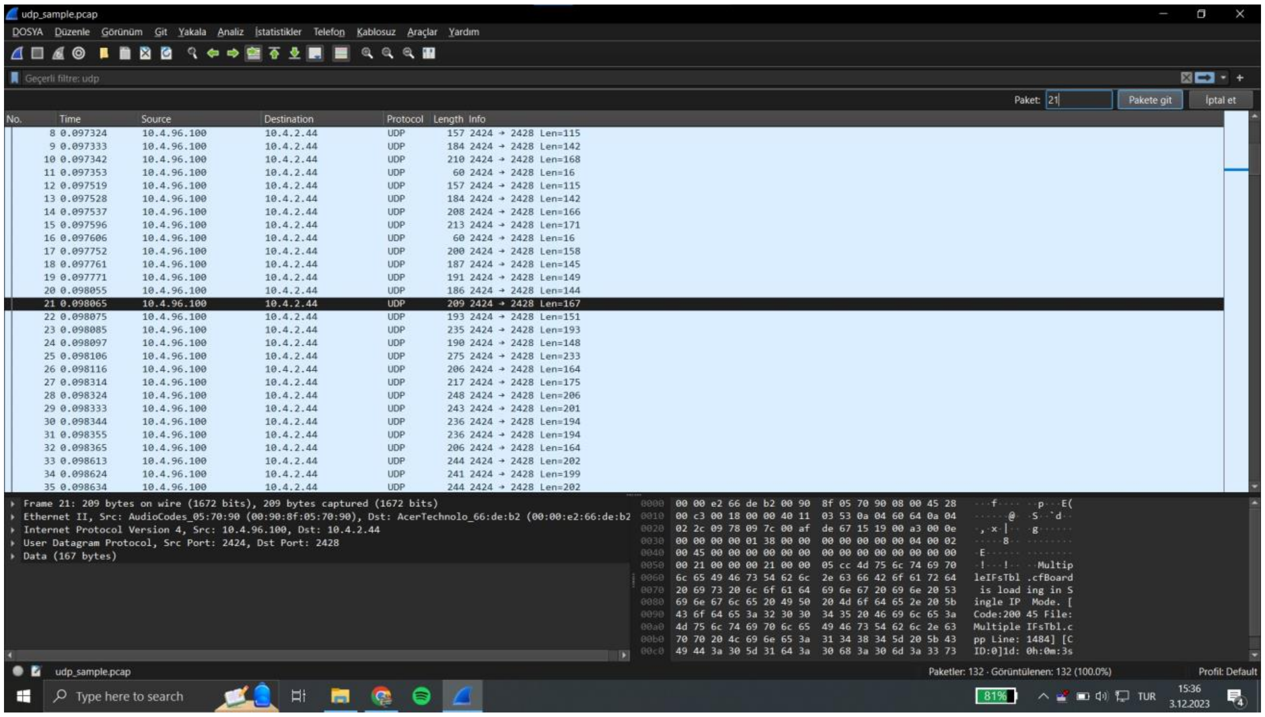
COMP416: Computer Networks
Project 2
Mehmet Eren Kılıç – 76621

Part 1-1)

Firstly, I have found the segment that I will work on:



```
No.  Time      Source       Destination  Protocol Length Info
   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
  14 0.097537  10.4.96.100  10.4.2.44    UDP      208 2424 → 2428 Len=166
  15 0.097596  10.4.96.100  10.4.2.44    UDP      213 2424 → 2428 Len=171
  16 0.097606  10.4.96.100  10.4.2.44    UDP       60 2424 → 2428 Len=16
  17 0.097752  10.4.96.100  10.4.2.44    UDP      200 2424 → 2428 Len=158
  18 0.097761  10.4.96.100  10.4.2.44    UDP      187 2424 → 2428 Len=145
  19 0.097771  10.4.96.100  10.4.2.44    UDP      191 2424 → 2428 Len=149
  20 0.098055  10.4.96.100  10.4.2.44    UDP      186 2424 → 2428 Len=144
  21 0.098065  10.4.96.100  10.4.2.44    UDP      209 2424 → 2428 Len=167
  22 0.098075  10.4.96.100  10.4.2.44    UDP      193 2424 → 2428 Len=151
  23 0.098085  10.4.96.100  10.4.2.44    UDP      235 2424 → 2428 Len=193
  24 0.098097  10.4.96.100  10.4.2.44    UDP      190 2424 → 2428 Len=148
  25 0.098106  10.4.96.100  10.4.2.44    UDP      275 2424 → 2428 Len=233
  26 0.098116  10.4.96.100  10.4.2.44    UDP      206 2424 → 2428 Len=164
  27 0.098314  10.4.96.100  10.4.2.44    UDP      217 2424 → 2428 Len=175
  28 0.098324  10.4.96.100  10.4.2.44    UDP      248 2424 → 2428 Len=206
  29 0.098333  10.4.96.100  10.4.2.44    UDP      243 2424 → 2428 Len=201
  30 0.098344  10.4.96.100  10.4.2.44    UDP      236 2424 → 2428 Len=194
  31 0.098355  10.4.96.100  10.4.2.44    UDP      236 2424 → 2428 Len=194
  32 0.098365  10.4.96.100  10.4.2.44    UDP      206 2424 → 2428 Len=164
  33 0.098613  10.4.96.100  10.4.2.44    UDP      244 2424 → 2428 Len=202
  34 0.098624  10.4.96.100  10.4.2.44    UDP      241 2424 → 2428 Len=199
  35 0.098634  10.4.96.100  10.4.2.44    UDP      244 2424 → 2428 Len=202
```

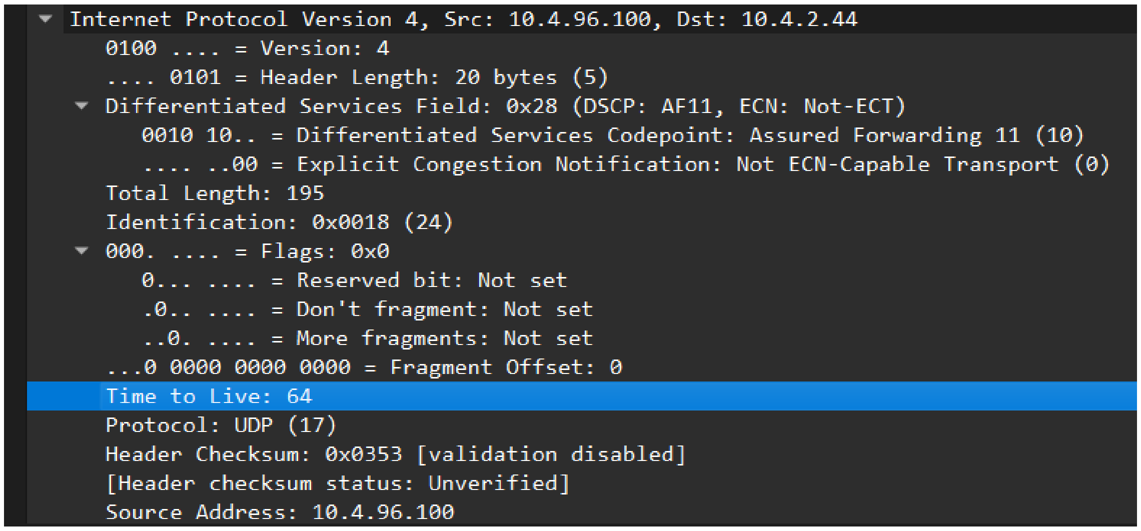
```
> Frame 21: 209 bytes on wire (1672 bits), 209 bytes captured (1672 bits)
> Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:00:e2:66:de:b2
> Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
> User Datagram Protocol, Src Port: 2424, Dst Port: 2428
> Data (167 bytes)
```

```
0000  00 00 e2 66 de b2 00 90  8f 05 70 90 08 00 45 28   ···f···· ··p···E(
0010  00 c3 00 18 00 00 40 11  03 53 0a 04 60 64 0a 04   ······@· ·S··`d··
0020  02 2c 09 78 09 7c 00 af  de 67 15 19 00 a3 00 0e   ·,·x·|·· ·g······
0030  00 00 00 00 01 38 00 00  00 00 00 00 00 04 00 02   ·····8·· ········
0040  00 45 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ·E······ ········
0050  00 21 00 00 00 21 00 00  05 cc 4d 75 6c 74 69 70   ·!···!·· ··Multip
0060  6c 65 49 46 73 54 62 6c  2e 63 66 42 6f 61 72 64   leIFsTbl .cfBoard
0070  20 69 73 20 6c 6f 61 64  69 6e 67 20 69 6e 20 53    is load ing in S
0080  69 6e 67 6c 65 20 49 50  20 4d 6f 64 65 2e 20 5b   ingle IP  Mode. [
0090  43 6f 64 65 3a 32 30 30  34 35 20 46 69 6c 65 3a   Code:200 45 File:
00a0  4d 75 6c 74 69 70 6c 65  49 46 73 54 62 6c 2e 63   Multiple IFsTbl.c
00b0  70 70 20 4c 69 6e 65 3a  31 34 38 34 5d 20 5b 43   pp Line: 1484] [C
00c0  49 44 3a 30 5d 31 64 3a  30 68 3a 30 6d 3a 33 73   ID:0]1d: 0h:0m:3s
```

```
udp_sample.pcap                    Paketler: 132 · Görüntülenen: 132 (100.0%)
```

Figure 1: Finding the packet I will work on 1.1/0.png

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?

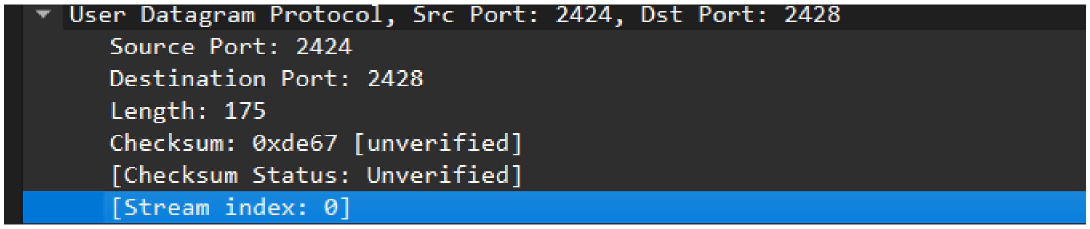


```
▾ Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▾ Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
        0010 10.. = Differentiated Services Codepoint: Assured Forwarding 11 (10)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
     Total Length: 195
     Identification: 0x0018 (24)
   ▾ 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)
     Header Checksum: 0x0353 [validation disabled]
     [Header checksum status: Unverified]
     Source Address: 10.4.96.100
```

Figure 2: Time to Live information of the packet 1.1/1.png

Time to live is 64. This means that the maximum number that packet can hop. In each hop it decreases by one, and if it is zero router drops the packet and maybe sends message to the sender about the failure. It is used to prevent the packet to circulating indefinetely and avoid congestion.

