Serkan Berk Bilgic
71571

Question 1.1:

1- My student number is **71571**. So I marked the packet with segment number **71**. As can be seen from the 3rd screenshot my segments TTL value is: **64**. TTL short for Time to Live is a maximum amount of time/hops the packet is allowed to live or traverse through network. It is a mechanism to control the lifespan and reach of network packets and if it reaches 0 before packet is arriving to a destination the packet is dropped and error message is sent. TTL helps prevent packets from circulating indefinitely in a network due to routing errors or other issues.

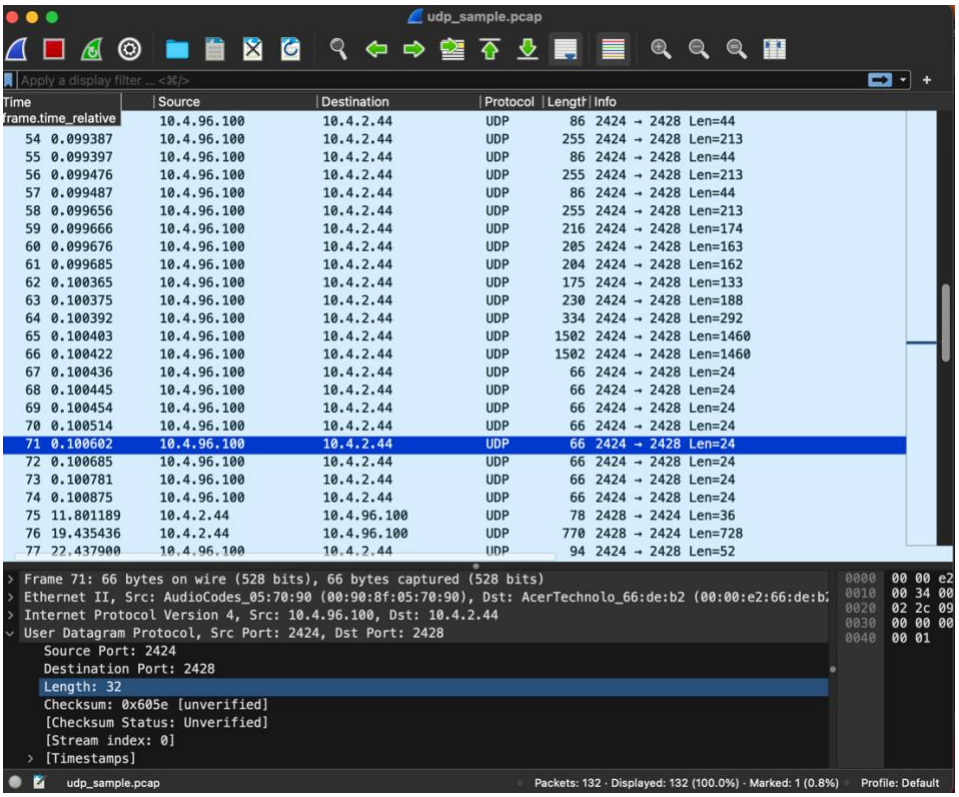


Fig1

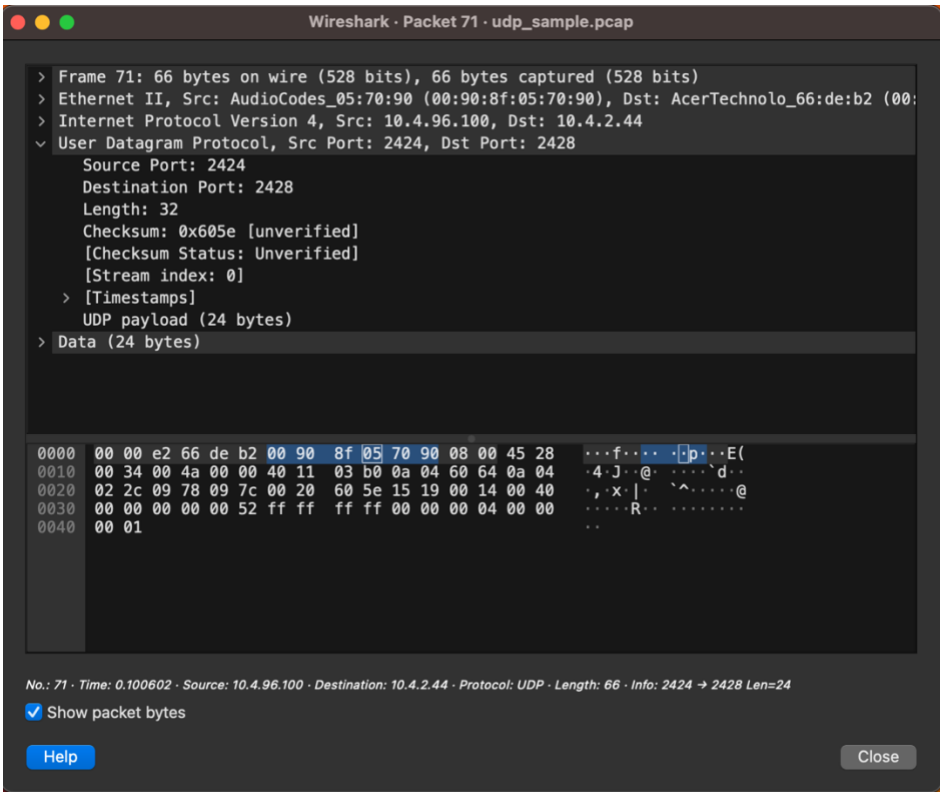

Wireshark · Packet 71 · udp_sample.pcap
Frame 71: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (00:
Internet Protocol Version 4, Src: 10.4.96.100, Dst: 10.4.2.44
User Datagram Protocol, Src Port: 2424, Dst Port: 2428
Source Port: 2424
Destination Port: 2428
Length: 32
Checksum: 0x605e [unverified]
[Checksum Status: Unverified]
[Stream index: 0]
[Timestamps]
UDP payload (24 bytes)
Data (24 bytes)
0000 00 00 e2 66 de b2 00 90 8f 05 70 90 08 00 45 28
0010 00 34 00 4a 00 00 40 11 03 b0 0a 04 60 64 0a 04
0020 02 2c 09 78 09 7c 00 20 60 5e 15 19 00 14 00 40
0030 00 00 00 00 00 52 ff ff ff ff 00 00 00 04 00 00
0040 00 01
No.: 71 · Time: 0.100602 · Source: 10.4.96.100 · Destination: 10.4.2.44 · Protocol: UDP · Length: 66 · Info: 2424 → 2428 Len=24
Show packet bytes
Help
Close

Fig2

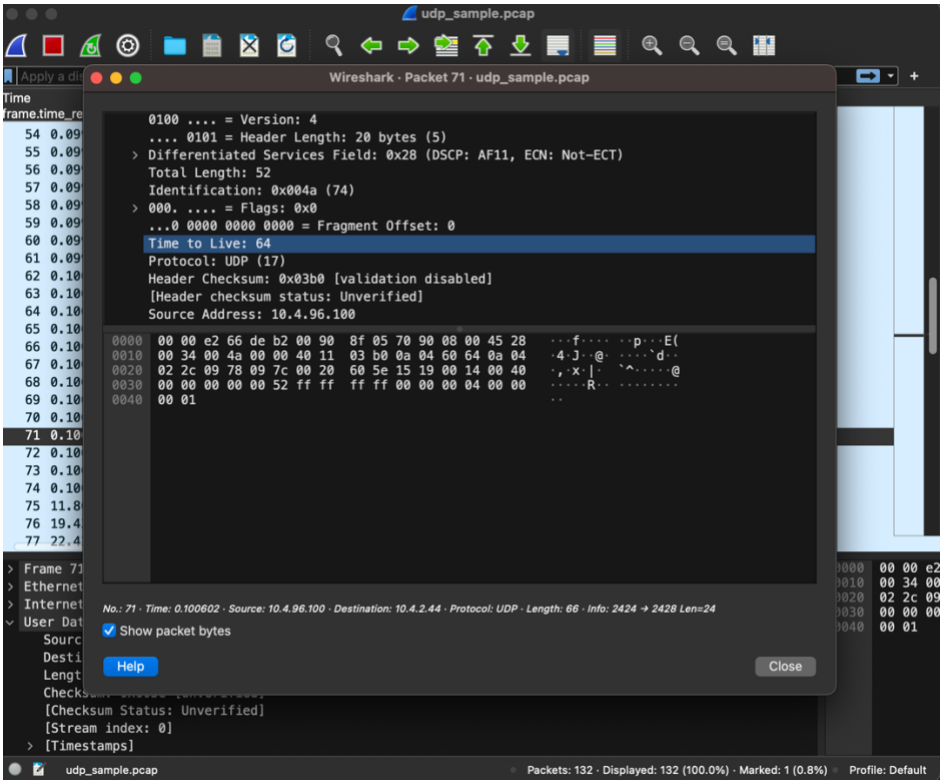

udp_sample.pcap
Wireshark · Packet 71 · udp_sample.pcap
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x28 (DSCP: AF11, ECN: Not-ECT)
Total Length: 52
Identification: 0x004a (74)
000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0x03b0 [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.4.96.100
0000 00 00 e2 66 de b2 00 90 8f 05 70 90 08 00 45 28
0010 00 34 00 4a 00 00 40 11 03 b0 0a 04 60 64 0a 04
0020 02 2c 09 78 09 7c 00 20 60 5e 15 19 00 14 00 40
0030 00 00 00 00 00 52 ff ff ff ff 00 00 00 04 00 00
0040 00 01
No.: 71 · Time: 0.100602 · Source: 10.4.96.100 · Destination: 10.4.2.44 · Protocol: UDP · Length: 66 · Info: 2424 → 2428 Len=24
Show packet bytes
Help
Close
[Checksum Status: Unverified]
[Stream index: 0]
[Timestamps]
udp_sample.pcap
Packets: 132 · Displayed: 132 (100.0%) · Marked: 1 (0.8%)
Profile: Default

Fig3

2- As can be seen from the below screenshots my segment's stream index is 0. The stream index for a UDP segment represents the numbering of conversations associated with TCP and UDP socket pairs. The calculations for TCP and UDP are done independently. The first conversation is assigned the stream index 0, the second conversation gets index 1, and so on.

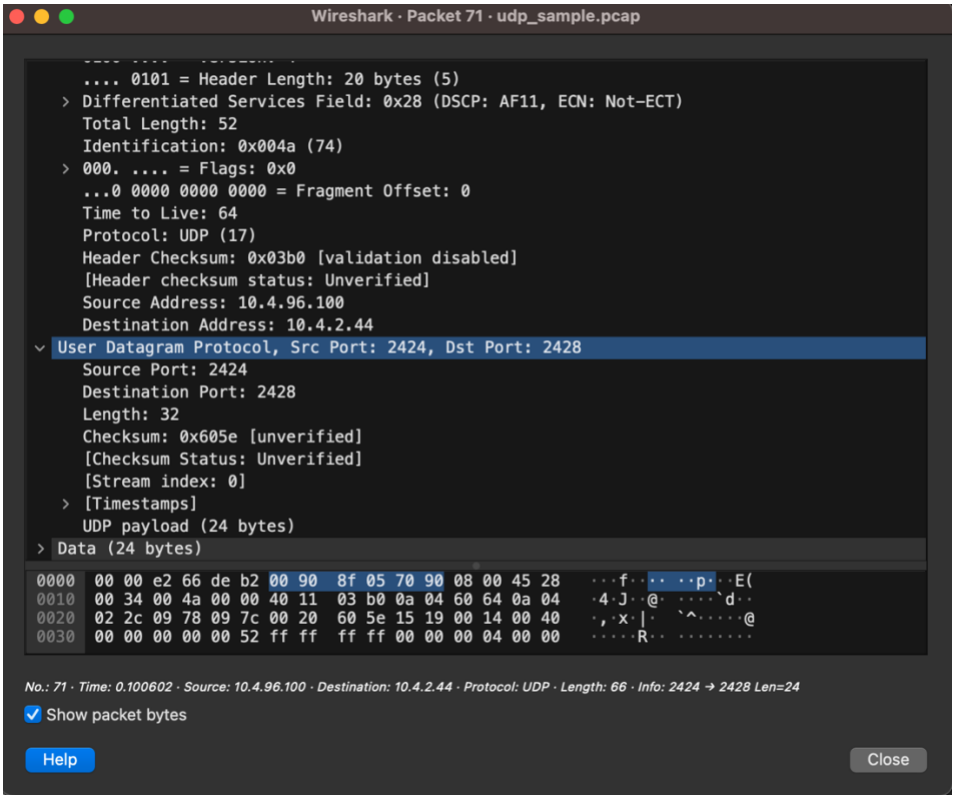


Fig1

3- Checksum value of the segment can be seen in the below screenshot. It is **0x605e(24670)** in the UDP header**.** Checksum is a mechanism to check the corruption of the packet, detect errors throughout its transmission. It is calculated by treating contents of UDP/TCP segment as sequence of 16 bit integers and adding all of them. It is present both on UDP and TCP. Main difference between them is while in TCP if checksum resulted different in server and client side, protocol retransmits the packet, in UDP retransmission is not done. Based on the application client may decide to use the packet which has error or discard it.

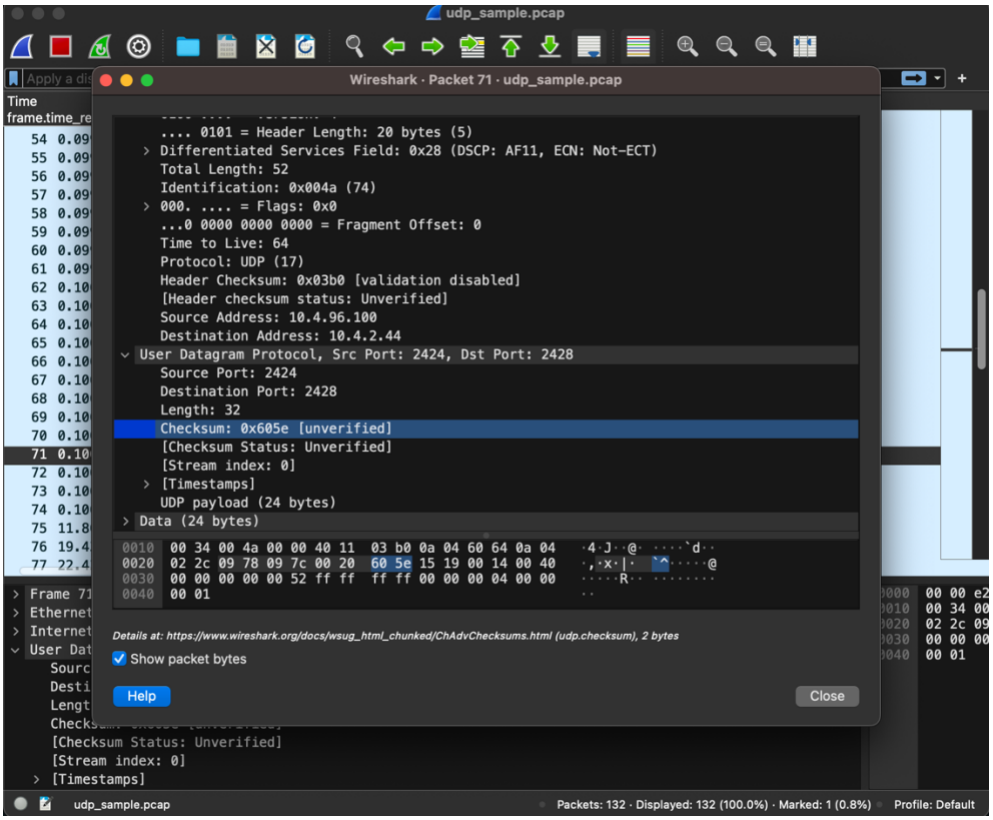


Fig1

4- As can be seen from the below screenshot reserved bit is 0. The IP v4 header has three flags: the reserved bit, the don't fragment bit, and the more fragments bit.
The least important bit in the Flags field is the reserved bit, or bit 0.
If set, the Don't Fragment (DF) bit (bit 1) denotes that the packet shouldn't be broken up while being transmitted. The More Fragments (MF) bit (bit 2), if set, indicates that there are more fragments to come and that the packet is a part of a fragmented collection of packets. In the current IPv4 specification, the Reserved bit is set aside for possible future usage and has no defined use. Devices processing IPv4 packets should always set it to 0 and ignore it.

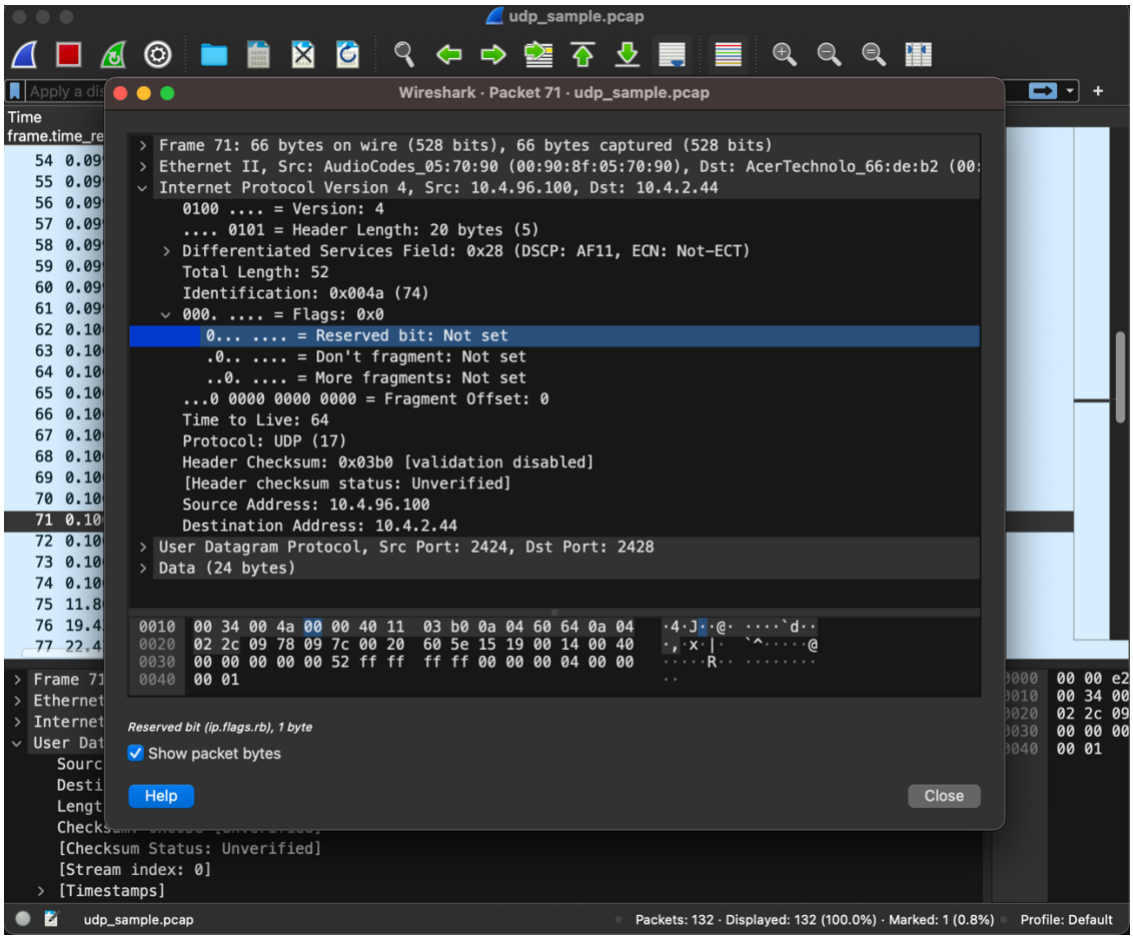


Fig1

5- The total length of the frame is 66 bytes incluuding header and data. The total length of UDP is 32 and UDP payload is 24 . You can see both of them in below.

