09 08+.....
0170 0a 08 0b 04 01 05 01 06 01 04 02 02 03 02 01 023-k-i.....i.....
0180 02 00 33 00 6b 00 69 00 1d 00 20 cd f3 a6 1b 69A1-2-.....H.....
0190 7e 41 21 19 f4 32 8b d4 bd 0a 48 00 8c d7 e4 aa ..-0...-HU-A.....
01a0 ef da 98 e4 c6 38 08 df a3 48 75 00 17 00 41 04-w\$-...\$-V.....
01b0 ba 8b 7b ac 0b 77 24 1a f4 fb 24 9e dc 22 56 85P....>...-c...Z.....
01c0 98 50 2e f5 f8 1f f1 3e 05 e8 9e 63 b8 82 cd 5agPNpt...-a...Z.....
01d0 67 56 36 4a aa 70 54 cd b4 bc 61 e8 9e 5a 9e e5R`...,t/q.....
01e0 52 e1 0d 2c 74 2f 71 14 a8 10 8f ba 04 ff 83 91 R`...,t/q.....

Figure 23. Content of Client Hello

```

> Frame 10: 171 bytes on wire (1368 bits), 171 bytes captured (1368 bits) on interface \Device\Null\Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 4444, Dst Port: 58419, Seq: 1, Ack: 453, Len: 127
< Transport Layer Security
  < TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 122
  < Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: 9700c9e95edc3dc1fc466f9abef064ecd3bce8a115684527db3939fd099e099e
    Session ID Length: 32
    Session ID: 0f63dd11a1848ccf6ad206cf1e50e5b80adb61a6f52a9a43e6f8ea84e6fa3d9
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Compression Method: null (0)
    Extensions Length: 46
    > Extension: supported_versions (len=2) TLS 1.3
    > Extension: key_share (len=36) x25519
      [JA3S FullString: 771,4866,43-51]
      [JA3S: 15af977ce25de452b96affa2addb1036]

```

0000 02 00 00 00 45 00 00 a7 4f a8 40 00 80 06 00 00E.....O@....
0010 7f 00 00 01 7f 00 00 01 11 5c e4 33 ce 20 de b4 3\.....P.....
0020 b6 39 1e f3 50 18 20 f8 bd db 00 00 16 03 02 00 9.....
0030 7a 02 00 00 76 03 03 97 00 c9 e9 5e dc 3d c1 fc z...v.....^=..
0040 46 6f 9a be 00 64 ec d3 bc a8 a1 15 68 45 27 db Fo...d.....he'.....
0050 39 39 fd 09 9e 20 0f 63 dd 1a 1a 18 48 cc 99...c.....H.....
0060 f6 ad 20 6c f1 e5 0e 5b 80 ad b6 1a 6f 52 a9 a4 ..1...[...-OR..
0070 3e 6f 8e a8 4e 6f a3 d9 13 02 00 00 24 00 0d 00 2f 44 08 41 61 >>No...+.
0080 02 03 04 00 33 00 24 00 1d 00 2f 44 08 41 61 ..3 \$...J Aa|8...C...jw-B.....
0090 88 0e ea 0b c3 7c 38 ce 92 00 00 83 fb 7d 77 95 42|8...C...jw-B.....
00a0 9e 62 c2 be d5 b3 2d e3 98 a2 00|8...C...jw-B.....

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".