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

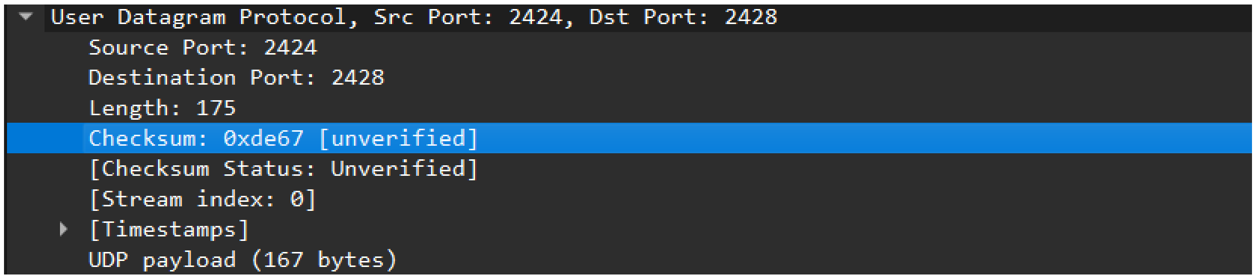


```
▾ User Datagram Protocol, Src Port: 2424, Dst Port: 2428
     Source Port: 2424
     Destination Port: 2428
     Length: 175
     Checksum: 0xde67 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
```

Figure 3: Stream index of the packet 1.1/2.png

Stream indexes are used to grouping packets starting from 0. For example my stream index is 0, means something like it is the part of the only conversation between the sockets or in the first conversation group.

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?

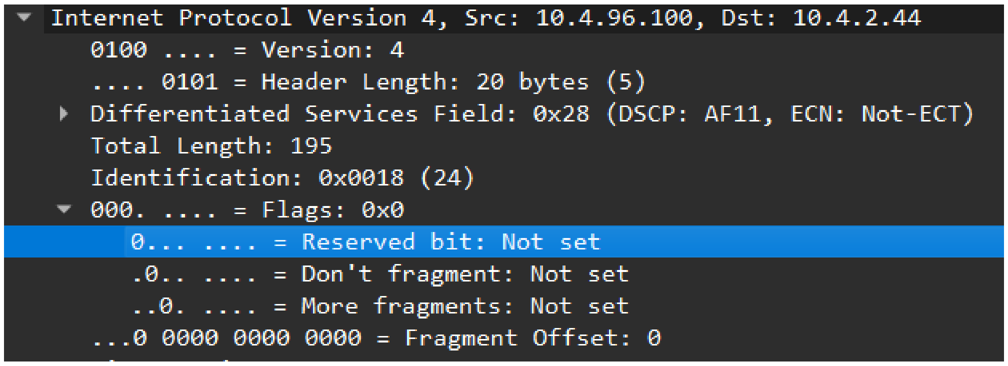


```
▾ User Datagram Protocol, Src Port: 2424, Dst Port: 2428
     Source Port: 2424
     Destination Port: 2428
     Length: 175
     Checksum: 0xde67 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   ▸ [Timestamps]
     UDP payload (167 bytes)
```

Figure 4: Checksum value of the packet 1.1/3.png

It is 0xde67, can be found under the User Datagram Protocol. Checksum is used between communication to ensure the data integrity. TCP and UDP has checksum functionality however unlike TCP, UDP does not have a recovery system for corrupted or altered packets. Thus, if the checksum value of the sender and the receiver are different, there is a data loss or corruption. In UDP the package might be discarded or handled by application layer. In TCP, the protocol might retransmit the data to ensure integrity.

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?

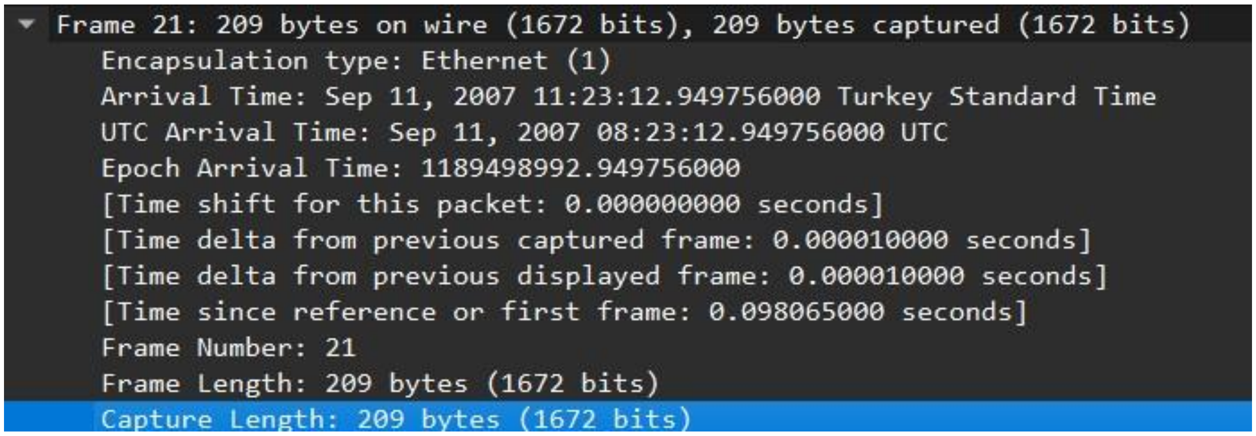


```
Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
    Total Length: 195
    Identification: 0x0018 (24)
  000. .... = Flags: 0x0
        0... .... = Reserved bit: Not set
        .0.. .... = Don't fragment: Not set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
```

Figure 5: Flags of the packet 1.1/4.png

It is not set, which means 0. Purpose of the reserved bit flag is ensuring the mechanism and controlling the behavior of the IP packet. It is 0 for legitimate packets, 1 for malicious contents.

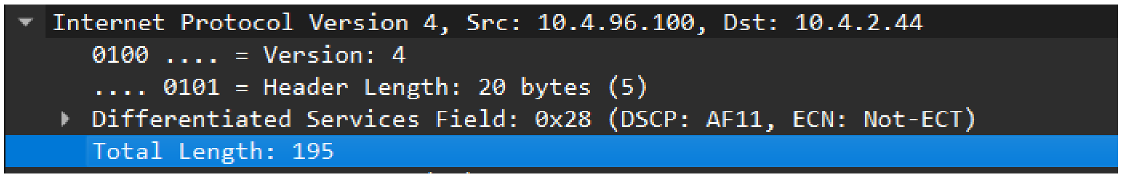
5. Explain the Length field. What does this value signify? Provide your packet's length.



```
Frame 21: 209 bytes on wire (1672 bits), 209 bytes captured (1672 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Sep 11, 2007 11:23:12.949756000 Turkey Standard Time
    UTC Arrival Time: Sep 11, 2007 08:23:12.949756000 UTC
    Epoch Arrival Time: 1189498992.949756000
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 0.000010000 seconds]
    [Time delta from previous displayed frame: 0.000010000 seconds]
    [Time since reference or first frame: 0.098065000 seconds]
    Frame Number: 21
    Frame Length: 209 bytes (1672 bits)
    Capture Length: 209 bytes (1672 bits)
```

Figure 6: Frame Capture length of the packet 1.1/5.1.png

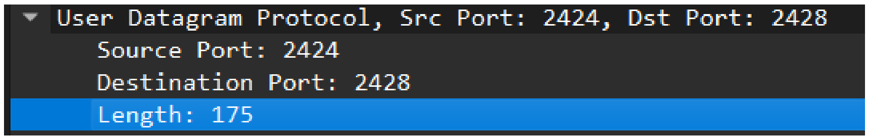
The frame length is 209 bytes, which is the total length of the packet in network interface level.



```
Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
    Total Length: 195
```

Figure 7: IPv4 length of the packet 1.1/5.2.png

Total length of the IP packet is 195 bytes. Includes IP header and payload.



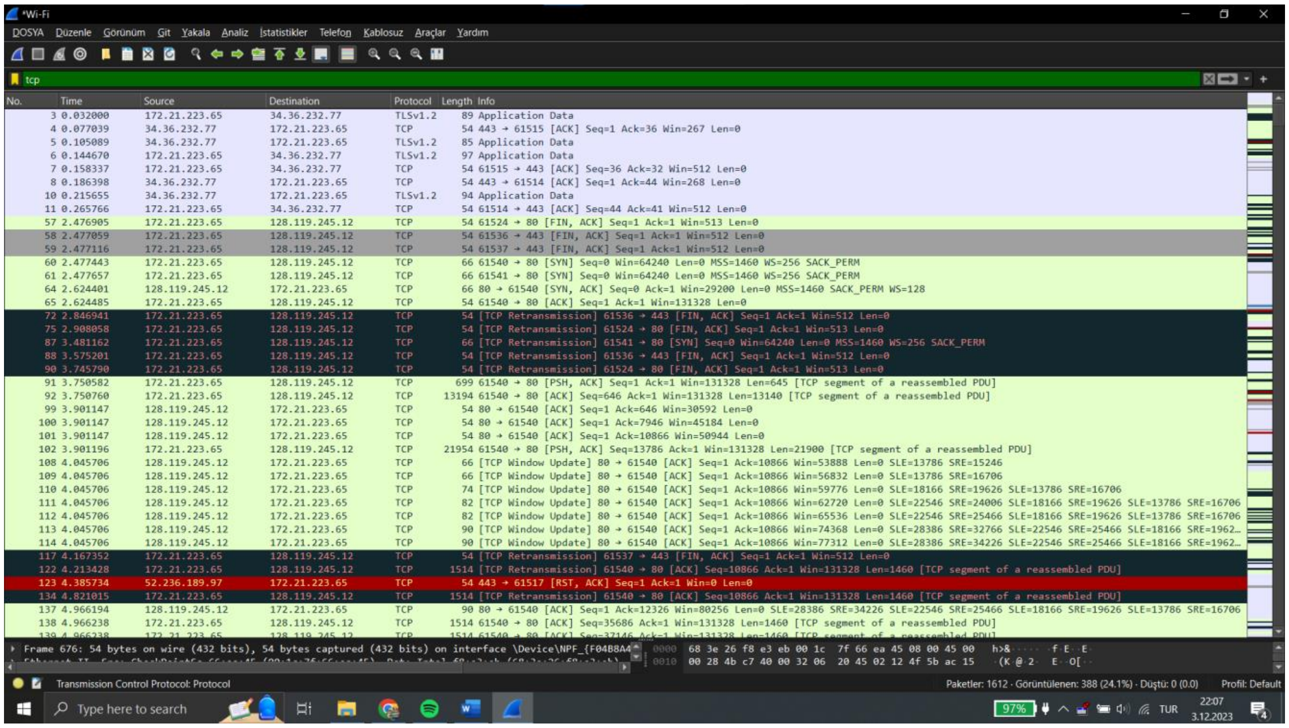
```
▼ User Datagram Protocol, Src Port: 2424, Dst Port: 2428
     Source Port: 2424
     Destination Port: 2428
     Length: 175
```

Figure 8: UDP length of the packet 1.1/5.3.png

Length of UDP datagram is 175 bytes. It includes UDP header and the actual data as payload. Values are getting smaller as packet goes to inner layers.