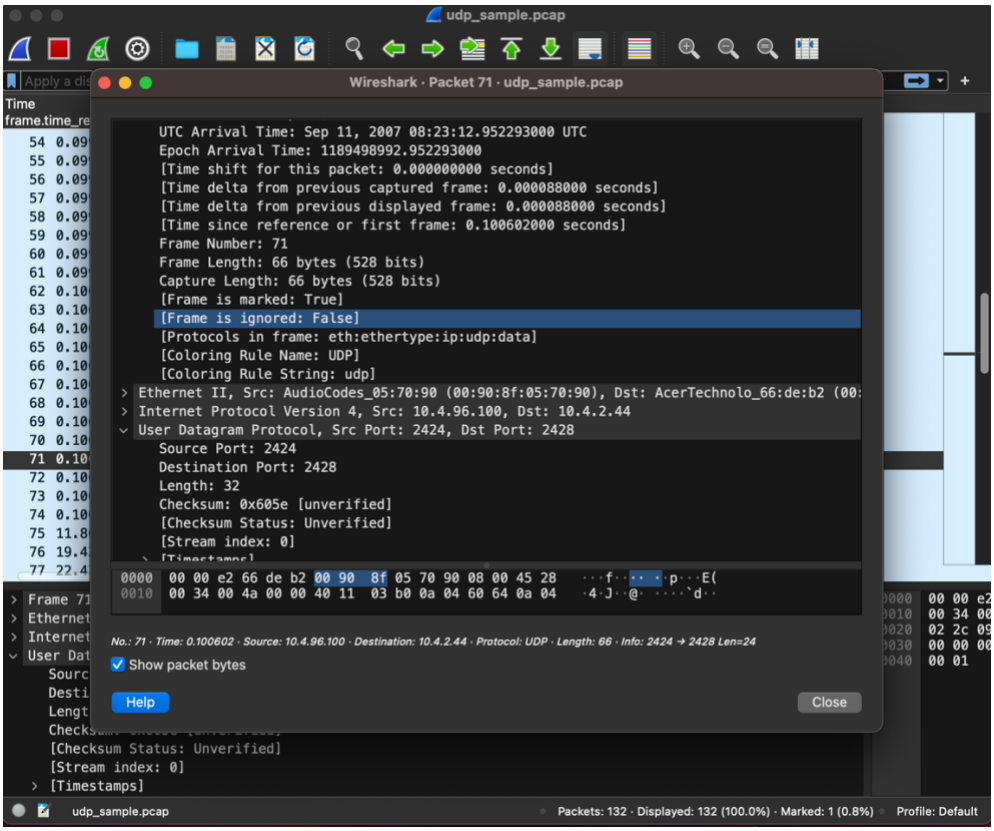

udp_sample.pcap
Wireshark · Packet 71 · udp_sample.pcap
UTC Arrival Time: Sep 11, 2007 08:23:12.952293000 UTC
Epoch Arrival Time: 1189498992.952293000
[Time shift for this packet: 0.000000000 seconds]
[Time delta from previous captured frame: 0.000088000 seconds]
[Time delta from previous displayed frame: 0.000088000 seconds]
[Time since reference or first frame: 0.100602000 seconds]
Frame Number: 71
Frame Length: 66 bytes (528 bits)
Capture Length: 66 bytes (528 bits)
[Frame is marked: True]
[Frame is ignored: False]
[Protocols in frame: eth:ethertype:ip:udp:data]
[Coloring Rule Name: UDP]
[Coloring Rule String: udp]
Ethernet II, Src: AudioCodes_05:70:90 (00:90:8f:05:70:90), Dst: AcerTechnolo_66:de:b2 (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: 32
Checksum: 0x605e [unverified]
[Checksum Status: Unverified]
[Stream index: 0]
0000 00 00 e2 66 de b2 00 90 8f 05 70 90 08 00 45 28
0010 00 34 00 4a 00 00 40 11 03 b0 0a 04 60 64 0a 04
No.: 71 · Time: 0.100602 · Source: 10.4.96.100 · Destination: 10.4.2.44 · Protocol: UDP · Length: 66 · Info: 2424 → 2428 Len=24
Show packet bytes
Help
Close
Packets: 132 · Displayed: 132 (100.0%) · Marked: 1 (0.8%)
Profile: Default

Fig1

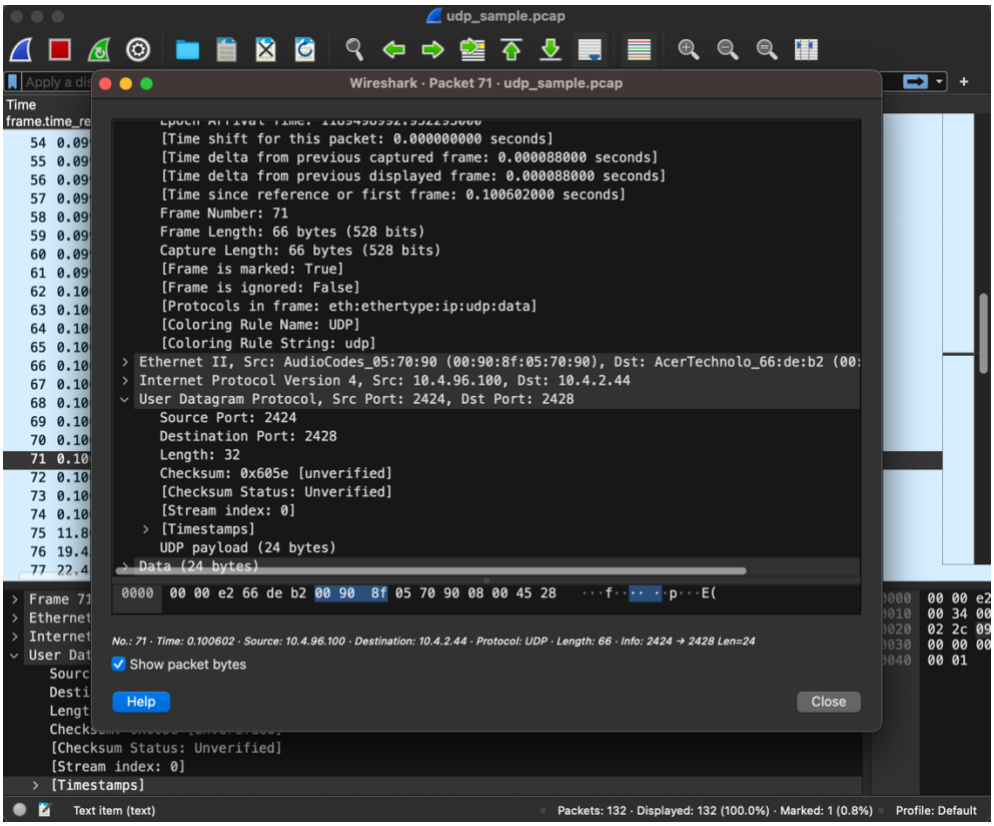


Fig2

Question 1.2:

1- From the first screenshots it can be seen that there are 2 TCP retransmissions. To verify that we can check the source and destination in TCP retranssmission and http post request which can be seen in the 2nd screenshot, they are the same. Srcport: 52790 and dst port is 80, source ip address is 172.21.157.81 destination IP address is 128.119.245.12 in both screenshots. 3rd screenshot shows TCP duplicate ACKs which is an indication of that a segment **might have been lost**. To check that I looked tcp.analysis.lost_segment and there is nothing so we can say that there is no loss.

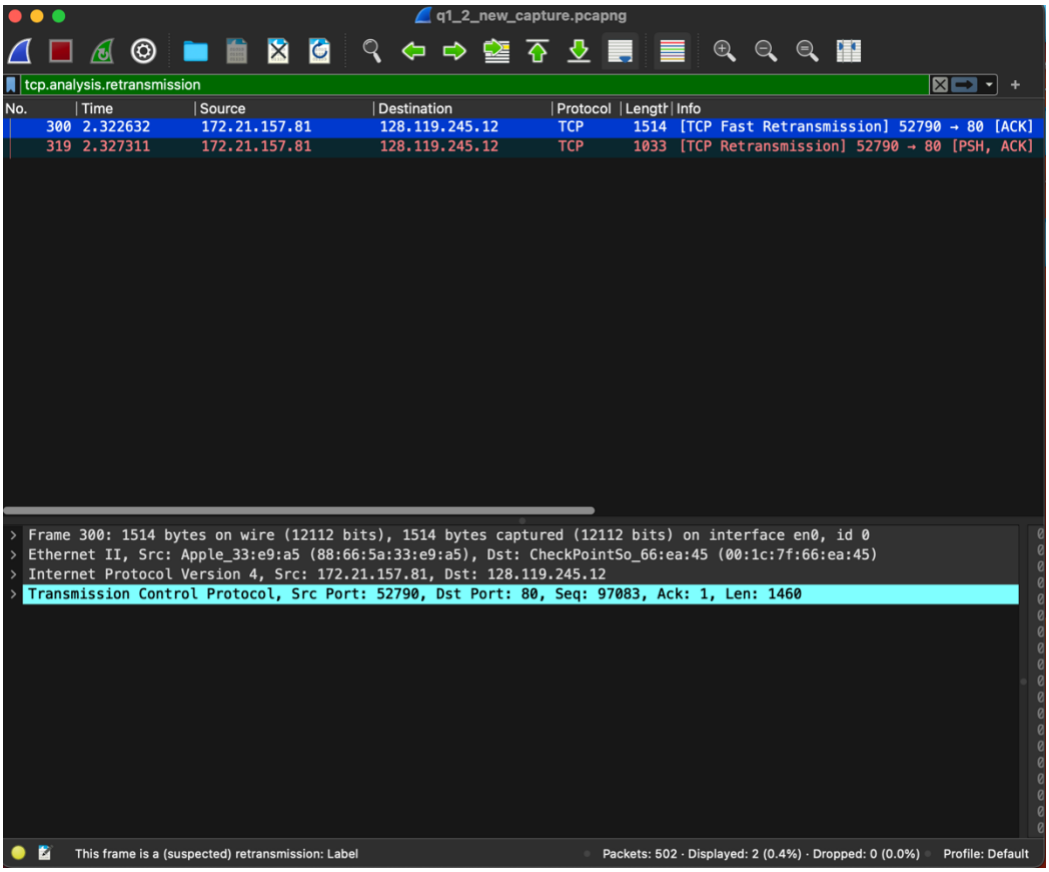

q1_2_new_capture.pcapng
tcp.analysis.retransmission
No. Time Source Destination Protocol Length Info
300 2.322632 172.21.157.81 128.119.245.12 TCP 1514 [TCP Fast Retransmission] 52790 → 80 [ACK]
319 2.327311 172.21.157.81 128.119.245.12 TCP 1033 [TCP Retransmission] 52790 → 80 [PSH, ACK]
Frame 300: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface en0, id 0
Ethernet II, Src: Apple_33:e9:a5 (88:66:5a:33:e9:a5), Dst: CheckPointSo_66:ea:45 (00:1c:7f:66:ea:45)
Internet Protocol Version 4, Src: 172.21.157.81, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 52790, Dst Port: 80, Seq: 97083, Ack: 1, Len: 1460
This frame is a (suspected) retransmission: Label
Packets: 502 · Displayed: 2 (0.4%) · Dropped: 0 (0.0%)
Profile: Default

Fig1

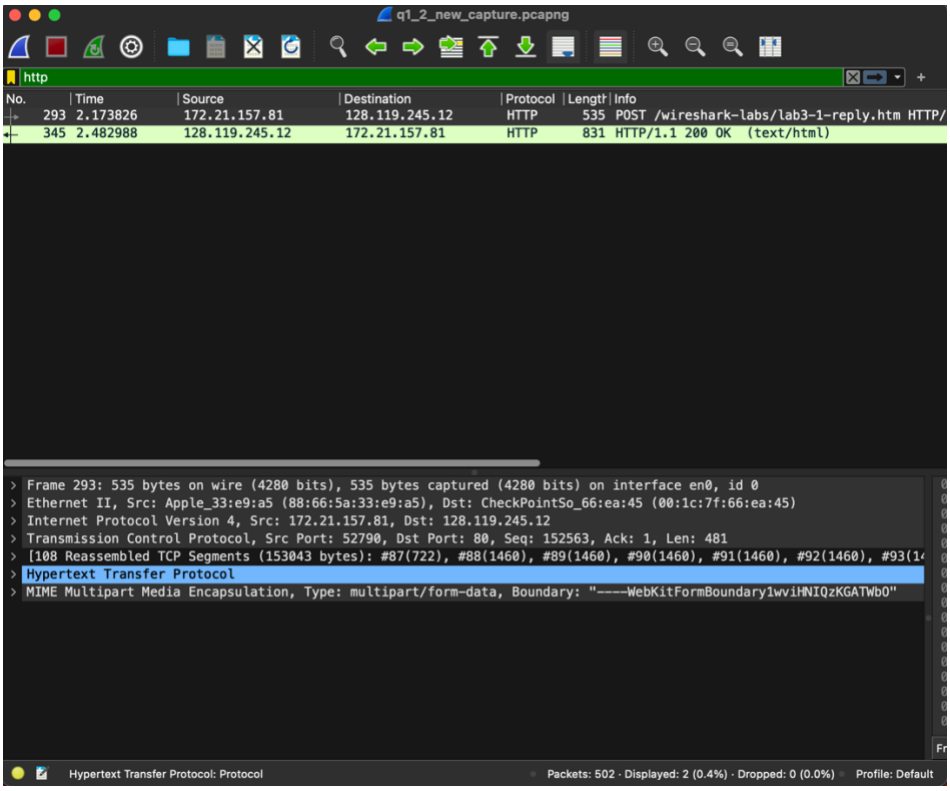

q1_2_new_capture.pcapng
http
No. Time Source Destination Protocol Length Info
293 2.173826 172.21.157.81 128.119.245.12 HTTP 535 POST /wireshark-labs/lab3-1-reply.htm HTTP/
345 2.482988 128.119.245.12 172.21.157.81 HTTP 831 HTTP/1.1 200 OK (text/html)
Frame 293: 535 bytes on wire (4280 bits), 535 bytes captured (4280 bits) on interface en0, id 0
Ethernet II, Src: Apple_33:e9:a5 (88:66:5a:33:e9:a5), Dst: CheckPointSo_66:ea:45 (00:1c:7f:66:ea:45)
Internet Protocol Version 4, Src: 172.21.157.81, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 52790, Dst Port: 80, Seq: 152563, Ack: 1, Len: 481
[108 Reassembled TCP Segments (153043 bytes): #87(722), #88(1460), #89(1460), #90(1460), #91(1460), #92(1460), #93(1
Hypertext Transfer Protocol
MIME Multipart Media Encapsulation, Type: multipart/form-data, Boundary: "----WebKitFormBoundary1wviHNIQzKGATWbO"
Hypertext Transfer Protocol: Protocol
Packets: 502 · Displayed: 2 (0.4%) · Dropped: 0 (0.0%)
Profile: Default