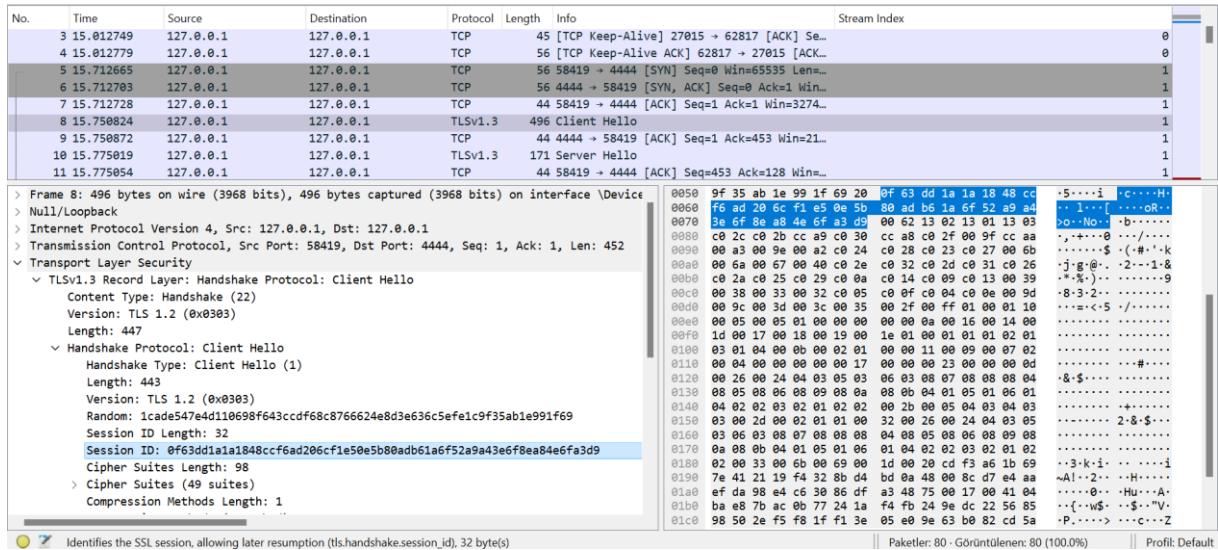


Figure 25. Session Parameters for later Resumption

Part 2.2 Creating Keystore and Trust Store

After following the instructions 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.

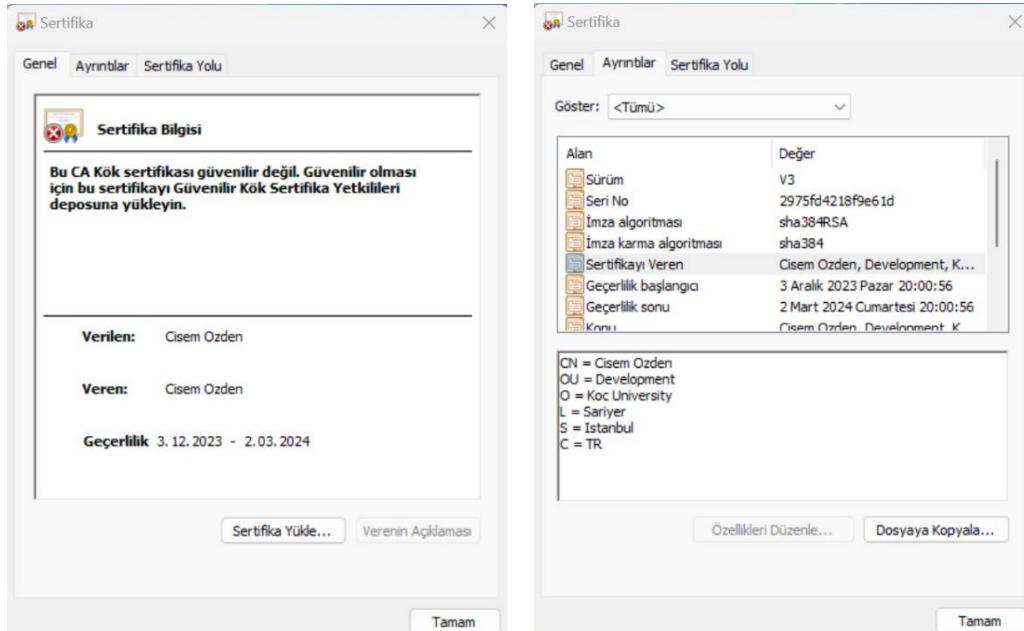


Figure 26. Certificate

References

- <https://www.cloudflare.com/learning/>
- <https://osqa-ask.wireshark.org/questions/>