Part 1.2)

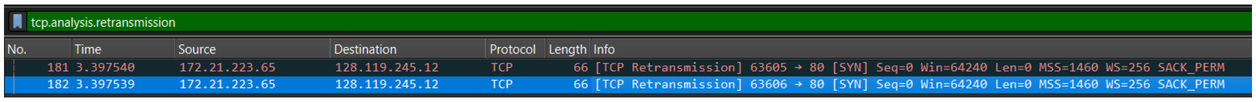
I have received the TCP logs from the Wireshark:



```
tcp
No.  Time      Source          Destination     Protocol Length Info
  3 0.032000   172.21.223.65   34.36.232.77    TLSv1.2     89 Application Data
  4 0.077039   34.36.232.77    172.21.223.65   TCP         54 443 → 61515 [ACK] Seq=1 Ack=36 Win=267 Len=0
  5 0.105089   34.36.232.77    172.21.223.65   TLSv1.2     85 Application Data
  6 0.144670   172.21.223.65   34.36.232.77    TLSv1.2     97 Application Data
  7 0.158337   172.21.223.65   34.36.232.77    TCP         54 61515 → 443 [ACK] Seq=36 Ack=32 Win=512 Len=0
  8 0.186398   34.36.232.77    172.21.223.65   TCP         54 443 → 61514 [ACK] Seq=1 Ack=44 Win=268 Len=0
 10 0.215655   34.36.232.77    172.21.223.65   TLSv1.2     94 Application Data
 11 0.265766   172.21.223.65   34.36.232.77    TCP         54 61514 → 443 [ACK] Seq=44 Ack=41 Win=512 Len=0
 57 2.476905   172.21.223.65   128.119.245.12  TCP         54 61524 → 80 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
 58 2.477059   172.21.223.65   128.119.245.12  TCP         54 61536 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
 59 2.477116   172.21.223.65   128.119.245.12  TCP         54 61537 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
 60 2.477443   172.21.223.65   128.119.245.12  TCP         66 61540 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 61 2.477657   172.21.223.65   128.119.245.12  TCP         66 61541 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 64 2.624401   128.119.245.12  172.21.223.65   TCP         66 80 → 61540 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM WS=128
 65 2.624485   172.21.223.65   128.119.245.12  TCP         54 61540 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
 72 2.846941   172.21.223.65   128.119.245.12  TCP         54 [TCP Retransmission] 61536 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
 75 2.908058   172.21.223.65   128.119.245.12  TCP         54 [TCP Retransmission] 61524 → 80 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
 87 3.481162   172.21.223.65   128.119.245.12  TCP         66 [TCP Retransmission] 61541 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 88 3.575201   172.21.223.65   128.119.245.12  TCP         54 [TCP Retransmission] 61536 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
 90 3.745790   172.21.223.65   128.119.245.12  TCP         54 [TCP Retransmission] 61524 → 80 [FIN, ACK] Seq=1 Ack=1 Win=513 Len=0
 91 3.750582   172.21.223.65   128.119.245.12  TCP        699 61540 → 80 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=645 [TCP segment of a reassembled PDU]
 92 3.750760   172.21.223.65   128.119.245.12  TCP      13194 61540 → 80 [ACK] Seq=646 Ack=1 Win=131328 Len=13140 [TCP segment of a reassembled PDU]
 99 3.901147   128.119.245.12  172.21.223.65   TCP         54 80 → 61540 [ACK] Seq=1 Ack=646 Win=30592 Len=0
100 3.901147   128.119.245.12  172.21.223.65   TCP         54 80 → 61540 [ACK] Seq=1 Ack=7946 Win=45184 Len=0
101 3.901147   128.119.245.12  172.21.223.65   TCP         54 80 → 61540 [ACK] Seq=1 Ack=10866 Win=50944 Len=0
102 3.901196   172.21.223.65   128.119.245.12  TCP      21954 61540 → 80 [PSH, ACK] Seq=13786 Ack=1 Win=131328 Len=21900 [TCP segment of a reassembled PDU]
108 4.045706   128.119.245.12  172.21.223.65   TCP         66 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=53888 Len=0 SLE=13786 SRE=15246
109 4.045706   128.119.245.12  172.21.223.65   TCP         66 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=56832 Len=0 SLE=13786 SRE=16706
110 4.045706   128.119.245.12  172.21.223.65   TCP         74 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=59776 Len=0 SLE=18166 SRE=19626 SLE=13786 SRE=16706
111 4.045706   128.119.245.12  172.21.223.65   TCP         82 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=62720 Len=0 SLE=22546 SRE=24006 SLE=18166 SRE=19626 SLE=13786 SRE=16706
112 4.045706   128.119.245.12  172.21.223.65   TCP         82 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=65536 Len=0 SLE=22546 SRE=25466 SLE=18166 SRE=19626 SLE=13786 SRE=16706
113 4.045706   128.119.245.12  172.21.223.65   TCP         90 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=74368 Len=0 SLE=28386 SRE=32766 SLE=22546 SRE=25466 SLE=18166 SRE=1962…
114 4.045706   128.119.245.12  172.21.223.65   TCP         90 [TCP Window Update] 80 → 61540 [ACK] Seq=1 Ack=10866 Win=77312 Len=0 SLE=28386 SRE=34226 SLE=22546 SRE=25466 SLE=18166 SRE=1962…
117 4.167352   172.21.223.65   128.119.245.12  TCP         54 [TCP Retransmission] 61537 → 443 [FIN, ACK] Seq=1 Ack=1 Win=512 Len=0
122 4.213428   172.21.223.65   128.119.245.12  TCP       1514 [TCP Retransmission] 61540 → 80 [ACK] Seq=10866 Ack=1 Win=131328 Len=1460 [TCP segment of a reassembled PDU]
123 4.385734   52.236.189.97   172.21.223.65   TCP         54 443 → 61517 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
134 4.821015   172.21.223.65   128.119.245.12  TCP       1514 [TCP Retransmission] 61540 → 80 [ACK] Seq=10866 Ack=1 Win=131328 Len=1460 [TCP segment of a reassembled PDU]
137 4.966194   128.119.245.12  172.21.223.65   TCP         90 80 → 61540 [ACK] Seq=1 Ack=12326 Win=80256 Len=0 SLE=28386 SRE=34226 SLE=22546 SRE=25466 SLE=18166 SRE=19626 SLE=13786 SRE=16706
138 4.966238   172.21.223.65   128.119.245.12  TCP       1514 61540 → 80 [ACK] Seq=35686 Ack=1 Win=131328 Len=1460 [TCP segment of a reassembled PDU]
139 4.966238   172.21.223.65   128.119.245.12  TCP       1514 61540 → 80 [ACK] Seq=37146 Ack=1 Win=131328 Len=1460 [TCP segment of a reassembled PDU]

Frame 676: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{F04B8A4
0000  68 3e 26 f8 e3 eb 00 1c  7f 66 ea 45 08 00 45 00   h>&·····f·E··E·
0010  00 28 4b c7 40 00 32 06  20 45 02 12 4f 5b ac 15   ·(K·@·2· E··O[··

Transmission Control Protocol: Protocol      Paketler: 1612 · Görüntülenen: 388 (24.1%) · Düştü: 0 (0.0)   Profil: Default
```

Figure 9: *Capture logs of the TCP connection 1.2/0.png*

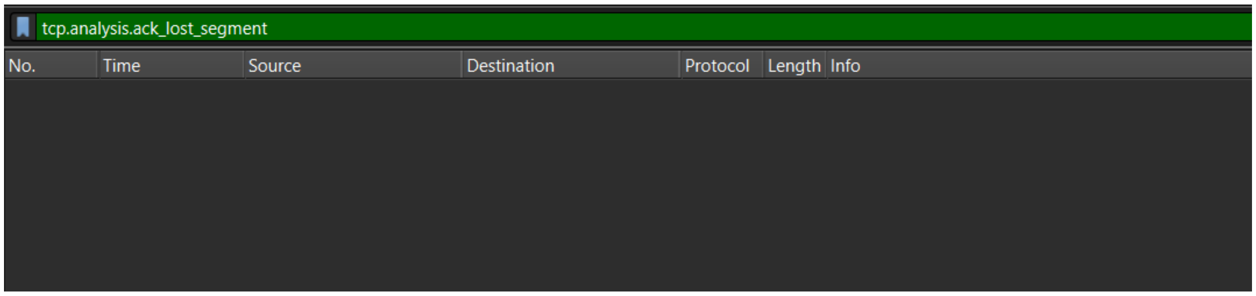
1. Can you identify any segments marked as retransmissions? Are there indications of segment loss in the TCP flow? (Hint: Use post-filters.)



```
tcp.analysis.retransmission
No.  Time      Source          Destination     Protocol Length Info
181 3.397540   172.21.223.65   128.119.245.12  TCP         66 [TCP Retransmission] 63605 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
182 3.397539   172.21.223.65   128.119.245.12  TCP         66 [TCP Retransmission] 63606 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
```

Figure 10: *retransmission logs 1.2/1.1.png*

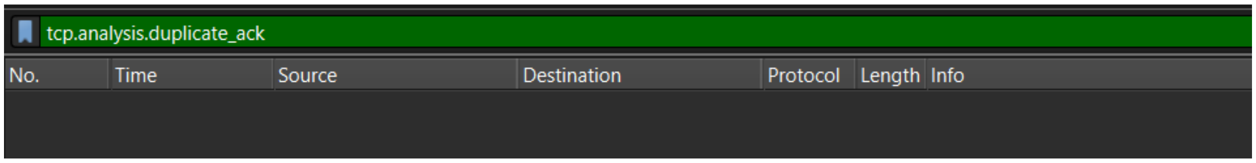
There are two retransmission in the communication, I could see by "tcp.analysis.retransmission" filter command. To find out segment loss I used "tcp.analysis.ack_lost_segment" but could not find one.



```
tcp.analysis.ack_lost_segment
No.  Time  Source  Destination  Protocol  Length  Info
```

Figure 11: *acknowledgment loss logs 1.2/1.2.png*

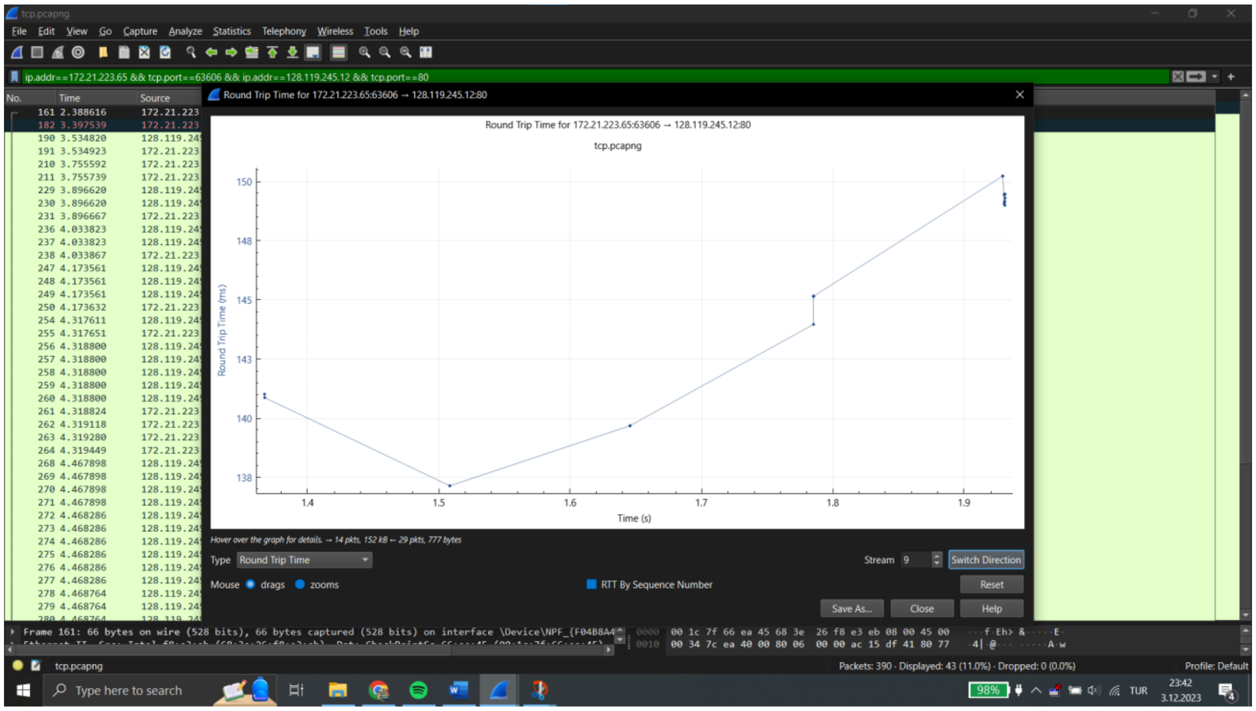
And there are no duplicate acknowledgements, I could not find any data loss. TCP could handle all issues by retransmissions.



```
tcp.analysis.duplicate_ack
No.  Time  Source  Destination  Protocol  Length  Info
```

Figure 12: duplicate acknowledgement logs 1.2/1.3.png

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.)