Fig2

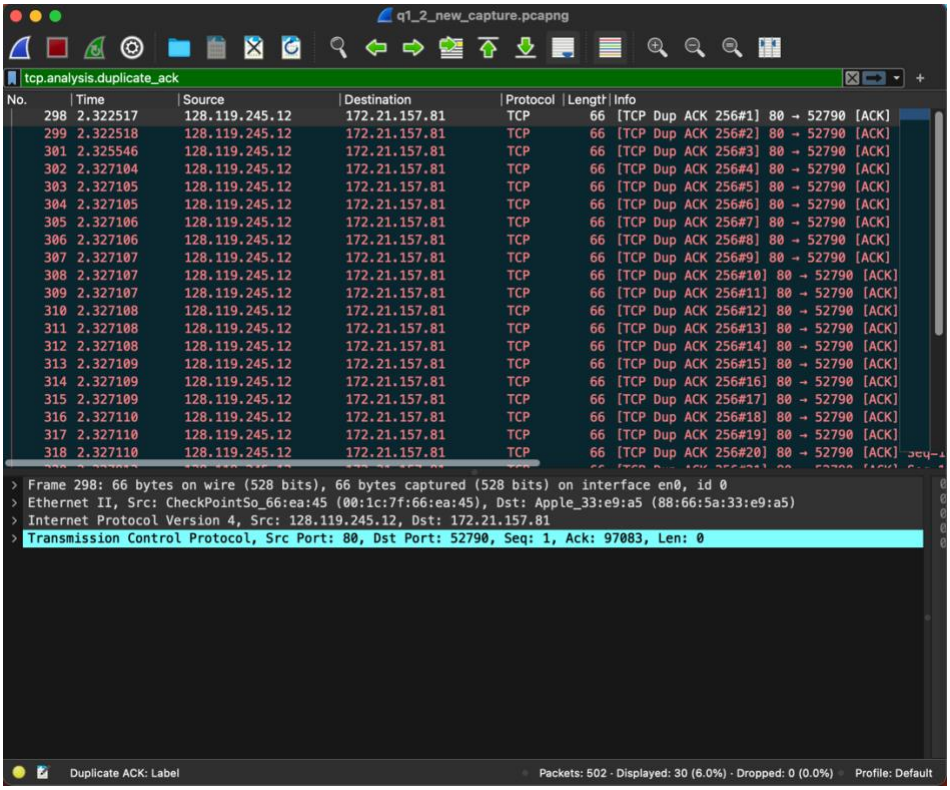

q1_2_new_capture.pcapng
tcp.analysis.duplicate_ack
No. Time Source Destination Protocol Length Info
298 2.322517 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#1] 80 → 52790 [ACK]
299 2.322518 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#2] 80 → 52790 [ACK]
301 2.325546 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#3] 80 → 52790 [ACK]
302 2.327104 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#4] 80 → 52790 [ACK]
303 2.327105 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#5] 80 → 52790 [ACK]
304 2.327105 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#6] 80 → 52790 [ACK]
305 2.327106 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#7] 80 → 52790 [ACK]
306 2.327106 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#8] 80 → 52790 [ACK]
307 2.327107 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#9] 80 → 52790 [ACK]
308 2.327107 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#10] 80 → 52790 [ACK]
309 2.327107 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#11] 80 → 52790 [ACK]
310 2.327108 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#12] 80 → 52790 [ACK]
311 2.327108 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#13] 80 → 52790 [ACK]
312 2.327108 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#14] 80 → 52790 [ACK]
313 2.327109 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#15] 80 → 52790 [ACK]
314 2.327109 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#16] 80 → 52790 [ACK]
315 2.327109 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#17] 80 → 52790 [ACK]
316 2.327110 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#18] 80 → 52790 [ACK]
317 2.327110 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#19] 80 → 52790 [ACK]
318 2.327110 128.119.245.12 172.21.157.81 TCP 66 [TCP Dup ACK 256#20] 80 → 52790 [ACK]
Frame 298: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface en0, id 0
Ethernet II, Src: CheckPointSo_66:ea:45 (00:1c:7f:66:ea:45), Dst: Apple_33:e9:a5 (88:66:5a:33:e9:a5)
Internet Protocol Version 4, Src: 128.119.245.12, Dst: 172.21.157.81
Transmission Control Protocol, Src Port: 80, Dst Port: 52790, Seq: 1, Ack: 97083, Len: 0
Duplicate ACK: Label
Packets: 502 · Displayed: 30 (6.0%) · Dropped: 0 (0.0%)
Profile: Default

Fig3

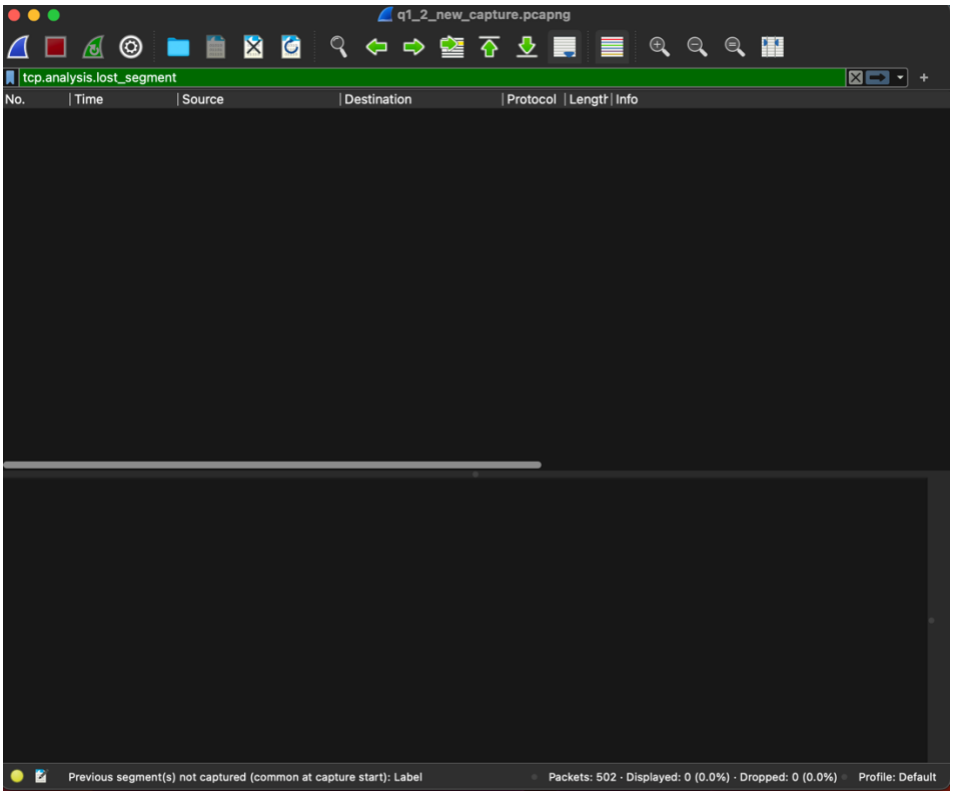


Fig4

2- RTT(Round-Trip Time) is time it takes for a packet to travel from its source to its destination and back for a particular TCP (Transmission Control Protocol) connection. It is a measurement of the delay or latency that data experiences during its round-trip through the network.shows how much data the recipient is willing to accept. As can be seen from the below screenshots first I filtered conversation between my port 52790 and the server 80(Using Statistics > Conversations). Then using the Statistics tab I visualized RTT graph(Using Statistics > TCP Stream Graphs > Round Trip Time). As expected RTT varies over time. The highest I found - which is the peak in the RTT graph as well – is 2.158s. There can be a lot of different reasons for that such as network congestion, routing changes, interference&noise in wireless connection etc.

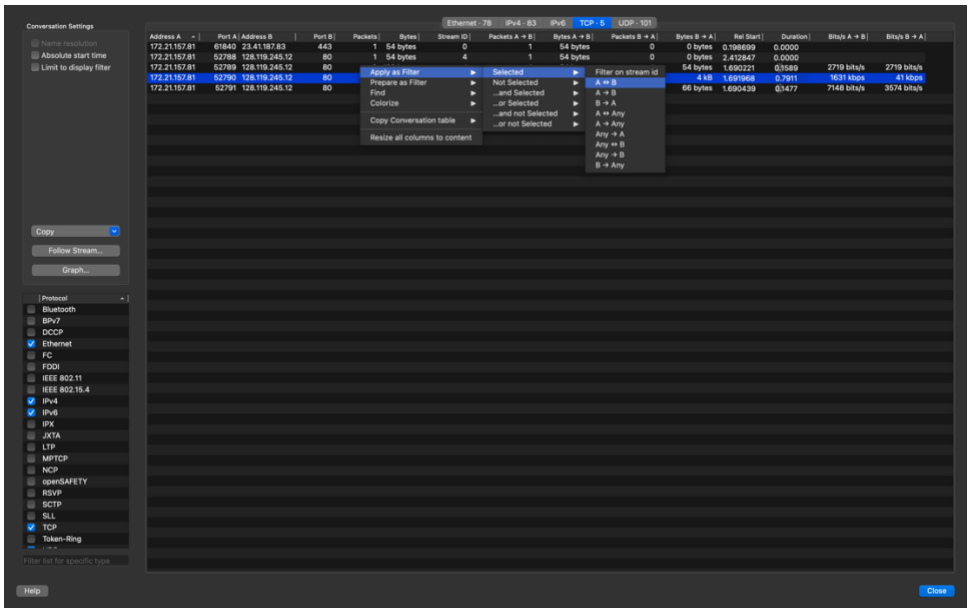

Conversation Settings
Name resolution
Absolute start time
Limit to display filter
Ethernet · 78
IPv4 · 83
IPv6
TCP · 5
UDP · 101
Address A
Port A
Address B
Port B
Packets
Bytes
Stream ID
Packets A → B
Bytes A → B
Packets B → A
Bytes B → A
Rel Start
Duration
Bits/s A → B
Bits/s B → A
172.21.157.81 61840 23.41.187.83 443 1 54 bytes 0 1 54 bytes 0 0 bytes 0.198699 0.0000
172.21.157.81 52788 128.119.245.12 80 1 54 bytes 4 1 54 bytes 0 0 bytes 2.412847 0.0000
172.21.157.81 52789 128.119.245.12 80 54 bytes 1.690221 0.3589 2719 bits/s 2719 bits/s
172.21.157.81 52790 128.119.245.12 80 4 kB 1.691968 0.7911 1631 kbps 41 kbps
172.21.157.81 52791 128.119.245.12 80 66 bytes 1.690439 0.3477 7148 bits/s 3574 bits/s
Apply as Filter
Prepare as Filter
Find
Colorize
Copy Conversation table
Resize all columns to content
Selected
Not Selected
...and Selected
...or Selected
...and not Selected
...or not Selected
Filter on stream id
A ↔ B
A → B
B → A
A ↔ Any
A → Any
Any → A
Any ↔ B
Any → B
B → Any
Copy
Follow Stream...
Graph...
Protocol
Bluetooth
BPv7
DCCP
Ethernet
FC
FDDI
IEEE 802.11
IEEE 802.15.4
IPv4
IPv6
IPX
JXTA
LTP
MPTCP
NCP
openSAFETY
RSVP
SCTP
SLL
TCP
Token-Ring
Filter list for specific type
Help
Close

Fig1

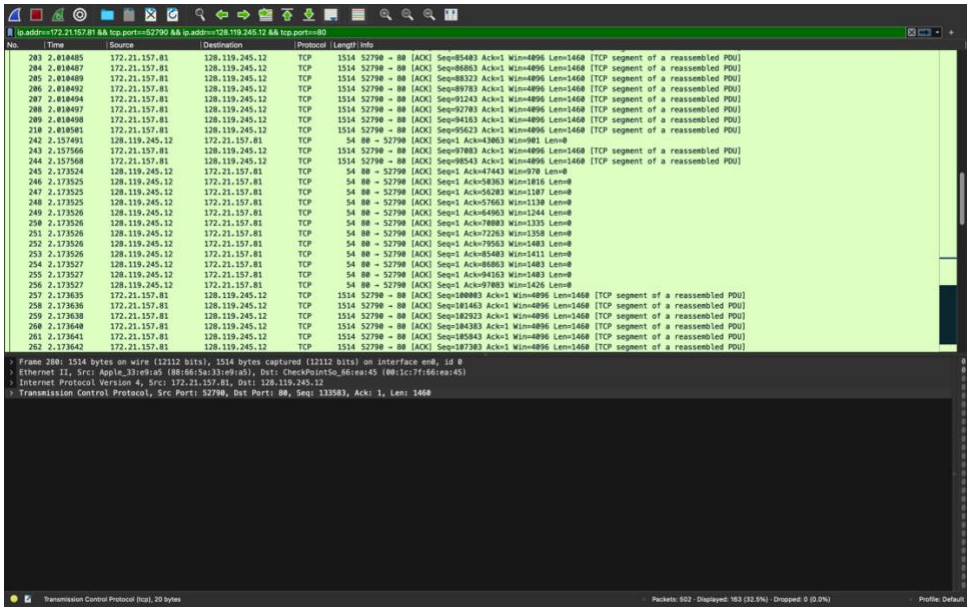

ip.addr==172.21.157.81 && tcp.port==52790 && ip.addr==128.119.245.12 && tcp.port==80
No. Time Source Destination Protocol Length Info
203 2.010485 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=85403 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
204 2.010487 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=86863 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
205 2.010489 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=88323 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
206 2.010492 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=89783 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
207 2.010494 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=91243 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
208 2.010497 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=92703 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
209 2.010498 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=94163 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
210 2.010501 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=95623 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
242 2.157491 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=43803 Win=981 Len=0
243 2.157566 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=97083 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
244 2.157568 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=98543 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
245 2.173524 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=47443 Win=970 Len=0
246 2.173525 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=50363 Win=1016 Len=0
247 2.173525 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=56203 Win=1107 Len=0
248 2.173525 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=57663 Win=1130 Len=0
249 2.173526 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=64963 Win=1244 Len=0
250 2.173526 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=70803 Win=1335 Len=0
251 2.173526 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=72263 Win=1358 Len=0
252 2.173526 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=79563 Win=1483 Len=0
253 2.173526 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=85403 Win=1411 Len=0
254 2.173527 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=86863 Win=1483 Len=0
255 2.173527 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=94163 Win=1483 Len=0
256 2.173527 128.119.245.12 172.21.157.81 TCP 54 80 → 52790 [ACK] Seq=1 Ack=97083 Win=1426 Len=0
257 2.173635 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=100003 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
258 2.173636 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=101463 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
259 2.173638 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=102923 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
260 2.173640 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=104383 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
261 2.173641 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=105843 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
262 2.173642 172.21.157.81 128.119.245.12 TCP 1514 52790 → 80 [ACK] Seq=107303 Ack=1 Win=4096 Len=1460 [TCP segment of a reassembled PDU]
Frame 280: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface en0, id 0
Ethernet II, Src: Apple_33:e9:a5 (88:66:5a:33:e9:a5), Dst: CheckPointSo_66:ea:45 (00:1c:7f:66:ea:45)
Internet Protocol Version 4, Src: 172.21.157.81, Dst: 128.119.245.12
Transmission Control Protocol, Src Port: 52790, Dst Port: 80, Seq: 133503, Ack: 1, Len: 1460
Transmission Control Protocol (tcp), 20 bytes
Packets: 502 · Displayed: 163 (32.5%) · Dropped: 0 (0.0%)
Profile: Default

Fig2

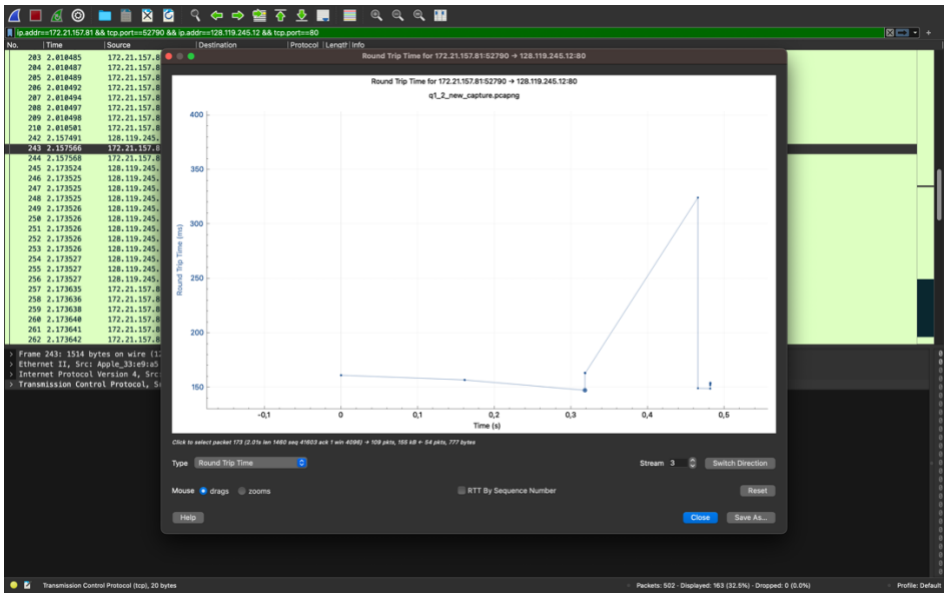


Fig3

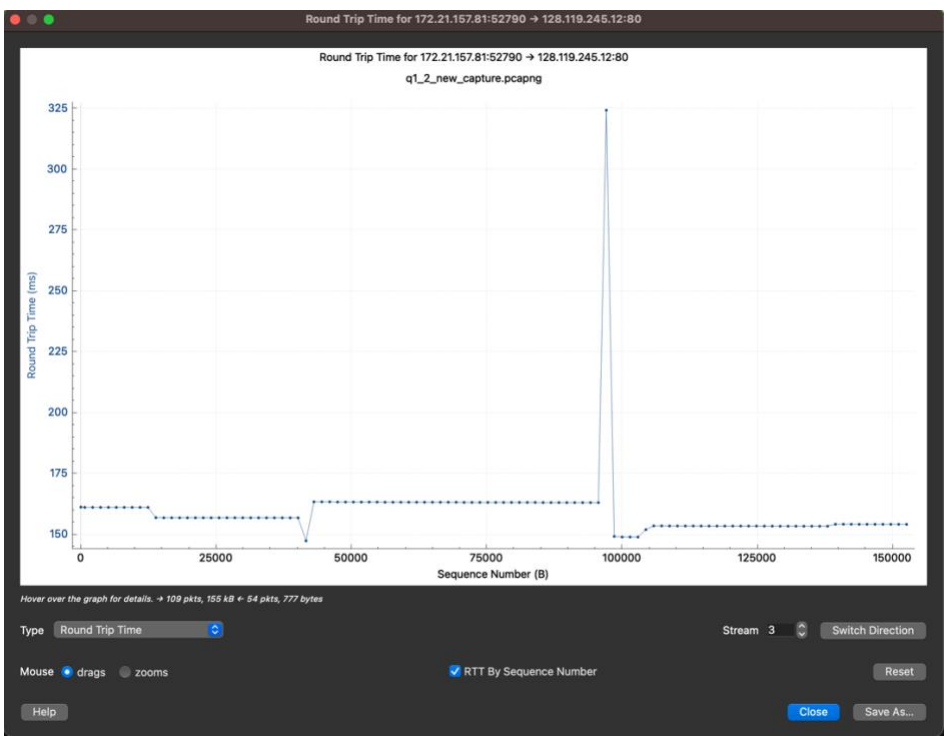


Fig4

3- We know that while TCP provides connection-oriented reliable transport, UDP provides connetionless transport. In addition to that TCP ensures flow & congestion control while UDP does not. To provide this kinds of features TCP header has sequence numbers, acknowledgment numbers, and flag. Those informations serves the purpose of enabling

reliable, connection-oriented communication with features like error recovery, flow control, and congestion control. For example sequence numbers are used to order and reassemble segments, while acknowledgment numbers acknowledge received data. This facilitates reliable and ordered data transfer. You can see the screenhots for both UDP and TCP below.