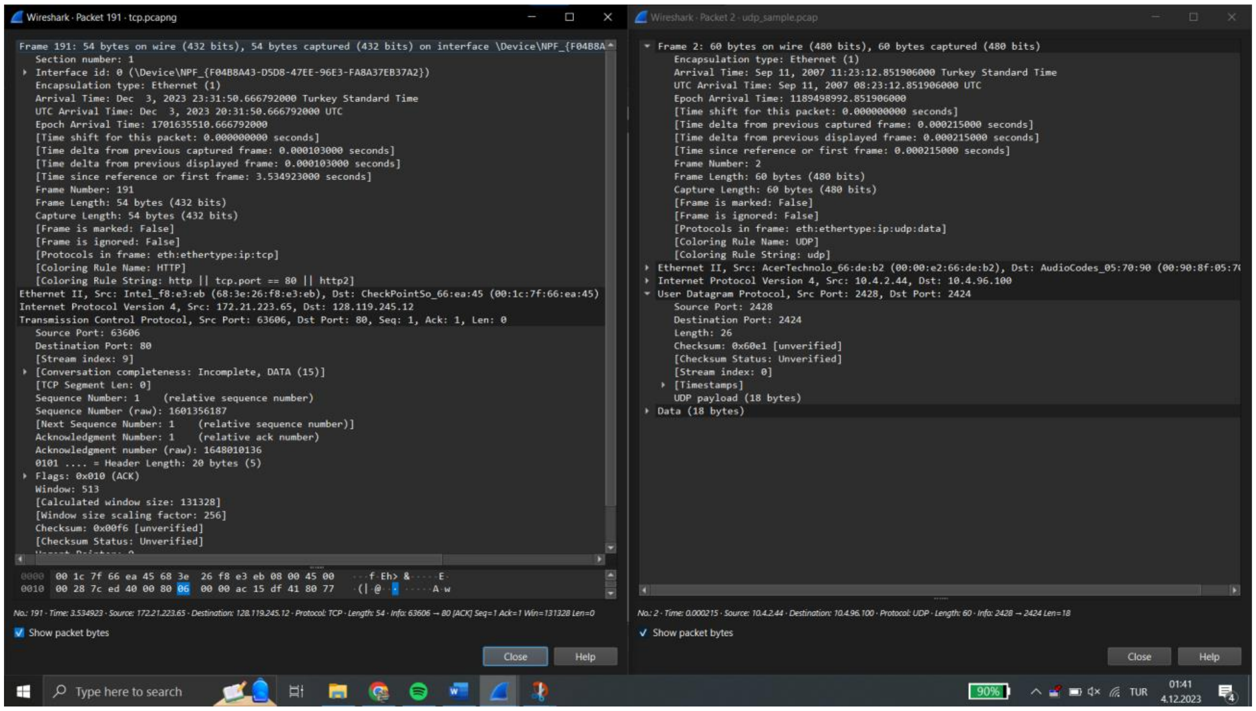


```
ip.addr==172.21.223.65 && tcp.port==63606 && ip.addr==128.119.245.12 && tcp.port==80

No.  Time      Source
161 2.388616   172.21.223
182 3.397539   172.21.223
190 3.534820   128.119.24
191 3.534923   172.21.223
210 3.755592   172.21.223
211 3.755739   172.21.223
229 3.896620   128.119.24
230 3.896620   128.119.24
231 3.896667   172.21.223
236 4.033823   128.119.24
237 4.033823   128.119.24
238 4.033867   172.21.223
247 4.173561   128.119.24
248 4.173561   128.119.24
249 4.173561   128.119.24
250 4.173632   172.21.223
254 4.317611   128.119.24
255 4.317651   172.21.223
256 4.318800   128.119.24
257 4.318800   128.119.24
258 4.318800   128.119.24
259 4.318800   128.119.24
260 4.318800   128.119.24
261 4.318824   172.21.223
262 4.319118   172.21.223
263 4.319280   172.21.223
264 4.319449   172.21.223
268 4.467898   128.119.24
269 4.467898   128.119.24
270 4.467898   128.119.24
271 4.467898   128.119.24
272 4.468286   128.119.24
273 4.468286   128.119.24
274 4.468286   128.119.24
275 4.468286   128.119.24
276 4.468286   128.119.24
277 4.468286   128.119.24
278 4.468764   128.119.24
279 4.468764   128.119.24

Round Trip Time for 172.21.223.65:63606 → 128.119.245.12:80
Hover over the graph for details. → 14 pkts, 152 kB ← 29 pkts, 777 bytes
Type Round Trip Time                                  Stream 9
Mouse ● drags ● zooms          RTT By Sequence Number

Frame 161: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{F04B8A4
0000  00 1c 7f 66 ea 45 68 3e  26 f8 e3 eb 08 00 45 00   ···f·Eh> &·····E·
0010  00 34 7c ea 40 00 80 06  00 00 ac 15 df 41 80 77   ·4|·@··· ·····A·w

tcp.pcapng   Packets: 390 · Displayed: 43 (11.0%) · Dropped: 0 (0.0%)   Profile: Default
```

Figure 13: RTT Chart of the connection 1.2/2.png

I have used the TCP connection with most packets from the conversation tab. The filter gives the conversion from A to B. Then I have found that RTT time decreased at first then linearly increases in 1.5 seconds to 1.9 seconds runtime. Then stays there for a while. There are some variations due to congestion and router changing etc.

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



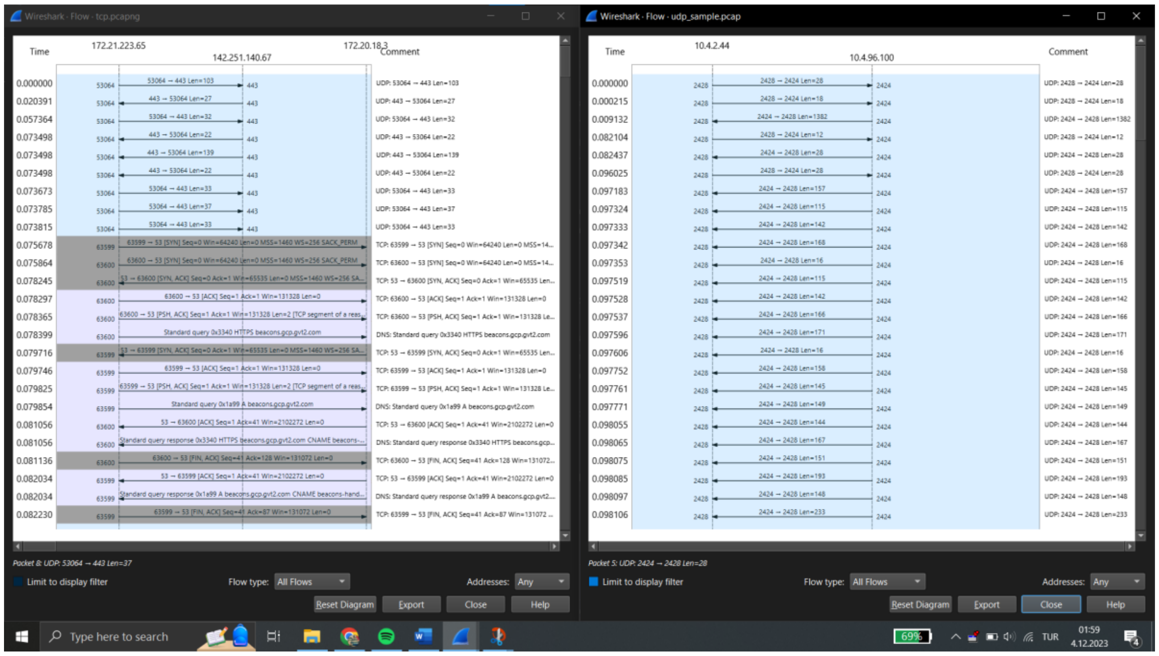
```
Frame 191: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{F04B8A
   Section number: 1
 ▸ Interface id: 0 (\Device\NPF_{F04B8A43-D5D8-47EE-96E3-FA8A37EB37A2})
   Encapsulation type: Ethernet (1)
   Arrival Time: Dec  3, 2023 23:31:50.666792000 Turkey Standard Time
   UTC Arrival Time: Dec  3, 2023 20:31:50.666792000 UTC
   Epoch Arrival Time: 1701635510.666792000
   [Time shift for this packet: 0.000000000 seconds]
   [Time delta from previous captured frame: 0.000103000 seconds]
   [Time delta from previous displayed frame: 0.000103000 seconds]
   [Time since reference or first frame: 3.534923000 seconds]
   Frame Number: 191
   Frame Length: 54 bytes (432 bits)
   Capture Length: 54 bytes (432 bits)
   [Frame is marked: False]
   [Frame is ignored: False]
   [Protocols in frame: eth:ethertype:ip:tcp]
   [Coloring Rule Name: HTTP]
   [Coloring Rule String: http || tcp.port == 80 || http2]
Ethernet II, Src: Intel_f8:e3:eb (68:3e:26:f8:e3:eb), Dst: CheckPointSo_66:ea:45 (00:1c:7f:66:ea:45)
Internet Protocol Version 4, Src: 172.21.223.65, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 63606, Dst Port: 80, Seq: 1, Ack: 1, Len: 0
   Source Port: 63606
   Destination Port: 80
   [Stream index: 9]
 ▸ [Conversation completeness: Incomplete, DATA (15)]
   [TCP Segment Len: 0]
   Sequence Number: 1    (relative sequence number)
   Sequence Number (raw): 1601356187
   [Next Sequence Number: 1    (relative sequence number)]
   Acknowledgment Number: 1    (relative ack number)
   Acknowledgment number (raw): 1648010136
   0101 .... = Header Length: 20 bytes (5)
 ▸ Flags: 0x010 (ACK)
   Window: 513
   [Calculated window size: 131328]
   [Window size scaling factor: 256]
   Checksum: 0x00f6 [unverified]
   [Checksum Status: Unverified]

0000  00 1c 7f 66 ea 45 68 3e  26 f8 e3 eb 08 00 45 00   ···f·Eh> &·····E·
0010  00 28 7c ed 40 00 80 06  00 00 ac 15 df 41 80 77   ·(|·@·········A·w

No.: 191 · Time: 3.534923 · Source: 172.21.223.65 · Destination: 128.119.245.12 · Protocol: TCP · Length: 54 · Info: 63606 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
```

```
▾ Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
     Encapsulation type: Ethernet (1)
     Arrival Time: Sep 11, 2007 11:23:12.851906000 Turkey Standard Time
     UTC Arrival Time: Sep 11, 2007 08:23:12.851906000 UTC
     Epoch Arrival Time: 1189498992.851906000
     [Time shift for this packet: 0.000000000 seconds]
     [Time delta from previous captured frame: 0.000215000 seconds]
     [Time delta from previous displayed frame: 0.000215000 seconds]
     [Time since reference or first frame: 0.000215000 seconds]
     Frame Number: 2
     Frame Length: 60 bytes (480 bits)
     Capture Length: 60 bytes (480 bits)
     [Frame is marked: False]
     [Frame is ignored: False]
     [Protocols in frame: eth:ethertype:ip:udp:data]
     [Coloring Rule Name: UDP]
     [Coloring Rule String: udp]
▸ Ethernet II, Src: AcerTechnolo_66:de:b2 (00:00:e2:66:de:b2), Dst: AudioCodes_05:70:90 (00:90:8f:05:70
▸ Internet Protocol Version 4, Src: 10.4.2.44, Dst: 10.4.96.100
▾ User Datagram Protocol, Src Port: 2428, Dst Port: 2424
     Source Port: 2428
     Destination Port: 2424
     Length: 26
     Checksum: 0x60e1 [unverified]
     [Checksum Status: Unverified]
     [Stream index: 0]
   ▸ [Timestamps]
     UDP payload (18 bytes)
▸ Data (18 bytes)

No.: 2 · Time: 0.000215 · Source: 10.4.2.44 · Destination: 10.4.96.100 · Protocol: UDP · Length: 60 · Info: 2428 → 2424 Len=18
```