- ACK (Acknowledge): When data is received and a connection has been established, the ACK flag is raised. It frequently appears alongside the SYN flag. To acknowledge receiving the SYN packet, for instance, the device responding to it sets the ACK flag during the three-way handshake.
- SYN(Synchronize): Two devices can establish a connection by using the SYN flag. A device sends a TCP packet to the target device with the SYN flag set when it wishes to establish a TCP connection. Handshake process starts with this.
- FIN (Finish): A TCP connection can be terminated by setting the FIN flag 1. A device sends a TCP packet with the FIN flag set when it wishes to terminate a connection.
- Window Size: As can be seen from the below screenshot window present in TCP header, indicates the amount of data that the receiver is willing to accept. This is important for flow & congestion control.

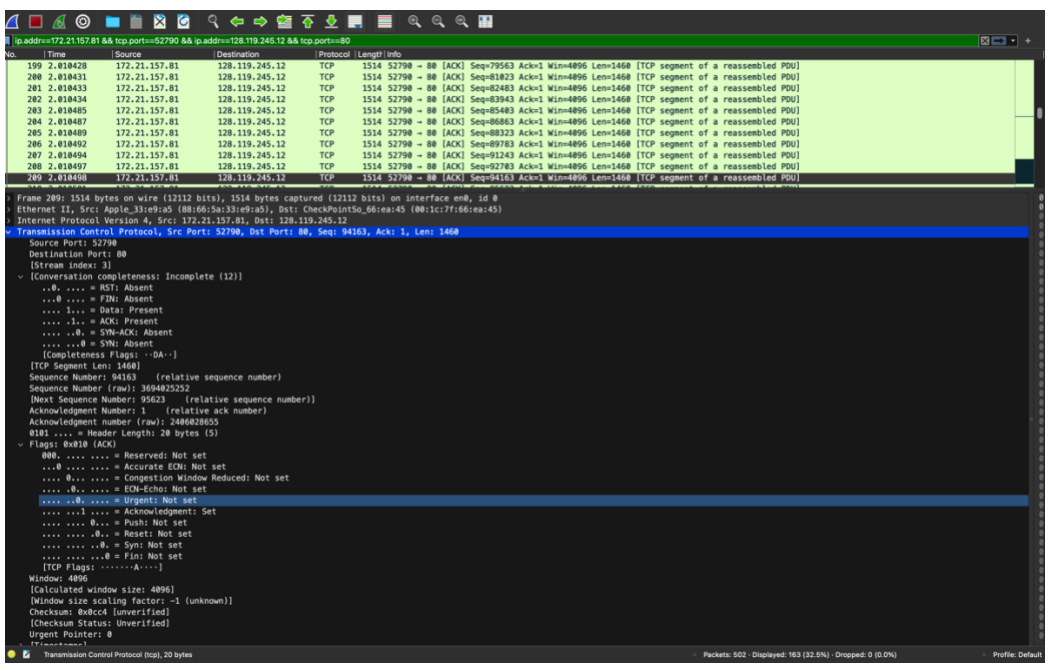


Fig1

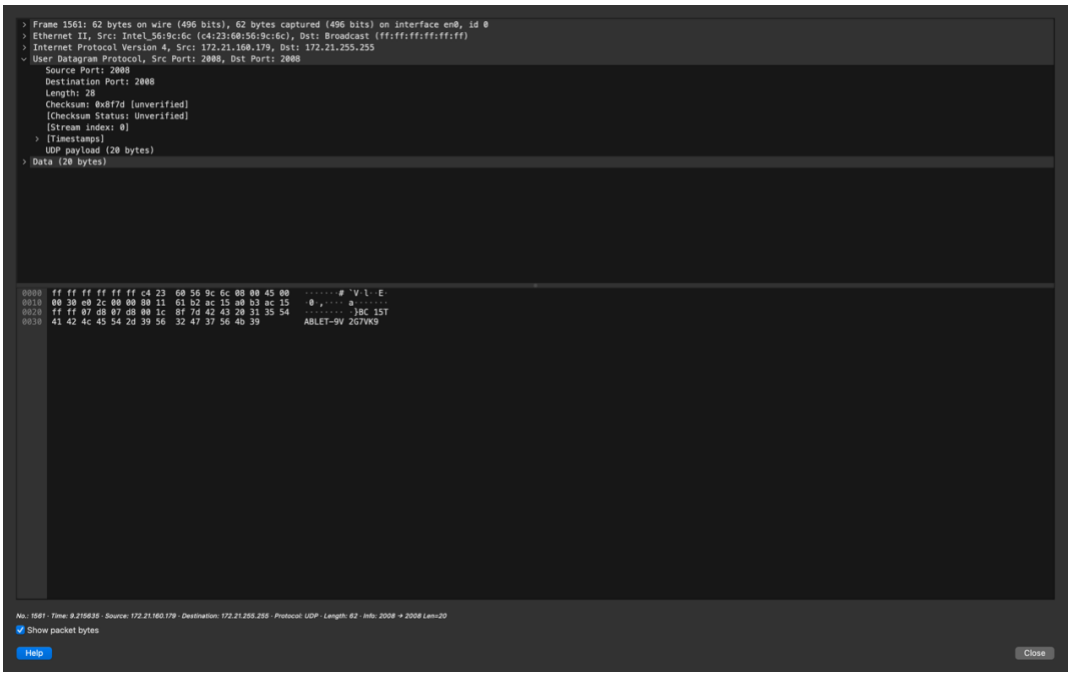


Fig2

4- In the screenshots below flow graphs for UDP and TCP can be seen. For both of them I included 3 screenshots. One of them is limiting to only display filter option is clicked and the other one is unclickled (which shows all of them). The third one shows only the TCP/UDP flow in the graph. First 4 screenshots for TCP and the last 4 for UDP. You can see the content of them in the 4th screenshot for each.

As can be seen below TCP diagram has PSH, ACK and connection closes with FIN,ACK at the end. The PSH flag instructs the receiving TCP stack to send the data it has received straight to the receiving application, bypassing the need to buffer additional data. While there is no ACK, FIN, PSH type of flagging in UDP since it is not reliable.

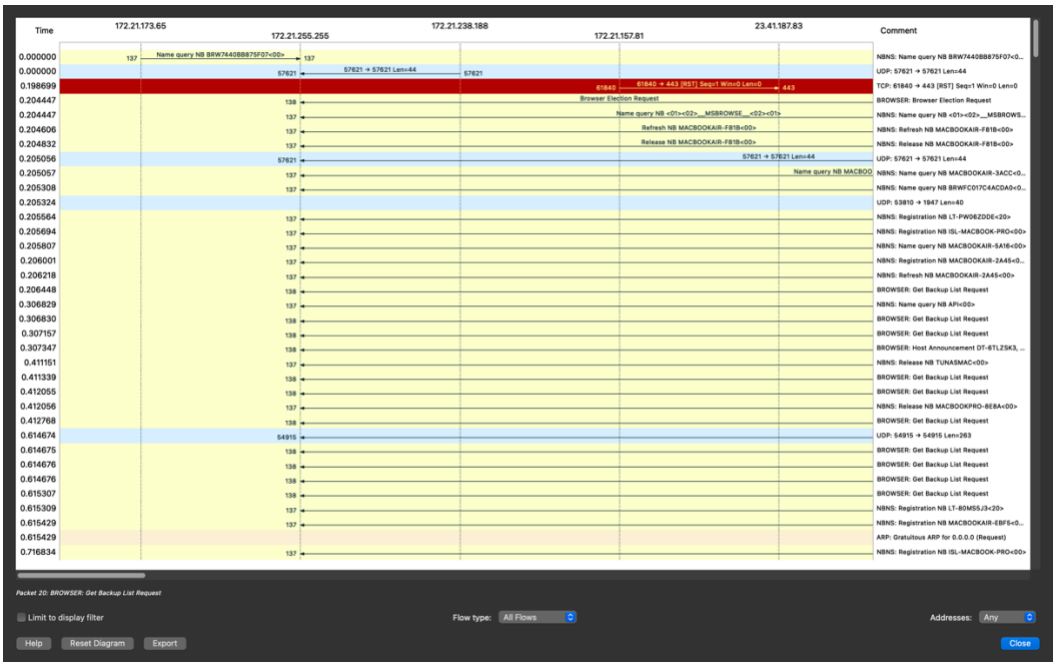

Time
172.21.173.65
172.21.255.255
172.21.238.188
172.21.157.81
23.41.187.83
Comment
Packet 20: BROWSER: Get Backup List Request
Limit to display filter
Flow type: All Flows
Addresses: Any
Help
Reset Diagram
Export
Close

Fig1

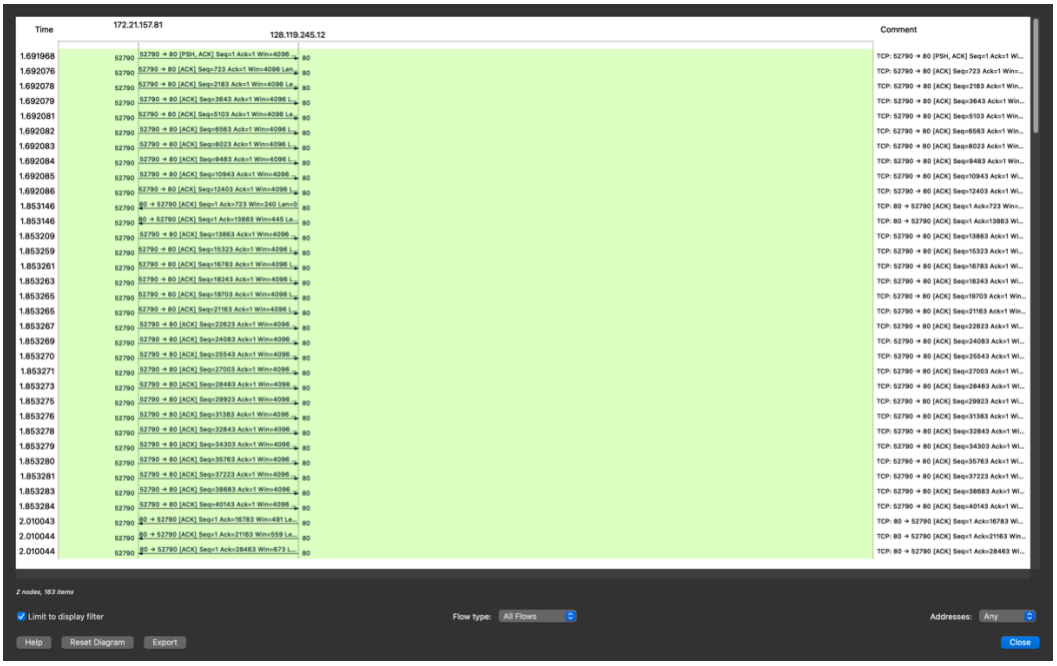

Time
172.21.157.81
128.119.245.12
Comment
2 nodes, 163 items
Limit to display filter
Flow type: All Flows
Addresses: Any
Help
Reset Diagram
Export
Close

Fig2

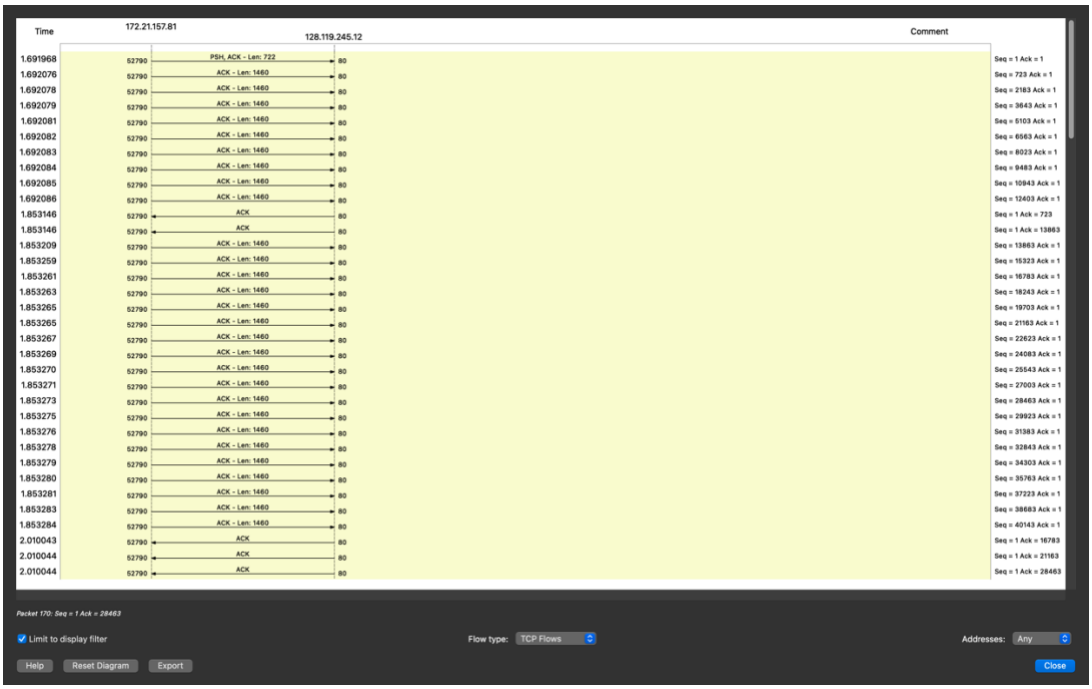

Time
172.21.157.81
128.119.245.12
Comment
Limit to display filter
Flow type: TCP Flows
Addresses: Any
Help
Reset Diagram
Export
Close

Fig3

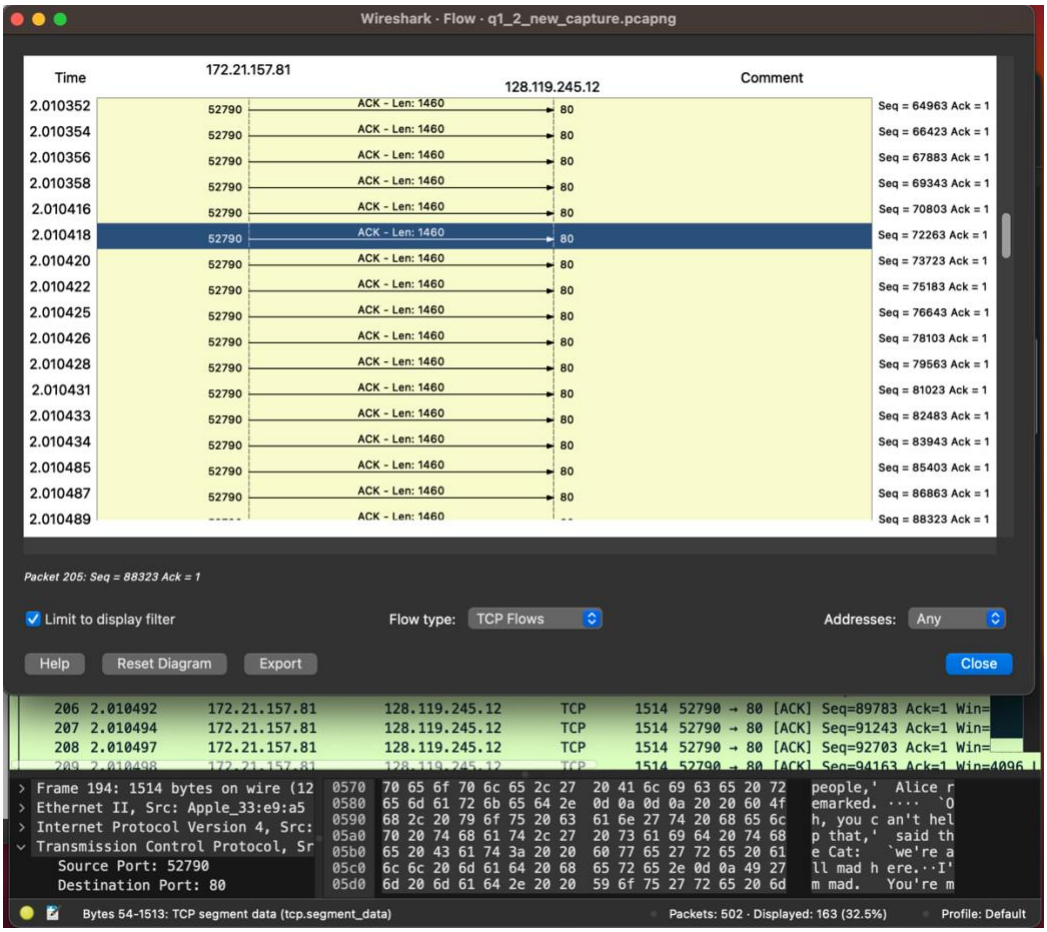

Wireshark · Flow · q1_2_new_capture.pcapng
Time
172.21.157.81
128.119.245.12
Comment
Packet 205: Seq = 88323 Ack = 1
Limit to display filter
Flow type: TCP Flows
Addresses: Any
Help
Reset Diagram
Export
Close
Frame 194: 1514 bytes on wire (12
Ethernet II, Src: Apple_33:e9:a5
Internet Protocol Version 4, Src:
Transmission Control Protocol, Sr
Source Port: 52790
Destination Port: 80
Bytes 54-1513: TCP segment data (tcp.segment_data)
Packets: 502 · Displayed: 163 (32.5%)
Profile: Default

Fig4

UDP:

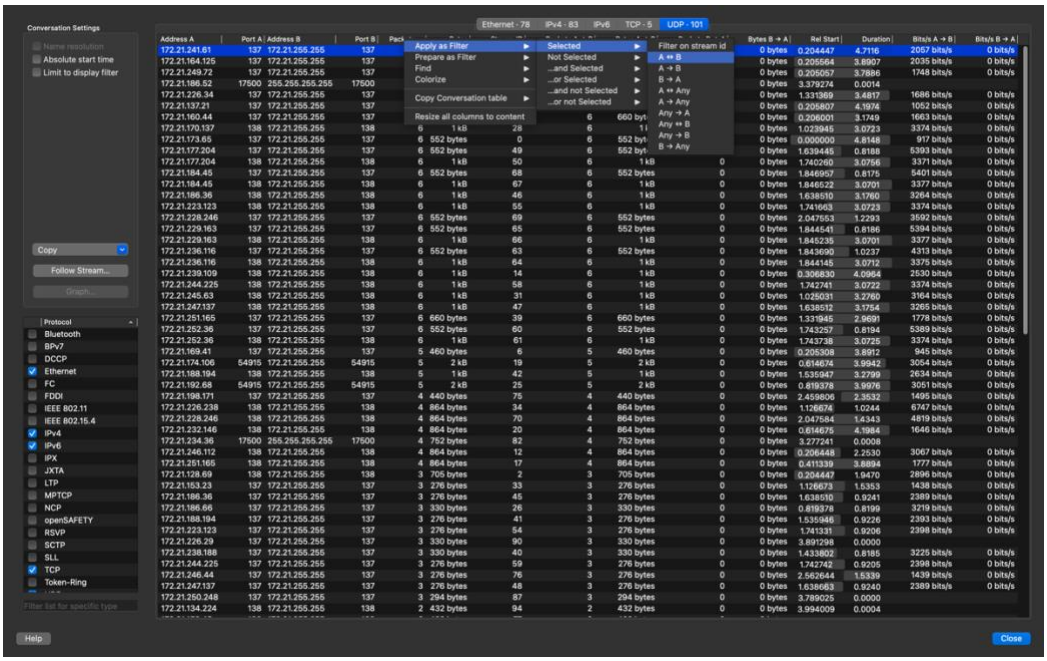


Fig1

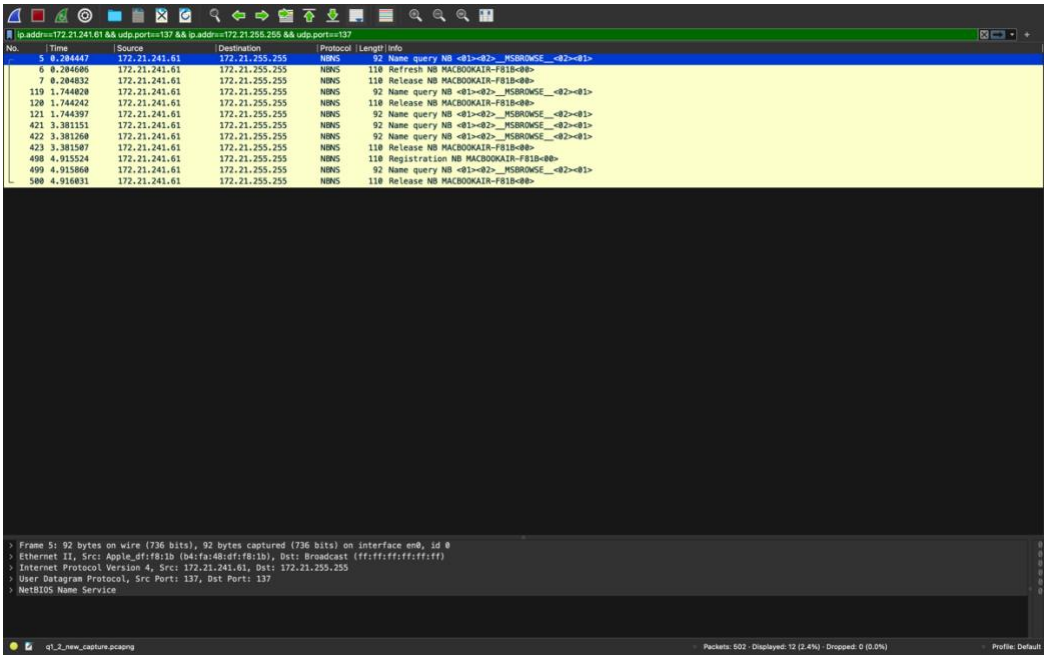


Fig2

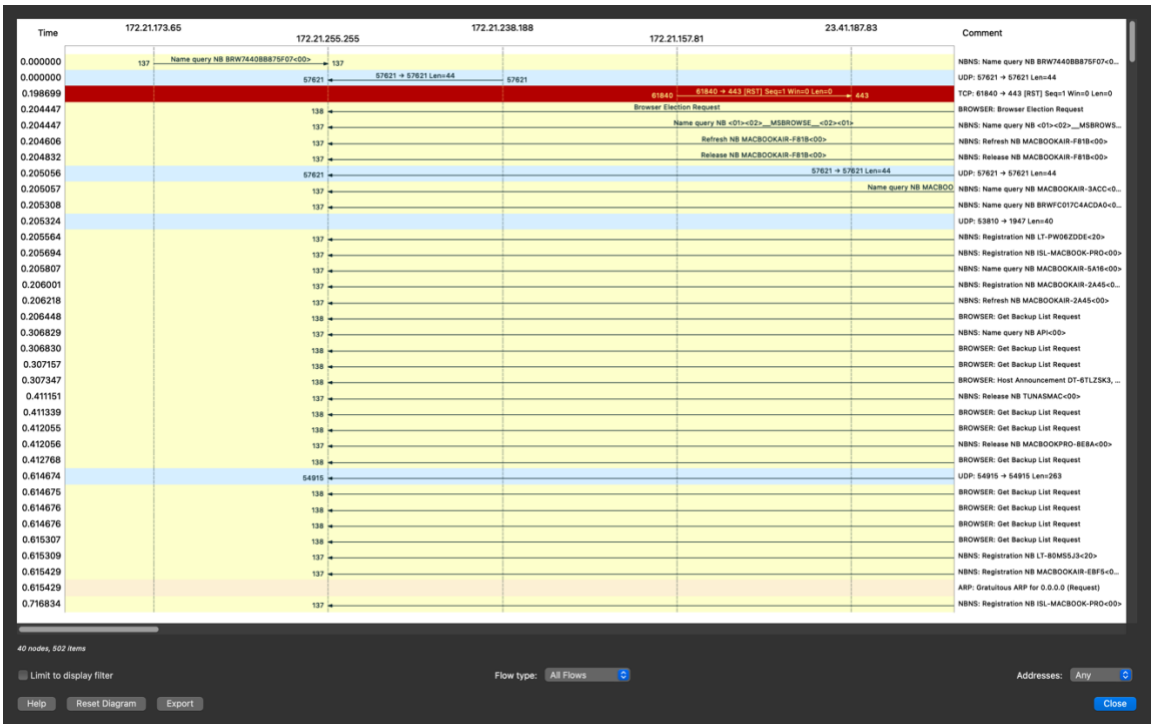

Time
172.21.173.65
172.21.255.255
172.21.238.188
172.21.157.81
23.41.187.83
Comment
0.000000 137 Name query NB BRW7440BB875F07<00> 137 NBNS: Name query NB BRW7440BB875F07<0...
0.000000 57621 57621 → 57621 Len=44 57621 UDP: 57621 → 57621 Len=44
0.198699 61840 61840 → 443 [RST] Seq=1 Win=0 Len=0 443 TCP: 61840 → 443 [RST] Seq=1 Win=0 Len=0
0.204447 138 Browser Election Request BROWSER: Browser Election Request
0.204447 137 Name query NB <01><02>__MSBROWSE__<02><01> NBNS: Name query NB <01><02>__MSBROWS...
0.204606 137 Refresh NB MACBOOKAIR-F81B<00> NBNS: Refresh NB MACBOOKAIR-F81B<00>
0.204832 137 Release NB MACBOOKAIR-F81B<00> NBNS: Release NB MACBOOKAIR-F81B<00>
0.205056 57621 57621 → 57621 Len=44 UDP: 57621 → 57621 Len=44
0.205057 137 Name query NB MACBOO NBNS: Name query NB MACBOOKAIR-3ACC<0...
0.205308 137 NBNS: Name query NB BRWFC017C4ACDA0<0...
0.205324 UDP: 53810 → 1947 Len=40
0.205564 137 NBNS: Registration NB LT-PW06ZDDE<20>
0.205694 137 NBNS: Registration NB ISL-MACBOOK-PRO<00>
0.205807 137 NBNS: Name query NB MACBOOKAIR-5A16<00>
0.206001 137 NBNS: Registration NB MACBOOKAIR-2A45<0...
0.206218 137 NBNS: Refresh NB MACBOOKAIR-2A45<00>
0.206448 138 BROWSER: Get Backup List Request
0.306829 137 NBNS: Name query NB API<00>
0.306830 138 BROWSER: Get Backup List Request
0.307157 138 BROWSER: Get Backup List Request
0.307347 138 BROWSER: Host Announcement DT-6TLZSK3, ...
0.411151 137 NBNS: Release NB TUNASMAC<00>
0.411339 138 BROWSER: Get Backup List Request
0.412055 138 BROWSER: Get Backup List Request
0.412056 137 NBNS: Release NB MACBOOKPRO-8E8A<00>
0.412768 138 BROWSER: Get Backup List Request
0.614674 54915 UDP: 54915 → 54915 Len=263
0.614675 138 BROWSER: Get Backup List Request
0.614676 138 BROWSER: Get Backup List Request
0.614676 138 BROWSER: Get Backup List Request
0.615307 138 BROWSER: Get Backup List Request
0.615309 137 NBNS: Registration NB LT-80M55J3<20>
0.615429 137 NBNS: Registration NB MACBOOKAIR-EBF5<0...
0.615429 ARP: Gratuitous ARP for 0.0.0.0 (Request)
0.716834 137 NBNS: Registration NB ISL-MACBOOK-PRO<00>
40 nodes, 502 items
Limit to display filter
Flow type: All Flows
Addresses: Any
Help
Reset Diagram
Export
Close

Fig3

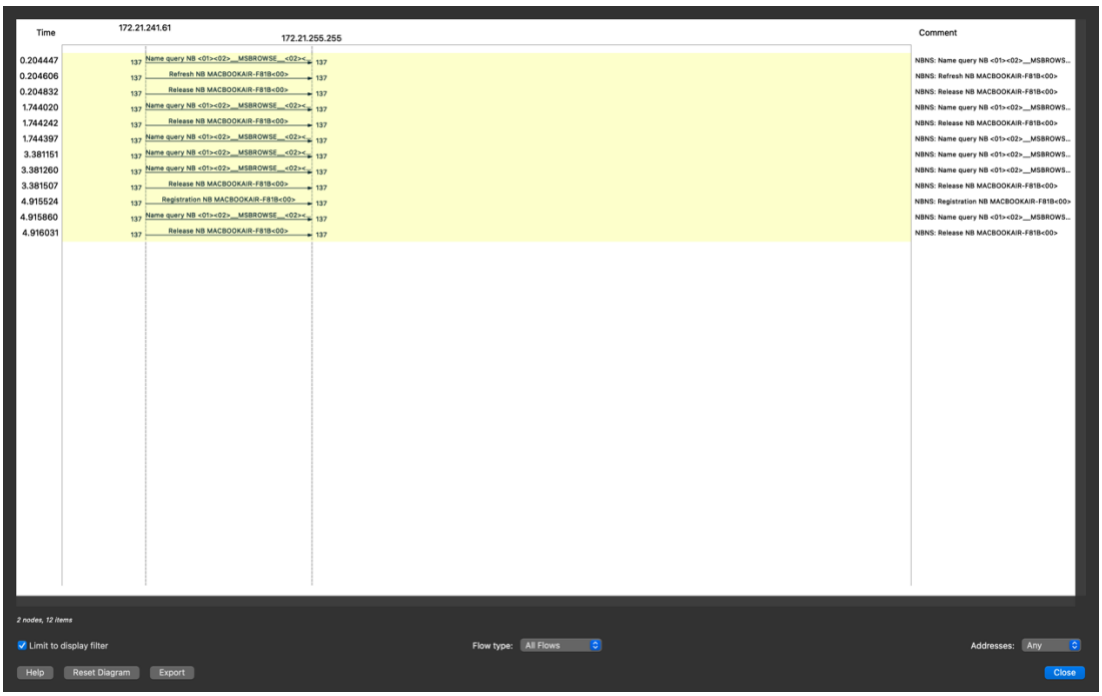

Time
172.21.241.61
172.21.255.255
Comment
0.204447 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
0.204606 137 Refresh NB MACBOOKAIR-F81B<00> 137 NBNS: Refresh NB MACBOOKAIR-F81B<00>
0.204832 137 Release NB MACBOOKAIR-F81B<00> 137 NBNS: Release NB MACBOOKAIR-F81B<00>
1.744020 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
1.744242 137 Release NB MACBOOKAIR-F81B<00> 137 NBNS: Release NB MACBOOKAIR-F81B<00>
1.744397 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
3.381151 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
3.381260 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
3.381507 137 Release NB MACBOOKAIR-F81B<00> 137 NBNS: Release NB MACBOOKAIR-F81B<00>
4.915524 137 Registration NB MACBOOKAIR-F81B<00> 137 NBNS: Registration NB MACBOOKAIR-F81B<00>
4.915860 137 Name query NB <01><02>__MSBROWSE__<02><... 137 NBNS: Name query NB <01><02>__MSBROWS...
4.916031 137 Release NB MACBOOKAIR-F81B<00> 137 NBNS: Release NB MACBOOKAIR-F81B<00>
2 nodes, 12 items
Limit to display filter
Flow type: All Flows
Addresses: Any
Help
Reset Diagram
Export
Close

Fig4

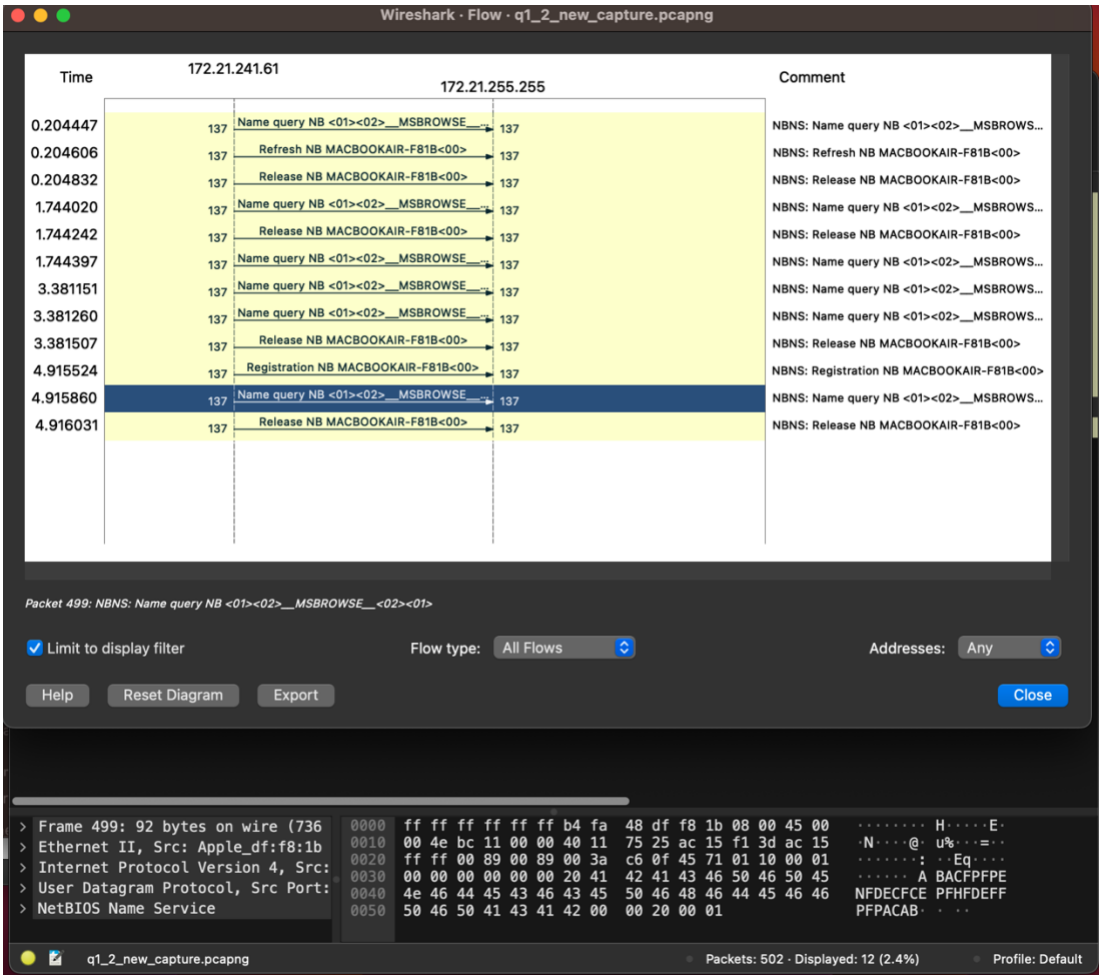


Fig5

Question 2.1

1- In my implementation there are 2 types of client. I mean based on the choice of client I initialize a class either TCP or SSL. While TCP client uses TCP, SSL Client uses TLS in addition to TCP which provides security by encoding the message. In the below screenshots you can see the process and outputs step by step.