Figure 14: TCP and UDP information differences 1.2/3.png

TCP has interface ID to contain interface identifier, Sequence number and acknowledgement numbers to monitor the result of the communication, by checking the sequence number the acknowledgement success can be tracked. TCP also has window size to perform flow control, maintaining the size ready at the moment. TCP also has many flags to control the communication and ensure integrity. Also SEQ/ACK analysis to monitor the performance of the communication.

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.



```
Time      Comment
0.000000  UDP: 53064 → 443 Len=103
0.020391  UDP: 443 → 53064 Len=27
0.057364  UDP: 53064 → 443 Len=32
0.073498  UDP: 443 → 53064 Len=22
0.073498  UDP: 443 → 53064 Len=139
0.073498  UDP: 443 → 53064 Len=22
0.073673  UDP: 53064 → 443 Len=33
0.073785  UDP: 53064 → 443 Len=37
0.073815  UDP: 53064 → 443 Len=33
0.075678  TCP: 63599 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=14...
0.075864  TCP: 63600 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=14...
0.078245  TCP: 53 → 63600 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len...
0.078297  TCP: 63600 → 53 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.078365  TCP: 63600 → 53 [PSH, ACK] Seq=1 Ack=1 Win=131328 Le...
0.078399  DNS: Standard query 0x3340 HTTPS beacons.gcp.gvt2.com
0.079716  TCP: 53 → 63599 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len...
0.079746  TCP: 63599 → 53 [ACK] Seq=1 Ack=1 Win=131328 Len=0
0.079825  TCP: 63599 → 53 [PSH, ACK] Seq=1 Ack=1 Win=131328 Le...
0.079854  DNS: Standard query 0x1a99 A beacons.gcp.gvt2.com
0.081056  TCP: 53 → 63600 [ACK] Seq=1 Ack=41 Win=2102272 Len=0
0.081056  DNS: Standard query response 0x3340 HTTPS beacons.gcp...
0.081136  TCP: 63600 → 53 [FIN, ACK] Seq=41 Ack=128 Win=131072...
0.082034  TCP: 53 → 63599 [ACK] Seq=1 Ack=41 Win=2102272 Len=0
0.082034  DNS: Standard query response 0x1a99 A beacons.gcp.gvt2...
0.082230  TCP: 63599 → 53 [FIN, ACK] Seq=41 Ack=87 Win=131072 ...

Packet 8: UDP: 53064 → 443 Len=37
```

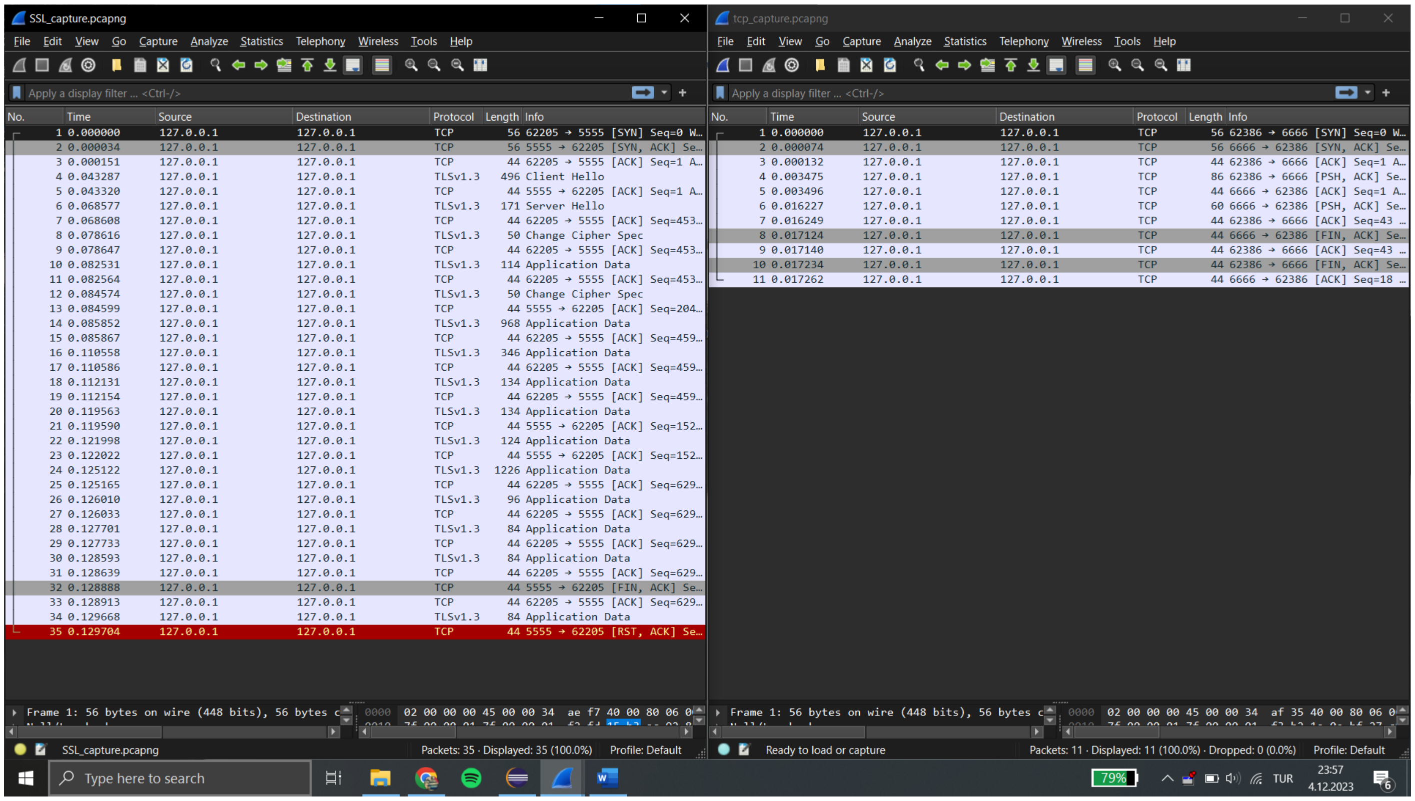
```
Time      Comment
0.000000  UDP: 2428 → 2424 Len=28
0.000215  UDP: 2428 → 2424 Len=18
0.009132  UDP: 2424 → 2428 Len=1382
0.082104  UDP: 2428 → 2424 Len=12
0.082437  UDP: 2424 → 2428 Len=28
0.096025  UDP: 2428 → 2424 Len=28
0.097183  UDP: 2424 → 2428 Len=157
0.097324  UDP: 2424 → 2428 Len=115
0.097333  UDP: 2424 → 2428 Len=142
0.097342  UDP: 2424 → 2428 Len=168
0.097353  UDP: 2424 → 2428 Len=16
0.097519  UDP: 2424 → 2428 Len=115
0.097528  UDP: 2424 → 2428 Len=142
0.097537  UDP: 2424 → 2428 Len=166
0.097596  UDP: 2424 → 2428 Len=171
0.097606  UDP: 2424 → 2428 Len=16
0.097752  UDP: 2424 → 2428 Len=158
0.097761  UDP: 2424 → 2428 Len=145
0.097771  UDP: 2424 → 2428 Len=149
0.098055  UDP: 2424 → 2428 Len=144
0.098065  UDP: 2424 → 2428 Len=167
0.098075  UDP: 2424 → 2428 Len=151
0.098085  UDP: 2424 → 2428 Len=193
0.098097  UDP: 2424 → 2428 Len=148
0.098106  UDP: 2424 → 2428 Len=233

Packet 5: UDP: 2424 → 2428 Len=28
```

Figure 15: TCP and UDP flow graphs 1.2/4.png

In TCP there are initialization of the communication like handshaking and many states to ensure that data being sent correctly. In UDP data is sent by datagrams without any acknowledgement or information about the integrity of the data. However, in TCP there are additional data aside from the payload about the status of the communications and necessary information to have a healthy connection like window size. Overall, TCP flows are more complex and have many additional information and flags to ensure the integrity, UDP does not have these standards. It sends the data directly to client without any control information or effort about the data.

Part 2-1)

I have finished the server and client part of the project. Then, run each client separately to see the traffic. I have used Adapter for loopback traffic capture part of the Wireshark capture mods. The results can be seen as follows:



```
SSL_capture.pcapng
No. Time      Source     Destination Protocol Length Info
  1 0.000000  127.0.0.1  127.0.0.1   TCP       56 62205 → 5555 [SYN] Seq=0 W…
  2 0.000034  127.0.0.1  127.0.0.1   TCP       56 5555 → 62205 [SYN, ACK] Se…
  3 0.000151  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=1 A…
  4 0.043287  127.0.0.1  127.0.0.1   TLSv1.3  496 Client Hello
  5 0.043320  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [ACK] Seq=1 A…
  6 0.068577  127.0.0.1  127.0.0.1   TLSv1.3  171 Server Hello
  7 0.068608  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=453…
  8 0.078616  127.0.0.1  127.0.0.1   TLSv1.3   50 Change Cipher Spec
  9 0.078647  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=453…
 10 0.082531  127.0.0.1  127.0.0.1   TLSv1.3  114 Application Data
 11 0.082564  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=453…
 12 0.084574  127.0.0.1  127.0.0.1   TLSv1.3   50 Change Cipher Spec
 13 0.084599  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [ACK] Seq=204…
 14 0.085852  127.0.0.1  127.0.0.1   TLSv1.3  968 Application Data
 15 0.085867  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=459…
 16 0.110558  127.0.0.1  127.0.0.1   TLSv1.3  346 Application Data
 17 0.110586  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=459…
 18 0.112131  127.0.0.1  127.0.0.1   TLSv1.3  134 Application Data
 19 0.112154  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=459…
 20 0.119563  127.0.0.1  127.0.0.1   TLSv1.3  134 Application Data
 21 0.119590  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [ACK] Seq=152…
 22 0.121998  127.0.0.1  127.0.0.1   TLSv1.3  124 Application Data
 23 0.122022  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [ACK] Seq=152…
 24 0.125122  127.0.0.1  127.0.0.1   TLSv1.3 1226 Application Data
 25 0.125165  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=629…
 26 0.126010  127.0.0.1  127.0.0.1   TLSv1.3   96 Application Data
 27 0.126033  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=629…
 28 0.127701  127.0.0.1  127.0.0.1   TLSv1.3   84 Application Data
 29 0.127733  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=629…
 30 0.128593  127.0.0.1  127.0.0.1   TLSv1.3   84 Application Data
 31 0.128639  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=629…
 32 0.128888  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [FIN, ACK] Se…
 33 0.128913  127.0.0.1  127.0.0.1   TCP       44 62205 → 5555 [ACK] Seq=629…
 34 0.129668  127.0.0.1  127.0.0.1   TLSv1.3   84 Application Data
 35 0.129704  127.0.0.1  127.0.0.1   TCP       44 5555 → 62205 [RST, ACK] Se…

Frame 1: 56 bytes on wire (448 bits), 56 bytes c
0000  02 00 00 00 45 00 00 34  ae f7 40 00 80 06 0
SSL_capture.pcapng    Packets: 35 · Displayed: 35 (100.0%)    Profile: Default
```

```
tcp_capture.pcapng
No. Time      Source     Destination Protocol Length Info
  1 0.000000  127.0.0.1  127.0.0.1   TCP       56 62386 → 6666 [SYN] Seq=0 W…
  2 0.000074  127.0.0.1  127.0.0.1   TCP       56 6666 → 62386 [SYN, ACK] Se…
  3 0.000132  127.0.0.1  127.0.0.1   TCP       44 62386 → 6666 [ACK] Seq=1 A…
  4 0.003475  127.0.0.1  127.0.0.1   TCP       86 62386 → 6666 [PSH, ACK] Se…
  5 0.003496  127.0.0.1  127.0.0.1   TCP       44 6666 → 62386 [ACK] Seq=1 A…
  6 0.016227  127.0.0.1  127.0.0.1   TCP       60 6666 → 62386 [PSH, ACK] Se…
  7 0.016249  127.0.0.1  127.0.0.1   TCP       44 62386 → 6666 [ACK] Seq=43 …
  8 0.017124  127.0.0.1  127.0.0.1   TCP       44 6666 → 62386 [FIN, ACK] Se…
  9 0.017140  127.0.0.1  127.0.0.1   TCP       44 62386 → 6666 [ACK] Seq=43 …
 10 0.017234  127.0.0.1  127.0.0.1   TCP       44 62386 → 6666 [FIN, ACK] Se…
 11 0.017262  127.0.0.1  127.0.0.1   TCP       44 6666 → 62386 [ACK] Seq=18 …

Frame 1: 56 bytes on wire (448 bits), 56 bytes c
0000  02 00 00 00 45 00 00 34  af 35 40 00 80 06 0
Ready to load or capture    Packets: 11 · Displayed: 11 (100.0%) · Dropped: 0 (0.0%)    Profile: Default
```

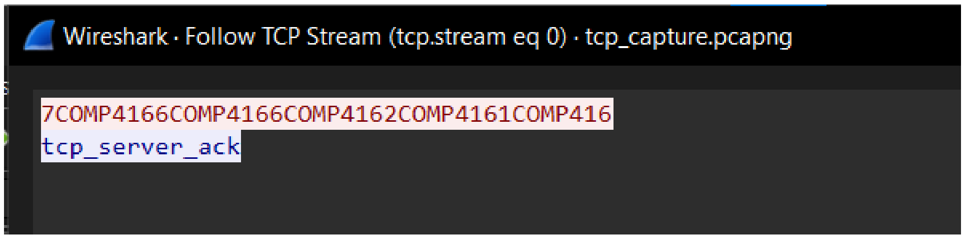
Figure 16: SSL method client and TCP method client capture logs 2.1/1.png

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

I have seen TCP and TLSv1.3 protocols in secure SSL transfer but in the unsecure there are only packets TCP protocols.

2. Can you find the sent data (the characters) within the sent packets? Show that as screenshots when possible.

In insecure method, when I right click an package in stream 0 and get the TCP Stream, I can find the data that is sent and the server acknowledgement message as follows:

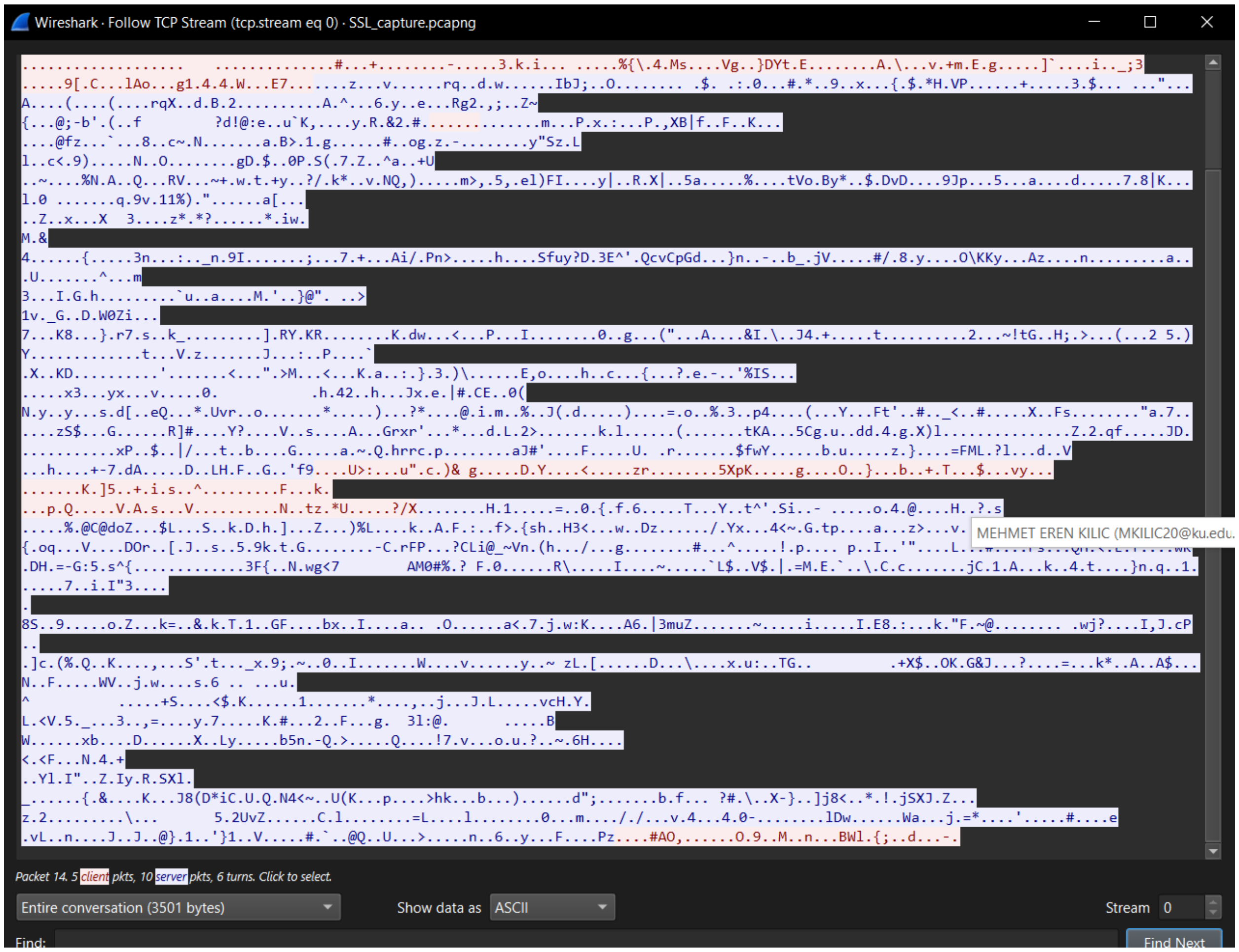


```
Wireshark · Follow TCP Stream (tcp.stream eq 0) · tcp_capture.pcapng

7COMP4166COMP4166COMP4162COMP4161COMP416
tcp_server_ack
```

Figure 17: Unsecure method stream capture message 2.1/2.png

3. By looking at the received response in the packets, can you understand what is sent? Show that as screenshots when possible.

However, in SSL communication I could not get the message. I have found some encrypted message, but I cannot see the plaintext without the appropriate decoding schema.



Wireshark · Follow TCP Stream (tcp.stream eq 0) · SSL_capture.pcapng

```
..................   ..............#...+........-.....3.k.i... .....%{\.4.Ms....Vg..}DYt.E........A.\...v.+m.E.g.....]`....i.._;3
.....9[.C...lAo...g1.4.4.W...E7.........z...v......rq..d.w......IbJ;..O........ .$. .:.0...#.*..9..x...{.$.*H.VP......+.....3.$... ..."...
A....(....(....rqX..d.B.2..........A.^...6.y..e...Rg2.,;..Z~
{...@;-b'.(..f        ?d!@:e..u`K,....y.R.&2.#.............m...P.x.:...P.,XB|f..F..K...
....@fz...`...8..c~.N.......a.B>.1.g......#..og.z.-........y"Sz.L
l..c<.9).....N..O........gD.$..0P.S(.7.Z..^a..+U
..~....%N.A..Q...RV...~+.w.t.+y..?/.k*..v.NQ,).....m>,.5,.el)FI....y|..R.X|..5a.....%....tVo.By*..$.DvD....9Jp...5...a....d.....7.8|K...
l.0 .......q.9v.11%)."......a[...
..Z..x...X  3....z*.*?......*.iw.
M.&
4......{.....3n...:.._n.9I.......;...7.+...Ai/.Pn>.....h....Sfuy?D.3E^'.QcvCpGd...}n..-..b_.jV.....#/.8.y....O\KKy...Az....n.........a..
.U.......^...m
3...I.G.h.........`u..a....M.'..}@". ..>
1v._G..D.W0Zi...
7...K8...}.r7.s..k_.........].RY.KR........K.dw...<...P...I........0..g...("...A....&I.\..J4.+.....t..........2...~!tG..H;.>...(...2 5.)
Y.............t...V.z.......J...:..P....`
.X..KD..........'.......<...".>M...<...K.a..:.}.3.)\......E,o....h..c...{...?.e.-..'%IS...
.....x3...yx...v.....0.          .h.42..h...Jx.e.|#.CE..0(
N.y..y...s.d[..eQ...*.Uvr..o.......*.....)...?*....@.i.m..%..J(.d.....)....=.o..%.3..p4....(...Y...Ft'..#.._<..#.....X..Fs........"a.7..
....zS$...G......R]#....Y?....V..s....A...Grxr'...*...d.L.2>.......k.l......(.......tKA...5Cg.u..dd.4.g.X)l...............Z.2.qf.....JD.
..........xP..$..|/...t..b....G.....a.~.Q.hrrc.p........aJ#'....F.....U. .r.......$fwY......b.u.....z.}....=FML.?l...d..V
...h....+-7.dA.....D..LH.F..G..'f9....U>:...u".c.)& g.....D.Y....<.....zr........5XpK.....g....O..}...b..+.T...$...vy...
.......K.]5..+.i.s..^.........F...k.
...p.Q.....V.A.s...V..........N..tz.*U.....?/X........H.1.....=..0.{.f.6.....T...Y..t^'.Si..- .....o.4.@....H..?.s
.....%.@C@doZ...$L...S..k.D.h.]...Z...)%L....k..A.F.:..f>.{sh..H3<...w..Dz....../.Yx...4<~.G.tp....a...z>...v.
{.oq...V....DOr..[.J..s..5.9k.t.G........-C.rFP...?CLi@_~Vn.(h.../...g........#...^.....!.p.... p..I..'"....L.
.DH.=-G:5.s^{.............3F{..N.wg<7       AM0#%.? F.0......R\.....I....~.....`L$..V$.|.=M.E.`..\.C.c.......jC.1.A...k..4.t....}n.q..1.
.....7..i.I"3....
.
8S..9.....o.Z...k=..&.k.T.1..GF....bx..I....a.. .O......a<.7.j.w:K....A6.|3muZ.......~.....i.....I.E8.:...k."F.~@........ .wj?....I,J.cP
..
.]c.(%.Q..K....,...S'.t..._x.9;.~..0..I.......W....v......y..~ zL.[......D...\....x.u:..TG..        .+X$..OK.G&J...?....=...k*..A..A$...
N..F.....WV..j.w....s.6 .. ...u.
^          .....+S....<$.K......1.......*....,..j...J.L.....vcH.Y.
L.<V.5._...3..,=....y.7.....K.#...2..F...g.  3l:@.       .....B
W......xb....D......X..Ly.....b5n.-Q.>.....Q....!7.v...o.u.?..~.6H....
<.<F...N.4.+
..Yl.I"..Z.Iy.R.SXl.
_......{.&....K...J8(D*iC.U.Q.N4<~..U(K...p....>hk...b...)......d";.......b.f... ?#.\..X-}..]j8<..*.!.jSXJ.Z...
z.2.........\...       5.2UvZ......C.l........=L....l........0...m..../../...v.4...4.0-.........lDw......Wa...j.=*....'.....#....e
.vL..n....J..J..@}.1..'}1..V.....#.`..@Q..U...>.....n..6..y...F....Pz....#AO,......O.9..M..n...BWl.{;..d...-.
```