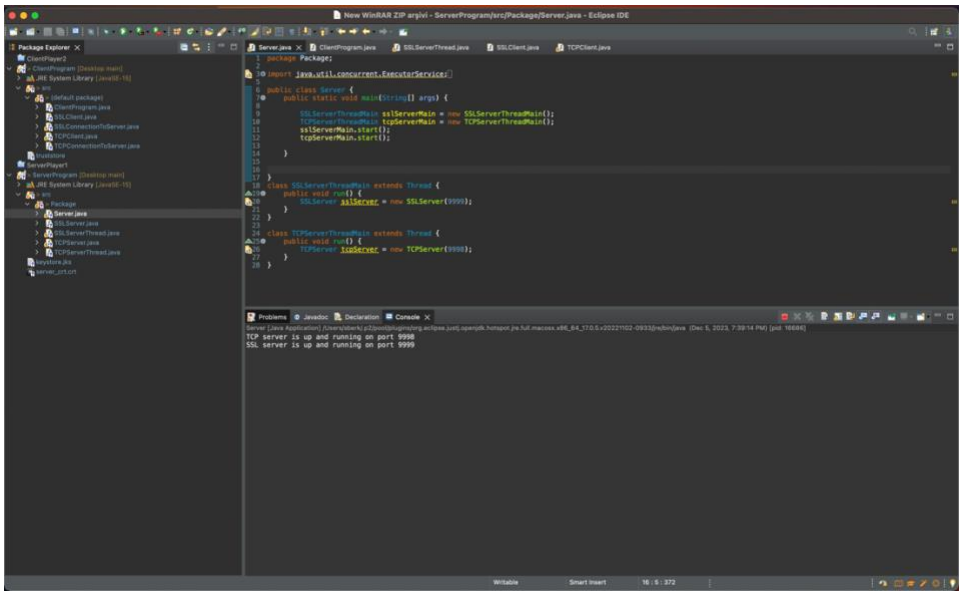

New WinRAR ZIP arşivi - ServerProgram/src/Package/Server.java - Eclipse IDE
Package Explorer
Server.java
ClientProgram.java
SSLServerThread.java
SSLClient.java
TCPClient.java
import java.util.concurrent.ExecutorService;
public class Server {
public static void main(String[] args) {
SSLServerThreadMain sslServerMain = new SSLServerThreadMain();
TCPServerThreadMain tcpServerMain = new TCPServerThreadMain();
sslServerMain.start();
tcpServerMain.start();
}
}
class SSLServerThreadMain extends Thread {
public void run() {
SSLServer sslServer = new SSLServer(9999);
}
}
class TCPServerThreadMain extends Thread {
public void run() {
TCPServer tcpServer = new TCPServer(9998);
}
}
Problems
Javadoc
Declaration
Console
TCP server is up and running on port 9998
SSL server is up and running on port 9999

Fig1

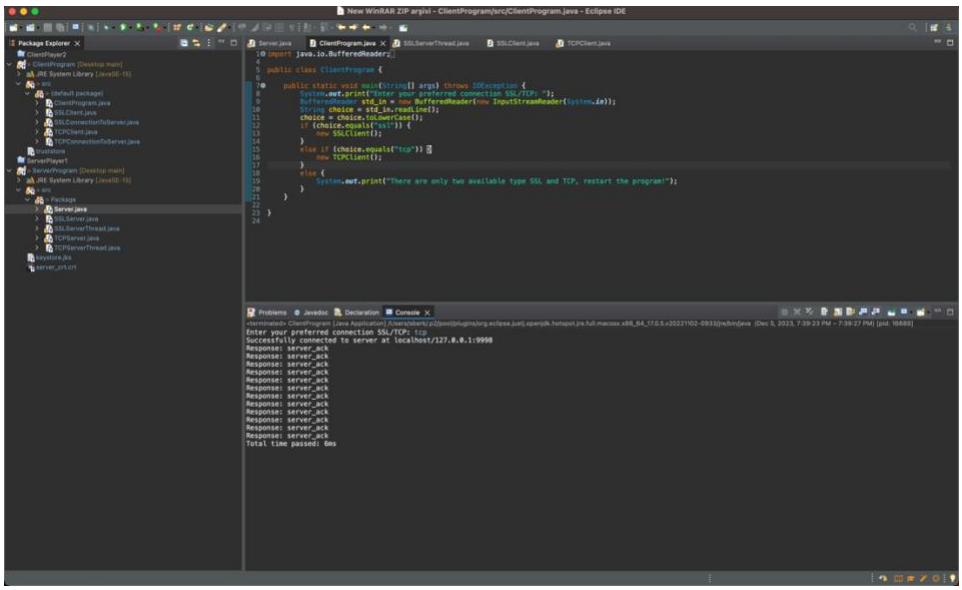

New WinRAR ZIP arşivi - ClientProgram/src/ClientProgram.java - Eclipse IDE
import java.io.BufferedReader;
public class ClientProgram {
public static void main(String[] args) throws IOException {
System.out.print("Enter your preferred connection SSL/TCP: ");
BufferedReader std_in = new BufferedReader(new InputStreamReader(System.in));
String choice = std_in.readLine();
choice = choice.toLowerCase();
if (choice.equals("ssl")) {
new SSLClient();
}
else if (choice.equals("tcp")) {
new TCPClient();
}
else {
System.out.print("There are only two available type SSL and TCP, restart the program!");
}
}
}
Enter your preferred connection SSL/TCP: tcp
Successfully connected to server at localhost/127.0.0.1:9998
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 6ms

Fig2

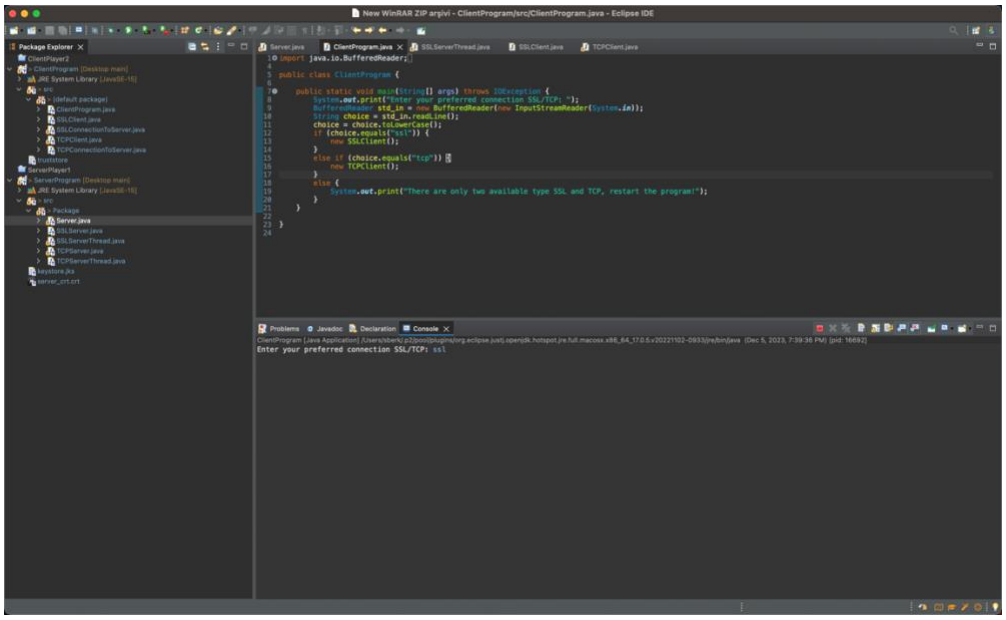

import java.io.BufferedReader;
public class ClientProgram {
public static void main(String[] args) throws IOException {
System.out.print("Enter your preferred connection SSL/TCP: ");
BufferedReader std_in = new BufferedReader(new InputStreamReader(System.in));
String choice = std_in.readLine();
choice = choice.toLowerCase();
if (choice.equals("ssl")) {
new SSLClient();
}
else if (choice.equals("tcp")) {
new TCPClient();
}
else {
System.out.print("There are only two available type SSL and TCP, restart the program!");
}
}
}
Enter your preferred connection SSL/TCP: ssl

Fig3

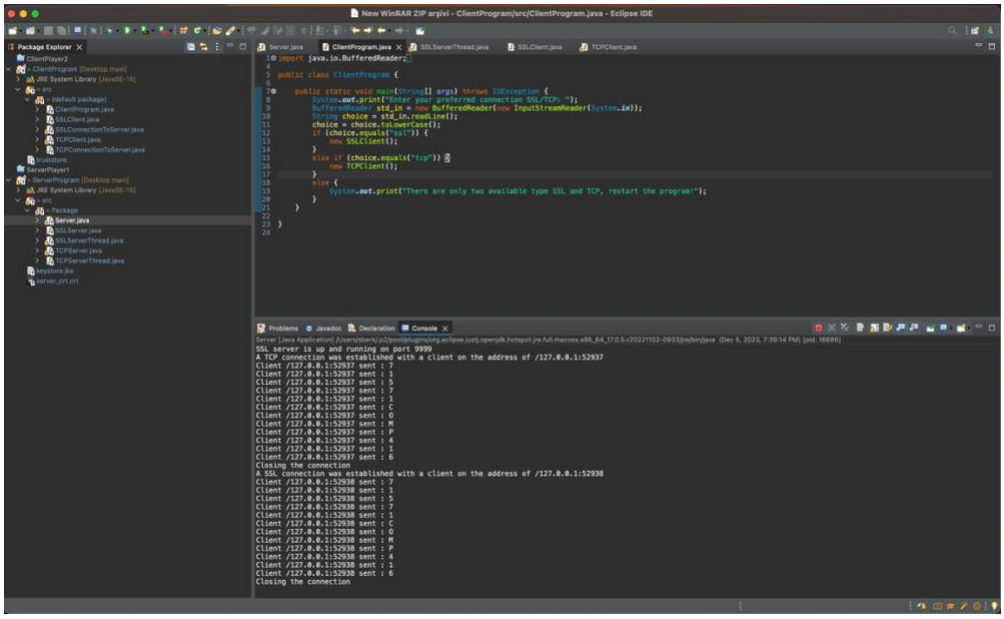

SSL server is up and running on port 9999
A TCP connection was established with a client on the address of /127.0.0.1:52937
Client /127.0.0.1:52937 sent : 7
Client /127.0.0.1:52937 sent : 1
Client /127.0.0.1:52937 sent : 5
Client /127.0.0.1:52937 sent : 7
Client /127.0.0.1:52937 sent : 1
Client /127.0.0.1:52937 sent : C
Client /127.0.0.1:52937 sent : O
Client /127.0.0.1:52937 sent : M
Client /127.0.0.1:52937 sent : P
Client /127.0.0.1:52937 sent : 4
Client /127.0.0.1:52937 sent : 1
Client /127.0.0.1:52937 sent : 6
Closing the connection
A SSL connection was established with a client on the address of /127.0.0.1:52938
Client /127.0.0.1:52938 sent : 7
Client /127.0.0.1:52938 sent : 1
Client /127.0.0.1:52938 sent : 5
Client /127.0.0.1:52938 sent : 7
Client /127.0.0.1:52938 sent : 1
Client /127.0.0.1:52938 sent : C
Client /127.0.0.1:52938 sent : O
Client /127.0.0.1:52938 sent : M
Client /127.0.0.1:52938 sent : P
Client /127.0.0.1:52938 sent : 4
Client /127.0.0.1:52938 sent : 1
Client /127.0.0.1:52938 sent : 6
Closing the connection

Fig4

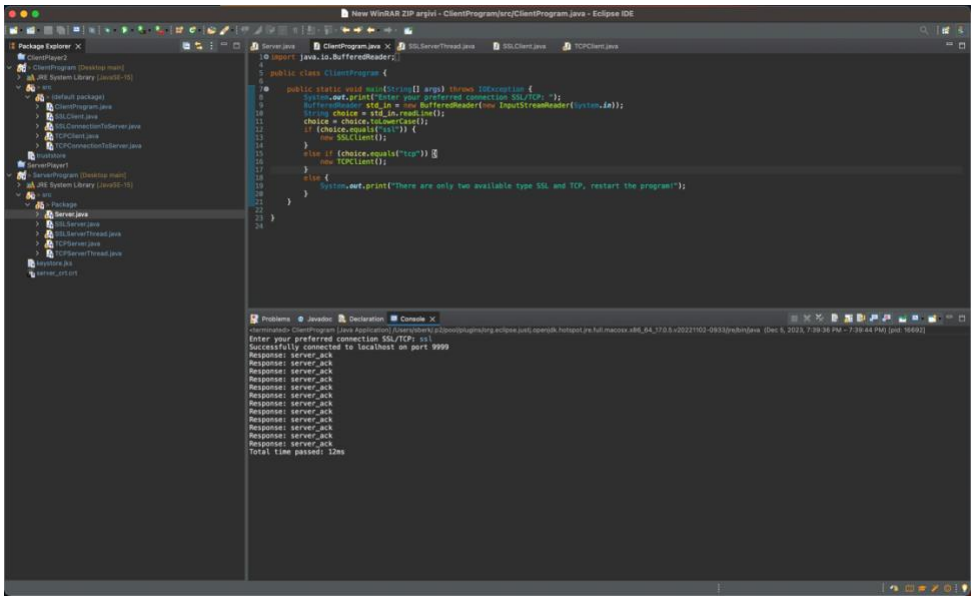


Fig5

2- In the screenshot below you can see the data from each connection. In TLS protocol since it is encrypted/secured we cannot understand(look at the first ss) while in TCP(no encryption) we can easily see and the read content. In TCP I look for the PSH flag since it contains the data in it. Screenshots from 2-5 shows some of the content that was send back and forth between TCP client and server. To see the data please look right bottom of the screenshots. Please also be aware that not all of the text is actual data it contains TCP header info, flags, window size etc. in Wireshark screenshots that I take, data is at the end.

- In the 2nd screenshot “server_ack” message that we sent is seen.
- In the third “7” seen. Which is included in my school number 71571.
- In the fourth “C” is seen. Which was included in our message.
- In the fifth “6” is seen 2 PSH before the FIN flag which denotes the end of conversation. The PSH right before the FIN flag is “server_ack” message.