MEHMET EREN KILIC (MKILIC20@ku.edu

Packet 14. 5 client pkts, 10 server pkts, 6 turns. Click to select.

Entire conversation (3501 bytes) | Show data as ASCII | Stream 0

Find: | Find Next

Figure 18: SSL method message 2.1/3.png

As I stated in question 2, I could see the message in insecure communication but not in the secure communication.

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).

I put the previous client commands in a for loop with 5 loops. In each iteration two lists saves TCP and SSL execution delays. Then, I used JFreeChart to visualize the chart and see the delay difference between these two-execution type. What I have seen, is SSL communication is way longer than normal TCP communication. I believe this is due to the encryption and decryption of the messages that are being sent and the handshake process at the beginning.

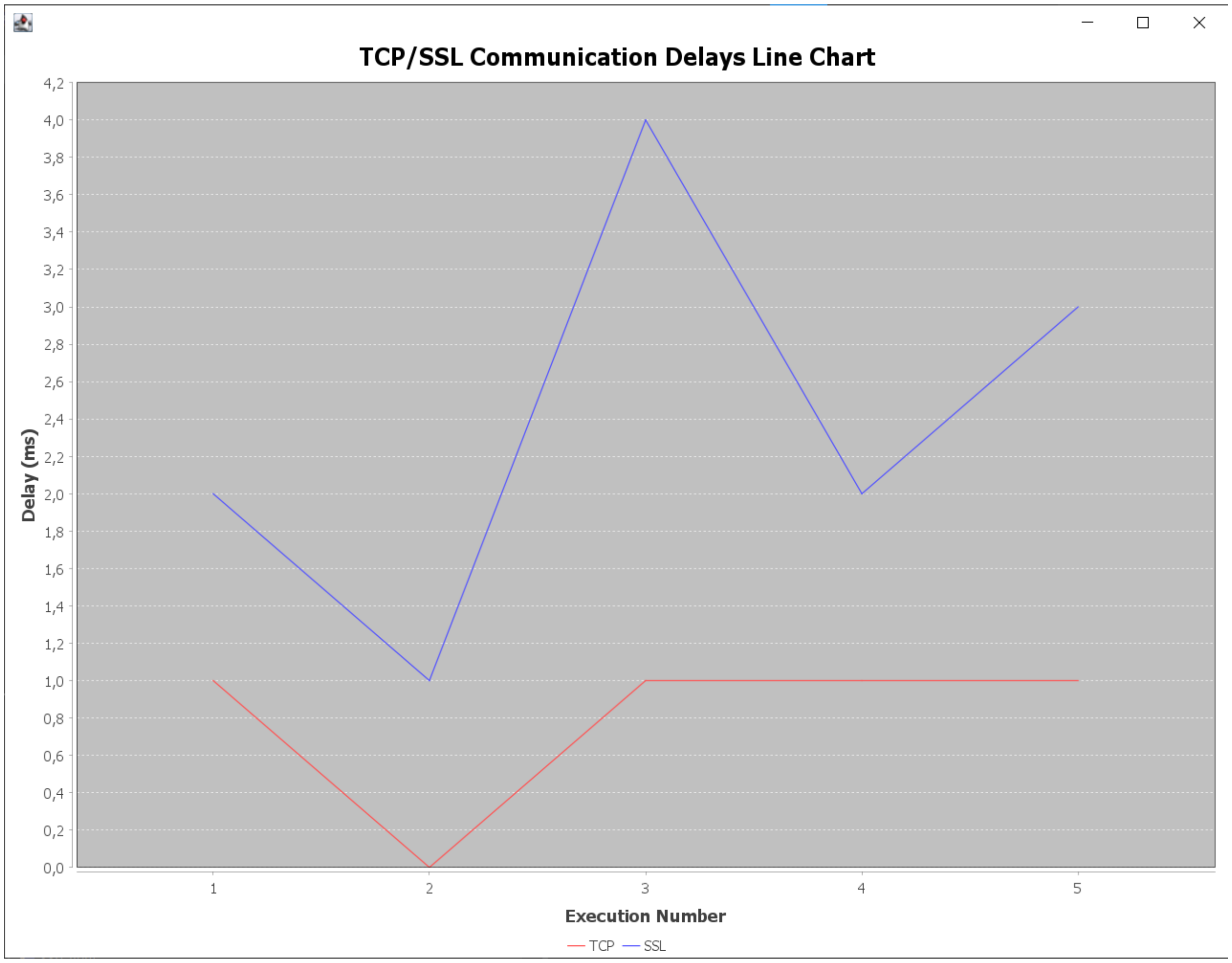
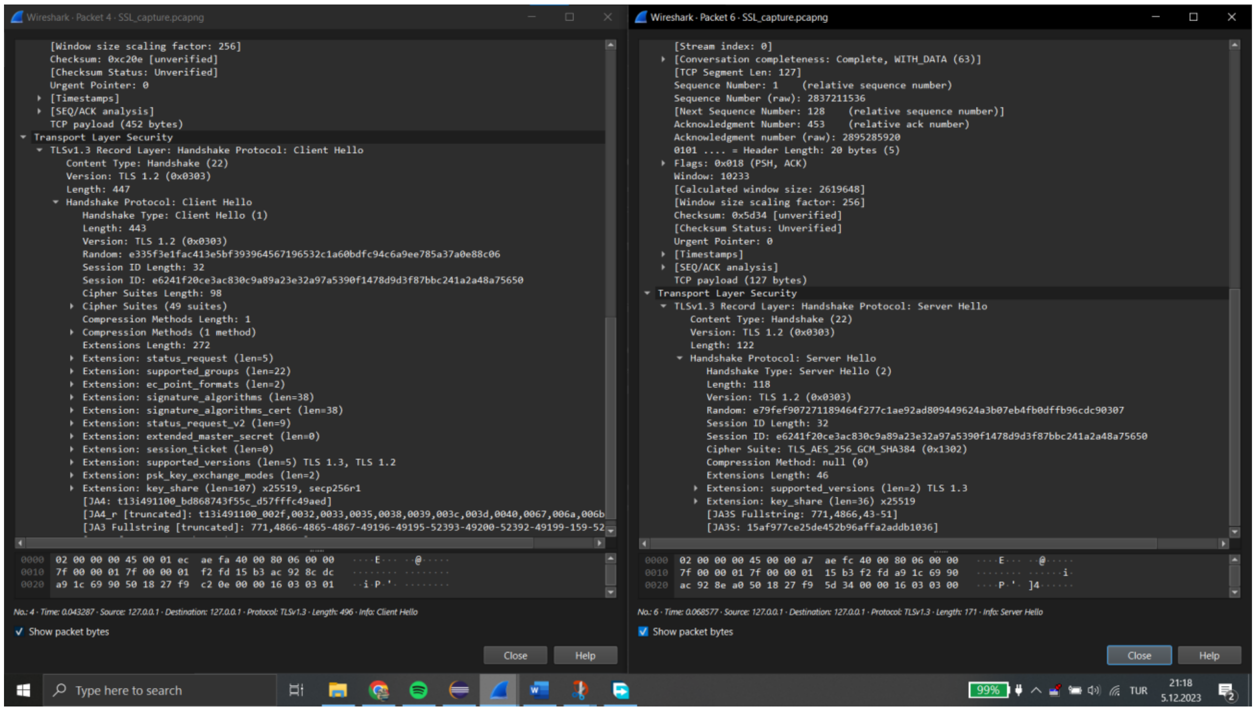


Figure 19: TCP vs SSL RTT comparison chart 2.1/4.png

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.



Wireshark · Packet 4 · SSL_capture.pcapng

```
      [Window size scaling factor: 256]
      Checksum: 0xc20e [unverified]
      [Checksum Status: Unverified]
      Urgent Pointer: 0
    ▸ [Timestamps]
    ▸ [SEQ/ACK analysis]
      TCP payload (452 bytes)
  ▾ 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: e335f3e1fac413e5bf393964567196532c1a60bdfc94c6a9ee785a37a0e88c06
          Session ID Length: 32
          Session ID: e6241f20ce3ac830c9a89a23e32a97a5390f1478d9d3f87bbc241a2a48a75650
          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: signature_algorithms (len=38)
        ▸ Extension: signature_algorithms_cert (len=38)
        ▸ Extension: status_request_v2 (len=9)
        ▸ Extension: extended_master_secret (len=0)
        ▸ Extension: session_ticket (len=0)
        ▸ Extension: supported_versions (len=5) TLS 1.3, TLS 1.2
        ▸ Extension: psk_key_exchange_modes (len=2)
        ▸ Extension: key_share (len=107) x25519, secp256r1
          [JA4: t13i491100_bd868743f55c_d57fffc49aed]
          [JA4_r [truncated]: t13i491100_002f,0032,0033,0035,0038,0039,003c,003d,0040,0067,006a,006b
          [JA3 Fullstring [truncated]: 771,4866-4865-4867-49196-49195-52393-49200-52392-49199-159-52

0000  02 00 00 00 45 00 01 ec  ae fa 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  f2 fd 15 b3 ac 92 8c dc   ········ ········
0020  a9 1c 69 90 50 18 27 f9  c2 0e 00 00 16 03 03 01   ··i·P·'· ········
```

No.: 4 · Time: 0.043287 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: TLSv1.3 · Length: 496 · Info: Client Hello