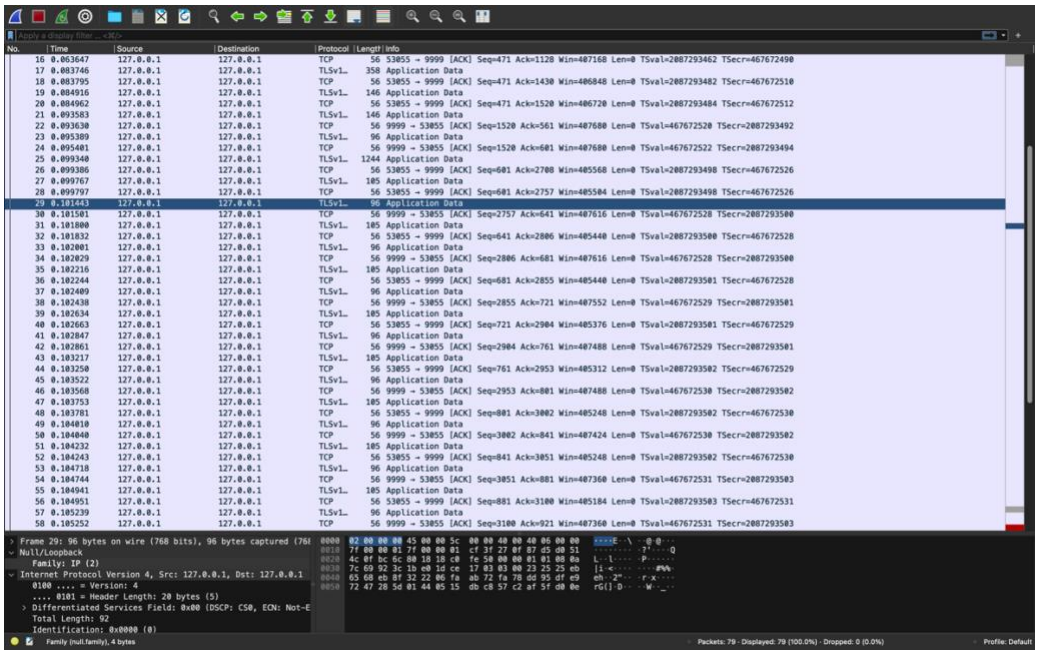


Fig1

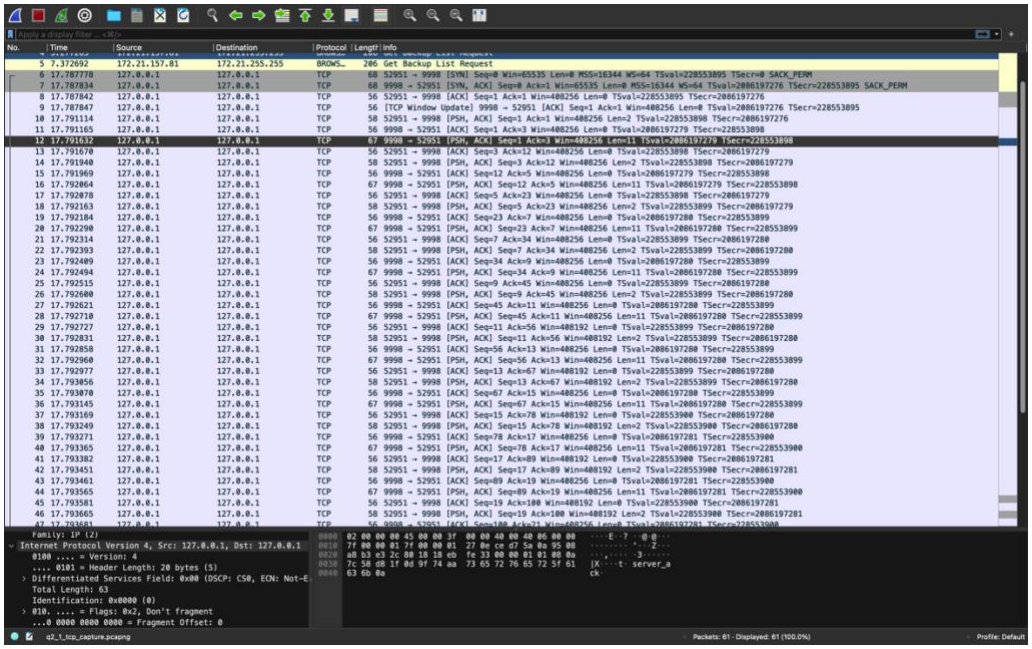


Fig1

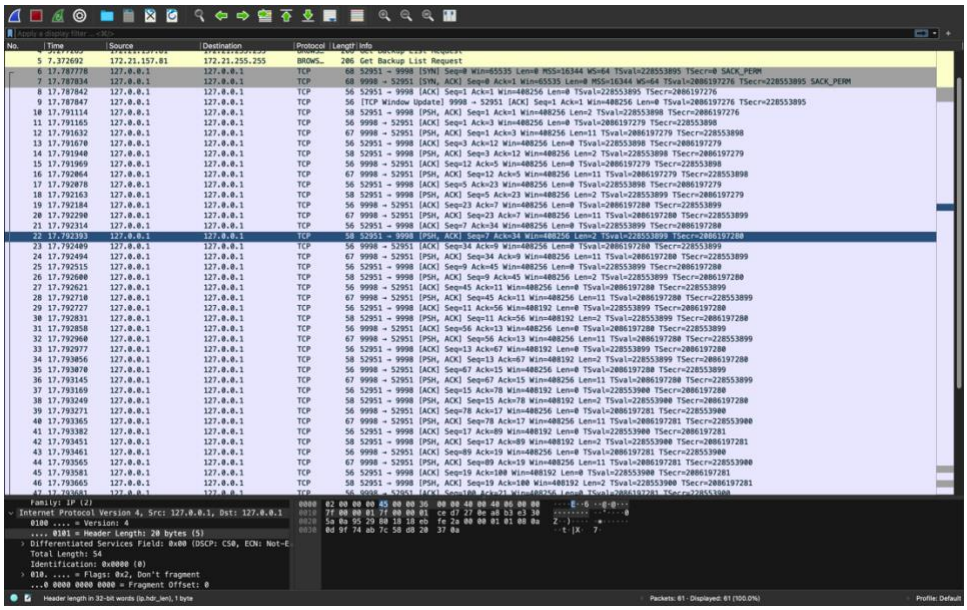


Fig2

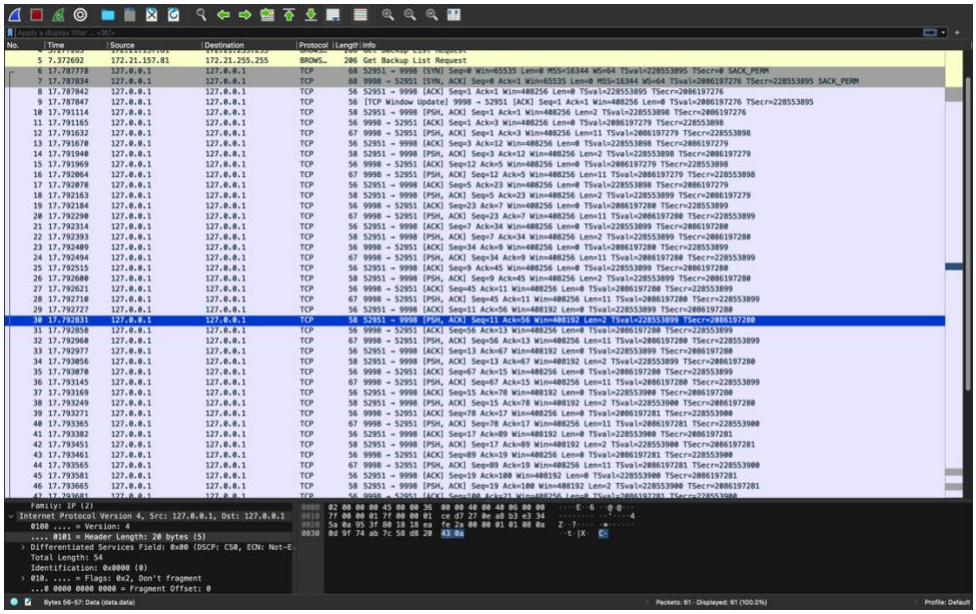


Fig3

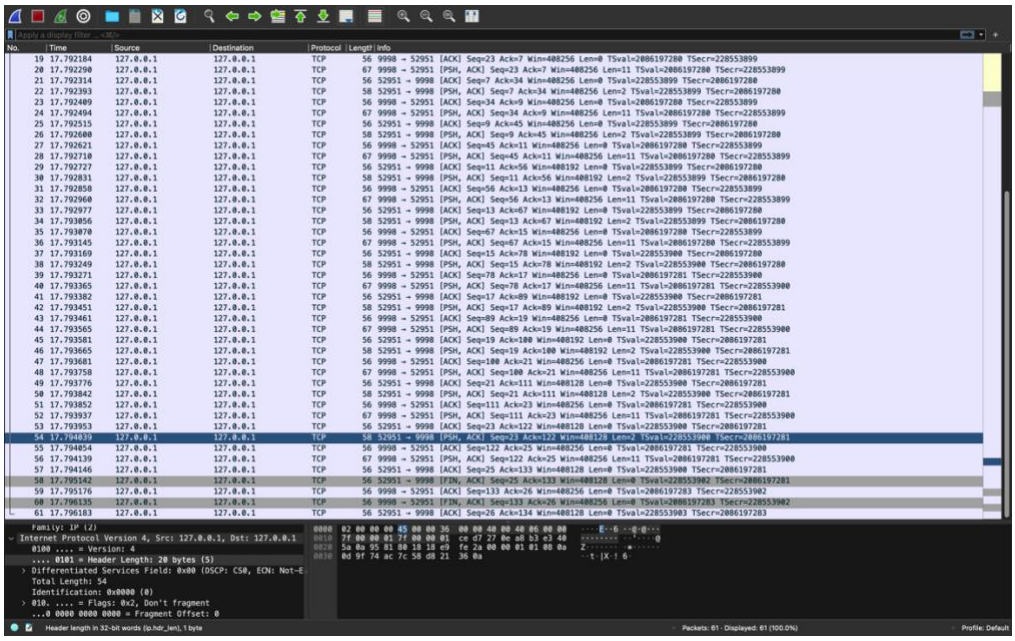


Fig4

3- From the above screenshots in TCP since it is not encrpyted we can read the content of the data easily and understand what is the message. However, since server does not send the data it received from the client to the client we cannot see inside the response message of a server. Inside the response message of a server we can only see server_ack. In TLS we cannot see none of them. S

Since we designed to send the text “71571COMP416” character by character in each packet, we can read the one character from the message in each packet that includes PSH flag. As can be seen from the first 2 screenshots two packet that includes PSH sent from client to the server includes “7” and “1” respectively. My school id starting with “71”. While in TLS we cannot see the content since it is encrypted and we could not see PSH flag.

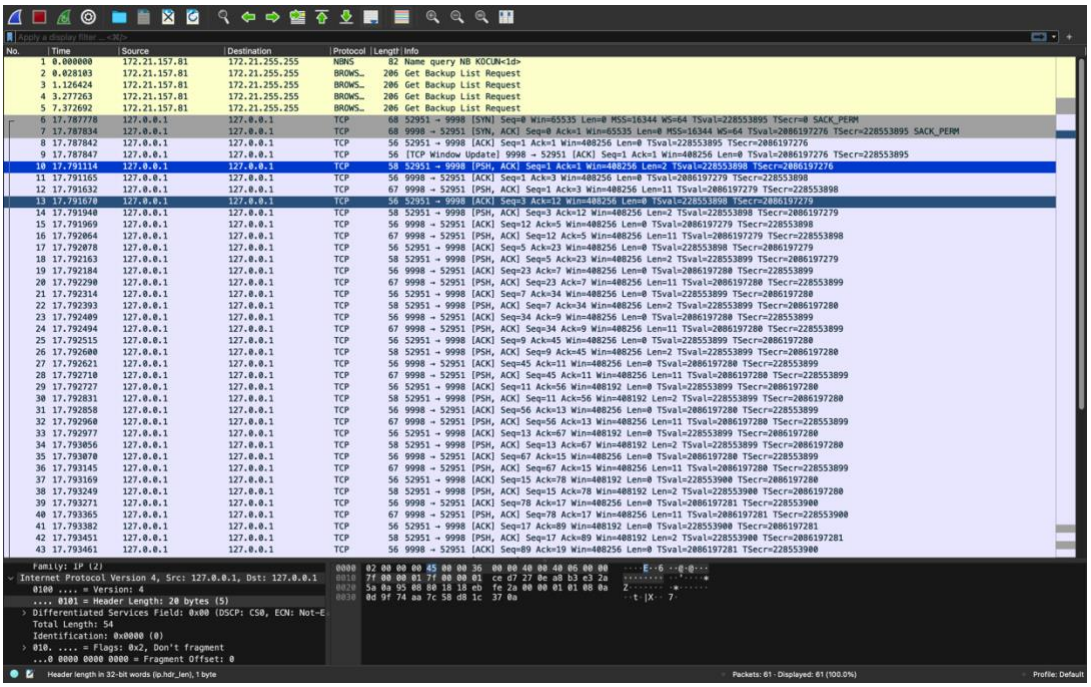


Fig1

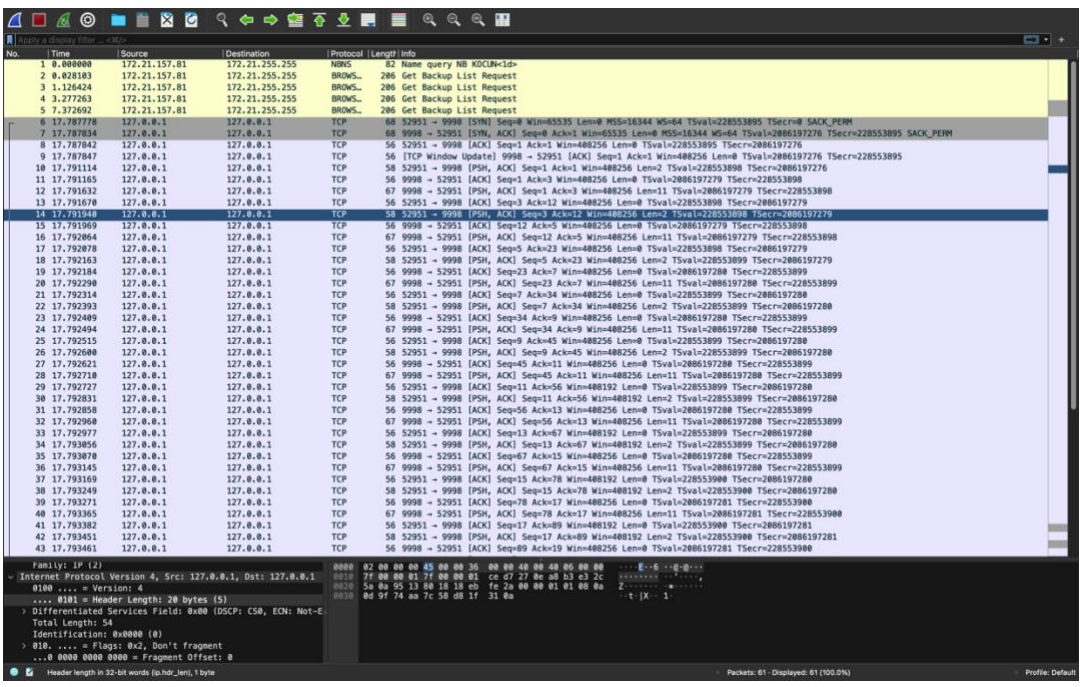
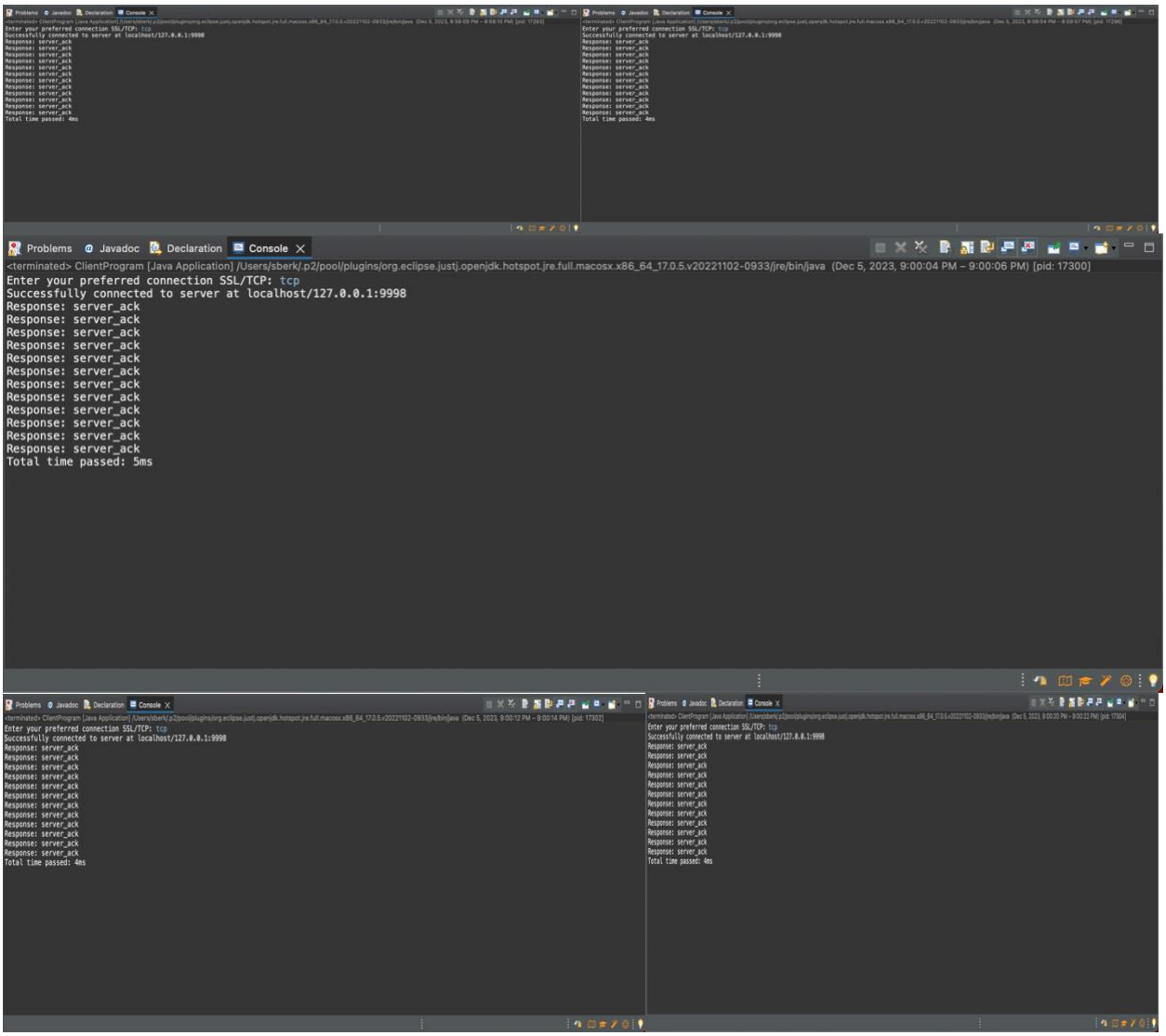


Fig2

4- Below in the screenshots you can see the results of 5 executions for both TCP and SSL. Graph is in the below as well. As expected SSL takes more time since encryption process takes time.

TCP Results: 4-4-5-4-4 = 23/5 => 4.6



TLS Results: 14-11-9-9-9 => 52/5=10.4

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> ClientProgram [Java Application] /Users/sberk/.p2/pool/plugins/org.eclip
Enter your preferred connection SSL/TCP: SSL
Successfully connected to localhost on port 9999
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 14ms
```

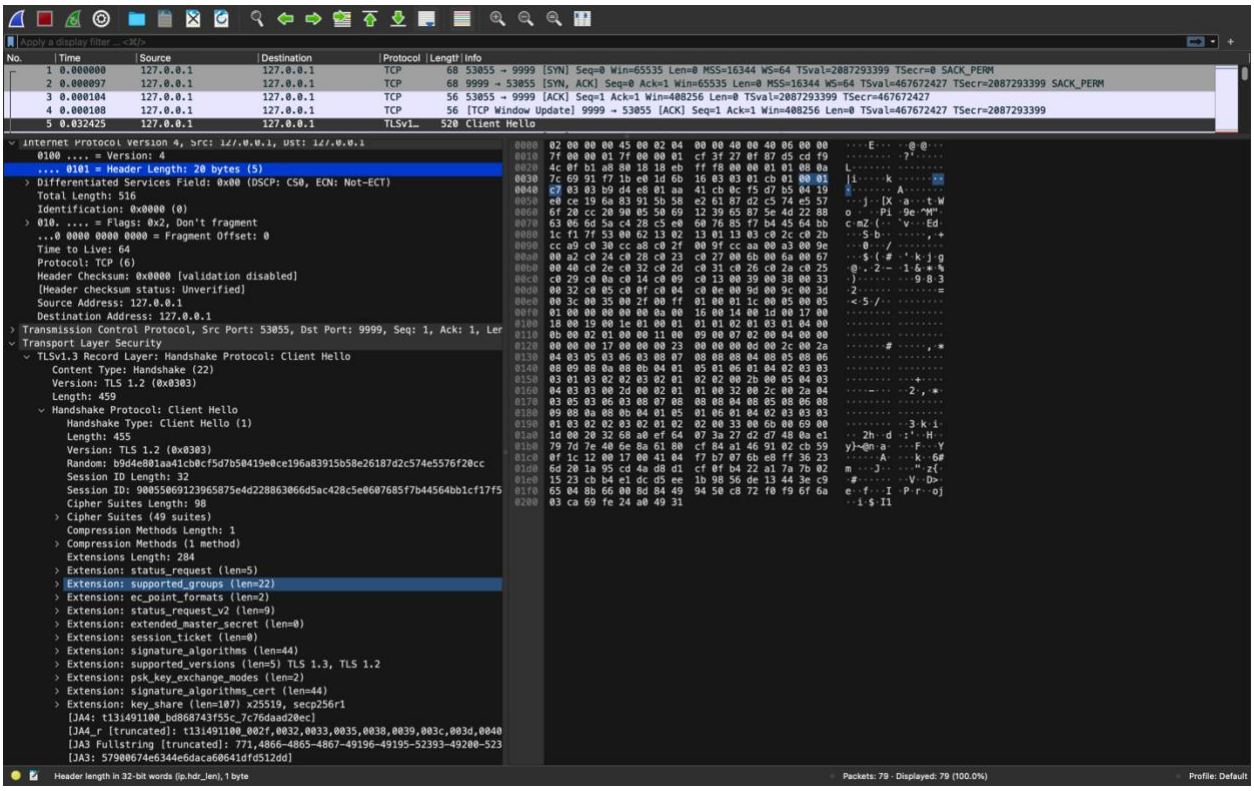
```
Problems  @ Javadoc  Declaration  Console ×
<terminated> ClientProgram [Java Application] /Users/sberk/.p2/pool/plugins/or
Enter your preferred connection SSL/TCP: ssl
Successfully connected to localhost on port 9999
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 11ms
```

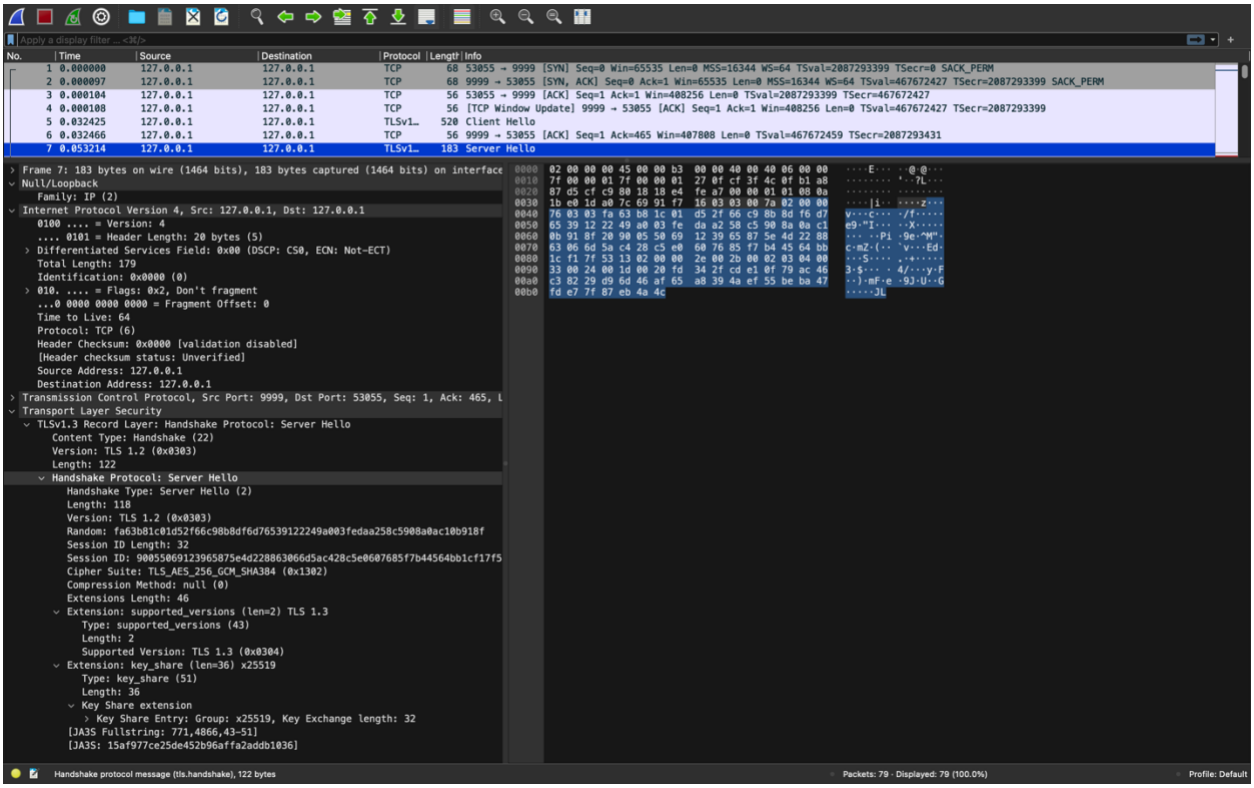
```
Problems  @ Javadoc  Declaration  Console ×
<terminated> ClientProgram [Java Application] /Users/sberk/.p2/pool/plugins/o
Enter your preferred connection SSL/TCP: ssl
Successfully connected to localhost on port 9999
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 9ms
```

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> ClientProgram [Java Application] /Users/sberk/.p2/pool/plugins/org.eclipse.just
Enter your preferred connection SSL/TCP: ssl
Successfully connected to localhost on port 9999
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 9ms
```

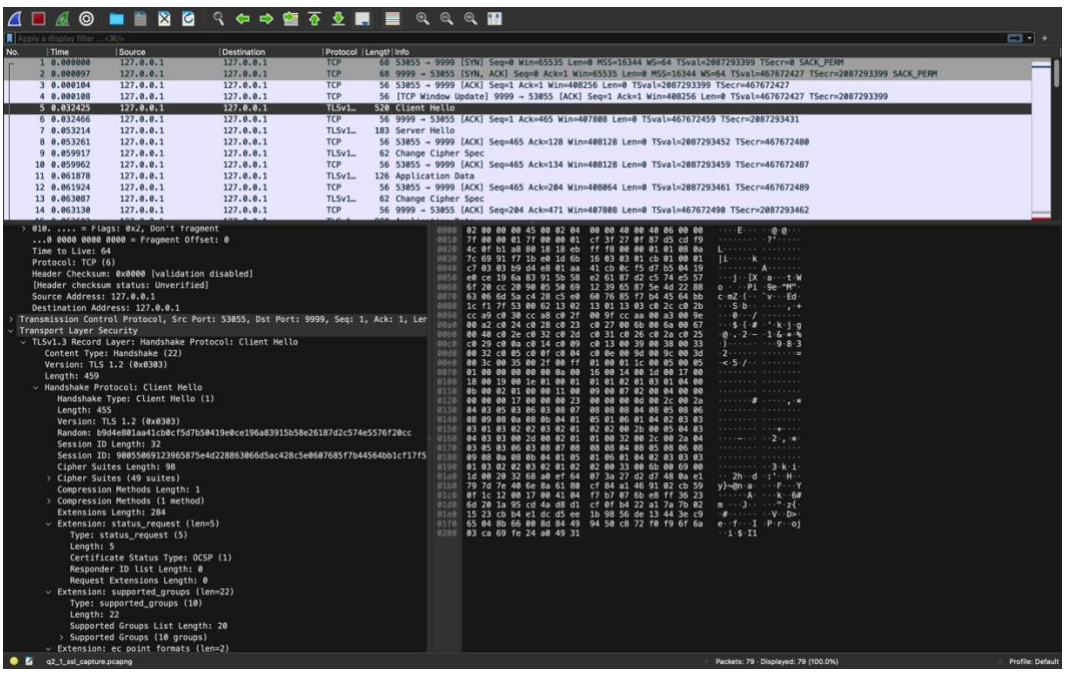
```
Problems  @ Javadoc  Declaration  Console ×
<terminated> ClientProgram [Java Application] /Users/sberk/.p2/pool/plugins/org.eclipse
Enter your preferred connection SSL/TCP: ssl
Successfully connected to localhost on port 9999
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Response: server_ack
Total time passed: 9ms
```

5- Client Hello and Server Hello packets are used for TCL handshake purpose. They are not present in TCP. As can be seen from the below screenshots they are Handshake Protocols. First with Client Hello packet client requests server a secure connection. It includes fields like sessionId, cipher suites, TLS version etc. At the end of handshaking process both parties agree on symmetric secret key.


Apply a display filter ... <⌘/>
No. Time Source Destination Protocol Length Info
1 0.000000 127.0.0.1 127.0.0.1 TCP 68 53055 → 9999 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=2087293399 TSecr=0 SACK_PERM
2 0.000097 127.0.0.1 127.0.0.1 TCP 68 9999 → 53055 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=467672427 TSecr=2087293399 SACK_PERM
3 0.000104 127.0.0.1 127.0.0.1 TCP 56 53055 → 9999 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=2087293399 TSecr=467672427
4 0.000108 127.0.0.1 127.0.0.1 TCP 56 [TCP Window Update] 9999 → 53055 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=467672427 TSecr=2087293399
5 0.032425 127.0.0.1 127.0.0.1 TLSv1… 520 Client Hello
Header length in 32-bit words (ip.hdr_len), 1 byte
Packets: 79 · Displayed: 79 (100.0%)
Profile: Default


Apply a display filter ... <⌘/>
No. Time Source Destination Protocol Length Info
1 0.000000 127.0.0.1 127.0.0.1 TCP 68 53055 → 9999 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=2087293399 TSecr=0 SACK_PERM
2 0.000097 127.0.0.1 127.0.0.1 TCP 68 9999 → 53055 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=467672427 TSecr=2087293399 SACK_PERM
3 0.000104 127.0.0.1 127.0.0.1 TCP 56 53055 → 9999 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=2087293399 TSecr=467672427
4 0.000108 127.0.0.1 127.0.0.1 TCP 56 [TCP Window Update] 9999 → 53055 [ACK] Seq=1 Ack=1 Win=408256 Len=0 TSval=467672427 TSecr=2087293399
5 0.032425 127.0.0.1 127.0.0.1 TLSv1… 520 Client Hello
6 0.032466 127.0.0.1 127.0.0.1 TCP 56 9999 → 53055 [ACK] Seq=1 Ack=465 Win=407808 Len=0 TSval=467672459 TSecr=2087293431
7 0.053214 127.0.0.1 127.0.0.1 TLSv1… 183 Server Hello
Handshake protocol message (tls.handshake), 122 bytes
Packets: 79 · Displayed: 79 (100.0%)
Profile: Default

6- Certificates and cryptographic parameters are important components of TLS. The extra overhead and latency involved in repeatedly transmitting the certificate would make it inefficient to send the certificate for every connection. To overcome this issue session resumption feature -which avoids sending the certificate for each connection- is used. After the establishment of secure connection the server generates unique session identifier and sends it to the client. The client includes the session ID that was previously negotiated in its Client Hello message in subsequent connections. After receiving the session data, the server determines whether it has the associated session state saved. The server can quickly reestablish the connection with the previously set cryptographic parameters if it discovers a matching session. As can be seen in the below *Figure 1.* Client Hello message includes session id in it.



Question 2.2

Screenshots for my keystore can be seen in below. The passwords I used is as following **serkan12345.** and **changeit** for -storepass.

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -genkey -alias mykey -keyalg RSA -keypass changeit -keystore keystore.jks -storepass changeit
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
  [Unknown]:  Serkan Berk Bilgic
What is the name of your organizational unit?
  [Unknown]:  Computer Engineering Dept
What is the name of your organization?
  [Unknown]:  Koc University
What is the name of your City or Locality?
  [Unknown]:  Istanbul
What is the name of your State or Province?
  [Unknown]:  Sariyer
What is the two-letter country code for this unit?
  [Unknown]:  TR
Is CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR correct?
  [no]:  yes
```

Fig1

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -list -v -keystore keystore.jks
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: mykey
Creation date: Dec 5, 2023
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Issuer: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Serial number: b40953bd0ba4d7ca
Valid from: Tue Dec 05 15:46:15 TRT 2023 until: Mon Mar 04 15:46:15 TRT 2024
Certificate fingerprints:
         SHA1: 0E:FE:E1:81:74:A3:38:9D:B9:5D:3C:E6:02:36:62:15:9B:8C:9A:75
         SHA256: E0:BE:43:D1:49:CE:E5:B3:B1:8B:AB:8A:67:24:39:DF:7F:B0:49:23:A3:DD:AB:47:2D:BA:AB:07:79:EF:AA:9A
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 3072-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 9C EF 32 D8 2C 9B BB DE   5C 2D 62 10 1A B0 80 8A  ..2.,...\-b.....
0010: 32 F7 F4 A2                                        2...
]
]



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


(base) sberk@Serkan-MacBook-Pro HW3 %
```

Fig2

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -export -alias mykey -file mykey.cer -keystore keystore.jks -storepass changeit
Certificate stored in file <mykey.cer>
(base) sberk@Serkan-MacBook-Pro HW3 % ls
Doc1.docx                                Screenshot 2023-12-05 at 15.47.57.png   mykey.cer
Screenshot 2023-12-05 at 15.46.14.png    keystore.jks                            ~$Doc1.docx
(base) sberk@Serkan-MacBook-Pro HW3 %
```

Fig3

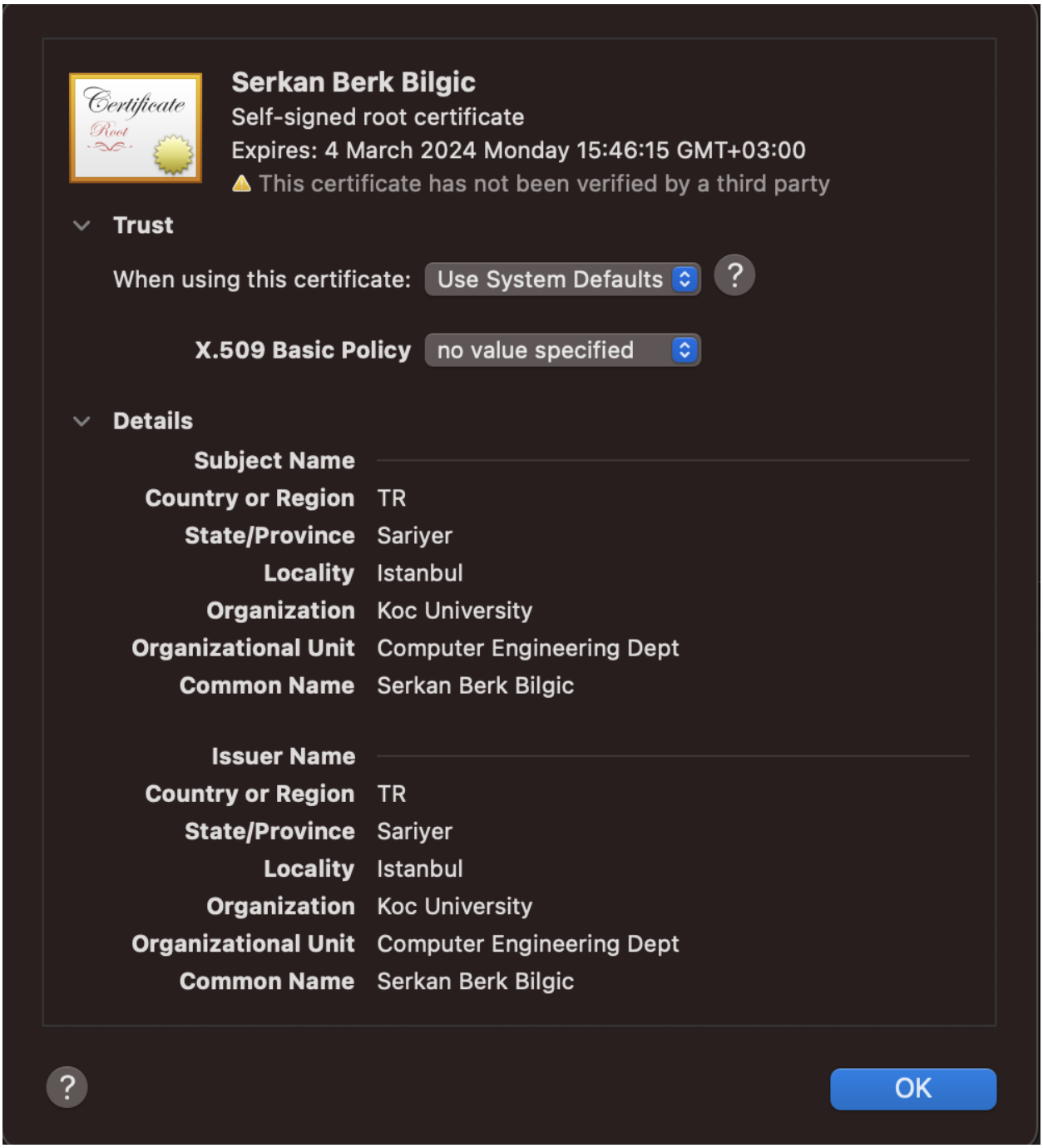


Fig4

Trust Store Part:

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -import -v -trustcacerts -alias mykey -keypass changeit -file mykey.cer -keystore cacerts.jks -storepass changeit
Owner: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Issuer: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Serial number: b40953bd0ba4d7ca
Valid from: Tue Dec 05 15:46:15 TRT 2023 until: Mon Mar 04 15:46:15 TRT 2024
Certificate fingerprints:
         SHA1: 0E:FE:E1:81:74:A3:38:9D:B9:5D:3C:E6:02:36:62:15:9B:8C:9A:75
         SHA256: E0:BE:43:D1:49:CE:E5:B3:B1:8B:AB:8A:67:24:39:DF:7F:B0:49:23:A3:DD:AB:47:2D:BA:AB:07:79:EF:AA:9A
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 3072-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 9C EF 32 D8 2C 9B BB DE   5C 2D 62 10 1A B0 80 8A  ..2.,...\-b.....
0010: 32 F7 F4 A2                                        2...
]
]

Trust this certificate? [no]:  yes
```

Fig1

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -import -v -trustcacerts -alias mykey -keypass changeit -file mykey.cer -keystore cacerts.jks -storepass changeit
Owner: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Issuer: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Serial number: b40953bd0ba4d7ca
Valid from: Tue Dec 05 15:46:15 TRT 2023 until: Mon Mar 04 15:46:15 TRT 2024
Certificate fingerprints:
         SHA1: 0E:FE:E1:81:74:A3:38:9D:B9:5D:3C:E6:02:36:62:15:9B:8C:9A:75
         SHA256: E0:BE:43:D1:49:CE:E5:B3:B1:8B:AB:8A:67:24:39:DF:7F:B0:49:23:A3:DD:AB:47:2D:BA:AB:07:79:EF:AA:9A
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 3072-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 9C EF 32 D8 2C 9B BB DE   5C 2D 62 10 1A B0 80 8A  ..2.,...\-b.....
0010: 32 F7 F4 A2                                        2...
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
[Storing cacerts.jks]
(base) sberk@Serkan-MacBook-Pro HW3 %
```

Fig2

```
(base) sberk@Serkan-MacBook-Pro HW3 % ls
Doc1.docx                                Screenshot 2023-12-05 at 15.47.57.png   Screenshot 2023-12-05 at 15.50.12.png   keystore.jks                            ~$Doc1.docx
Screenshot 2023-12-05 at 15.46.14.png    Screenshot 2023-12-05 at 15.48.58.png   cacerts.jks                             mykey.cer
(base) sberk@Serkan-MacBook-Pro HW3 %
```

Fig3

```
(base) sberk@Serkan-MacBook-Pro HW3 % keytool -list -v -keystore cacerts.jks
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: mykey
Creation date: Dec 5, 2023
Entry type: trustedCertEntry

Owner: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Issuer: CN=Serkan Berk Bilgic, OU=Computer Engineering Dept, O=Koc University, L=Istanbul, ST=Sariyer, C=TR
Serial number: b40953bd0ba4d7ca
Valid from: Tue Dec 05 15:46:15 TRT 2023 until: Mon Mar 04 15:46:15 TRT 2024
Certificate fingerprints:
         SHA1: 0E:FE:E1:81:74:A3:38:9D:B9:5D:3C:E6:02:36:62:15:9B:8C:9A:75
         SHA256: E0:BE:43:D1:49:CE:E5:B3:B1:8B:AB:8A:67:24:39:DF:7F:B0:49:23:A3:DD:AB:47:2D:BA:AB:07:79:EF:AA:9A
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 3072-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 9C EF 32 D8 2C 9B BB DE   5C 2D 62 10 1A B0 80 8A  ..2.,...\-b.....
0010: 32 F7 F4 A2                                        2...
]
]



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


(base) sberk@Serkan-MacBook-Pro HW3 %
```