Wireshark · Packet 6 · SSL_capture.pcapng

```
      [Stream index: 0]
    ▸ [Conversation completeness: Complete, WITH_DATA (63)]
      [TCP Segment Len: 127]
      Sequence Number: 1    (relative sequence number)
      Sequence Number (raw): 2837211536
      [Next Sequence Number: 128    (relative sequence number)]
      Acknowledgment Number: 453    (relative ack number)
      Acknowledgment number (raw): 2895285920
      0101 .... = Header Length: 20 bytes (5)
    ▸ Flags: 0x018 (PSH, ACK)
      Window: 10233
      [Calculated window size: 2619648]
      [Window size scaling factor: 256]
      Checksum: 0x5d34 [unverified]
      [Checksum Status: Unverified]
      Urgent Pointer: 0
    ▸ [Timestamps]
    ▸ [SEQ/ACK analysis]
      TCP payload (127 bytes)
  ▾ 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: e79fef907271189464f277c1ae92ad809449624a3b07eb4fb0dffb96cdc90307
          Session ID Length: 32
          Session ID: e6241f20ce3ac830c9a89a23e32a97a5390f1478d9d3f87bbc241a2a48a75650
          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  ae fc 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  15 b3 f2 fd a9 1c 69 90   ········ ······i·
0020  ac 92 8e a0 50 18 27 f9  5d 34 00 00 16 03 03 00   ····P·'· ]4······
```

No.: 6 · Time: 0.068577 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: TLSv1.3 · Length: 171 · Info: Server Hello

Figure 20: SSL Client Hello and Server Hello packages information 2.1/5.png

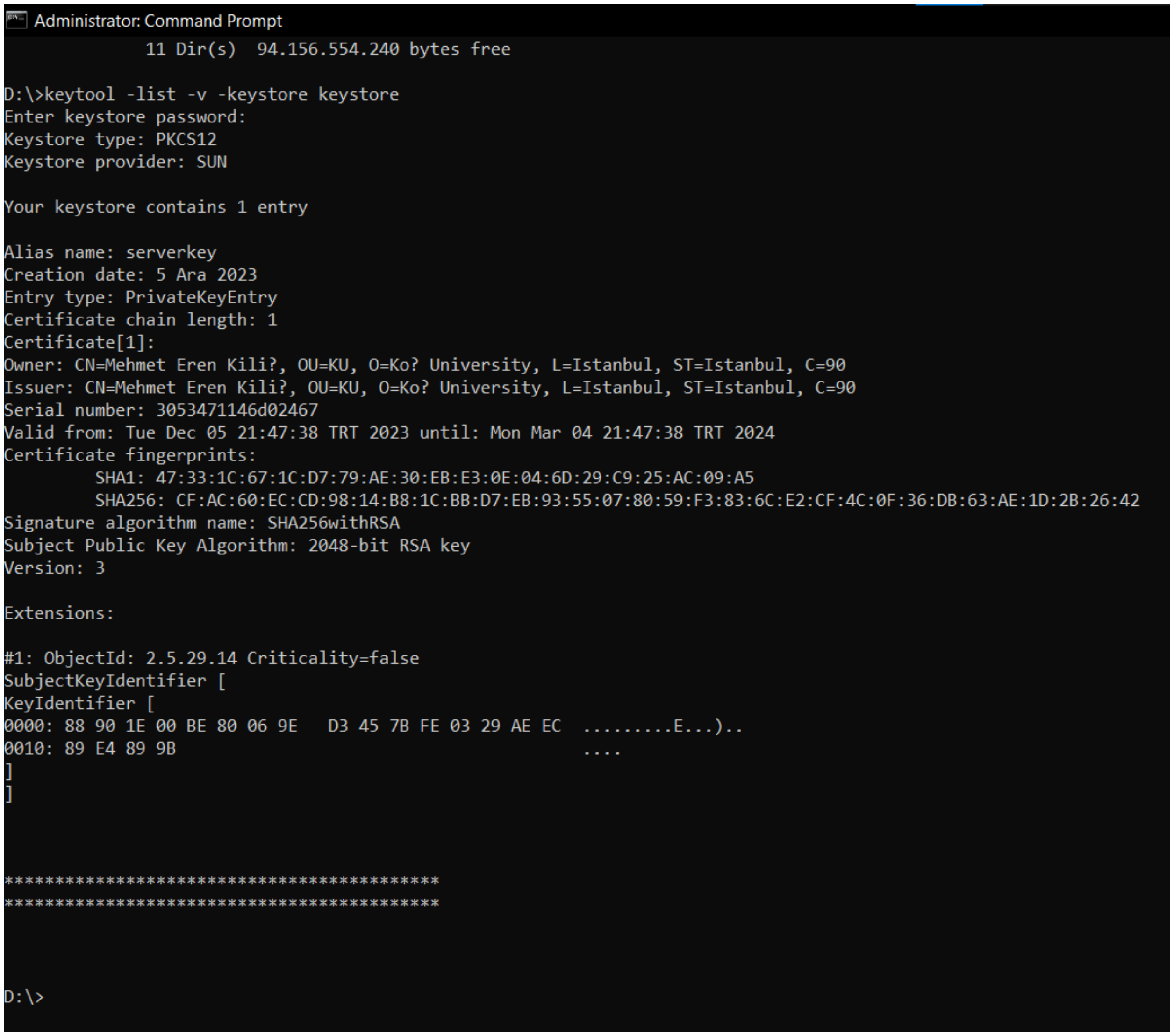
They are both part of the handshake process of the SSL communication. In order to establish the communication these both packets are being sent and received, thus the communication can successfully hold. The packages contain version information, keys for encryption and decryption schemas and certificate verification etc.

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

Both Client and Server Hello continues from the same Session ID. In order to avoid sending the certificate again and again and cause delay and performance reduction, if they share the same Session ID, they continue to be communicating without duplication of certificate verification. Thus, in this capture, there is only one handshake between the server and the client.

Part 2.2)

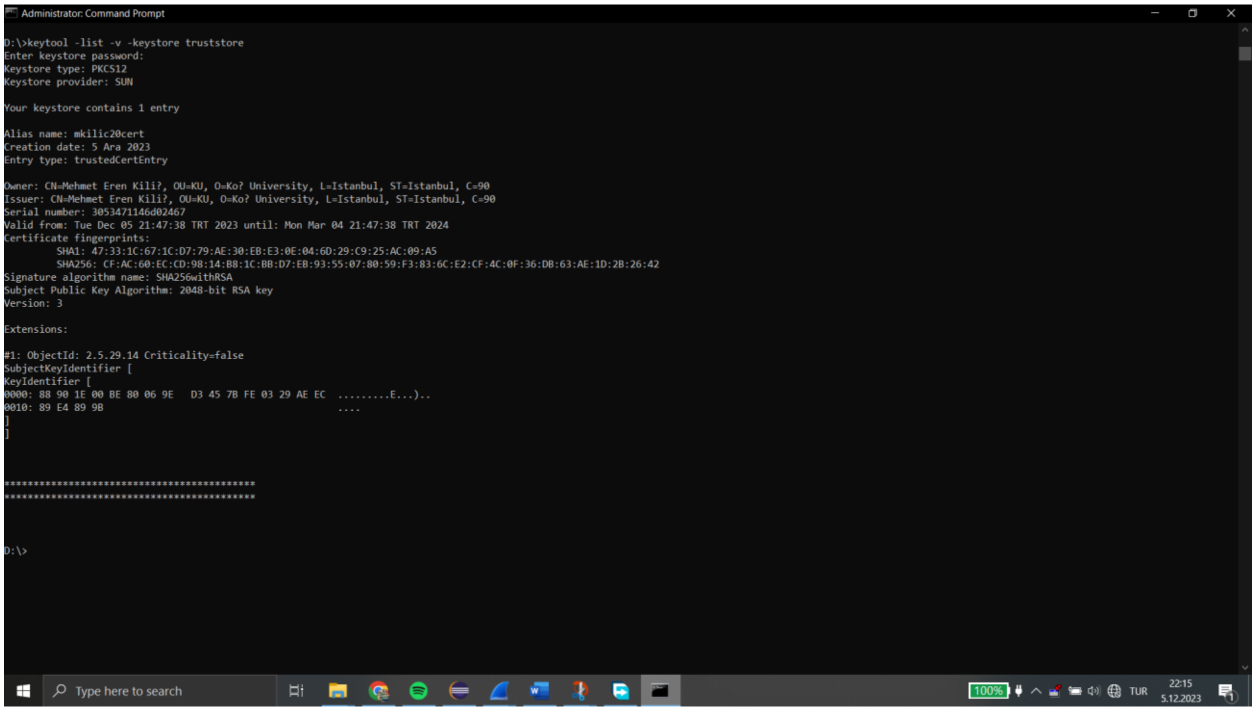
I have created the keystore with the help of the tutorial provided. My keystore information is attached below:



```
Administrator: Command Prompt
              11 Dir(s)  94.156.554.240 bytes free

D:\>keytool -list -v -keystore keystore
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: serverkey
Creation date: 5 Ara 2023
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Mehmet Eren Kili?, OU=KU, O=Ko? University, L=Istanbul, ST=Istanbul, C=90
Issuer: CN=Mehmet Eren Kili?, OU=KU, O=Ko? University, L=Istanbul, ST=Istanbul, C=90
Serial number: 3053471146d02467
Valid from: Tue Dec 05 21:47:38 TRT 2023 until: Mon Mar 04 21:47:38 TRT 2024
Certificate fingerprints:
         SHA1: 47:33:1C:67:1C:D7:79:AE:30:EB:E3:0E:04:6D:29:C9:25:AC:09:A5
         SHA256: CF:AC:60:EC:CD:98:14:B8:1C:BB:D7:EB:93:55:07:80:59:F3:83:6C:E2:CF:4C:0F:36:DB:63:AE:1D:2B:26:42
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 88 90 1E 00 BE 80 06 9E   D3 45 7B FE 03 29 AE EC  .........E...)..
0010: 89 E4 89 9B                                        ....
]
]



*******************************************
*******************************************



D:\>
```

Figure 21: keystore information 2.2/1.png

Then I have created my truststore, the information of the truststore is attached below:



```
D:\>keytool -list -v -keystore truststore
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: mkilic20cert
Creation date: 5 Ara 2023
Entry type: trustedCertEntry

Owner: CN=Mehmet Eren Kili?, OU=KU, O=Ko? University, L=Istanbul, ST=Istanbul, C=90
Issuer: CN=Mehmet Eren Kili?, OU=KU, O=Ko? University, L=Istanbul, ST=Istanbul, C=90
Serial number: 3053471146d02467
Valid from: Tue Dec 05 21:47:38 TRT 2023 until: Mon Mar 04 21:47:38 TRT 2024
Certificate fingerprints:
         SHA1: 47:33:1C:67:1C:D7:79:AE:30:EB:E3:0E:04:6D:29:C9:25:AC:09:A5
         SHA256: CF:AC:60:EC:CD:98:14:B8:1C:BB:D7:EB:93:55:07:80:59:F3:83:6C:E2:CF:4C:0F:36:DB:63:AE:1D:2B:26:42
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 88 90 1E 00 BE 80 06 9E   D3 45 7B FE 03 29 AE EC  .........E...)..
0010: 89 E4 89 9B                                        ....
]
]



*******************************************
*******************************************



D:\>
```

Figure 22: Truststore information 2.2/2.png