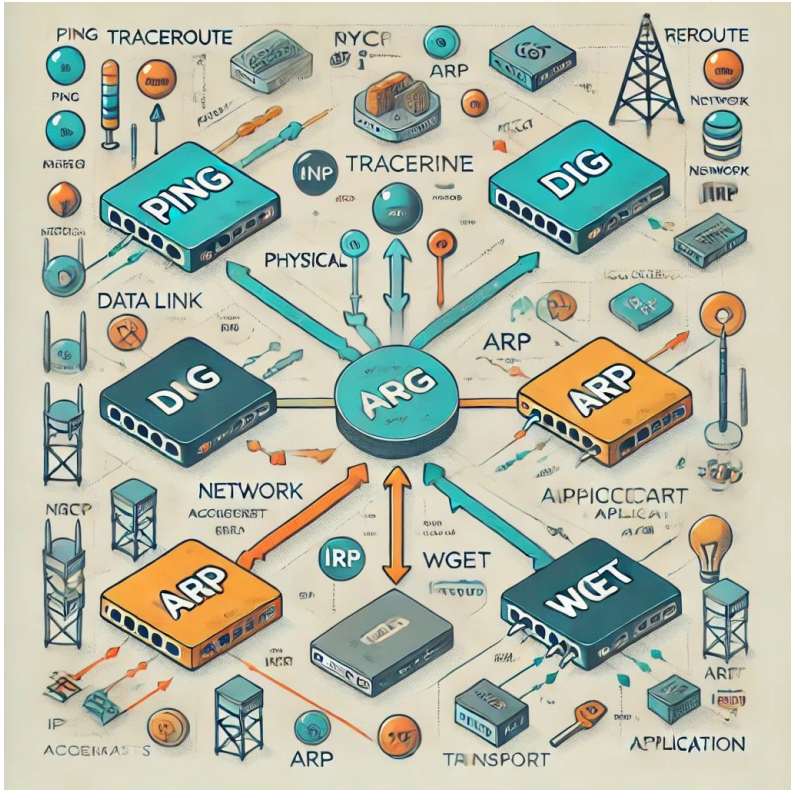
Assignment 2:
Exploring Wireshark Tool
Computer Networks Lab (CS 3272)

Made By: Ramesh Chandra Soren
Enrollment No: 2022CSB086

1. Analyse the packets (across all layers) exchanged with your computer while executing the following commands: (i) `ping`, (ii) `traceroute`, (iii) `dig`, (iv) `arp`, (v) `wget`.

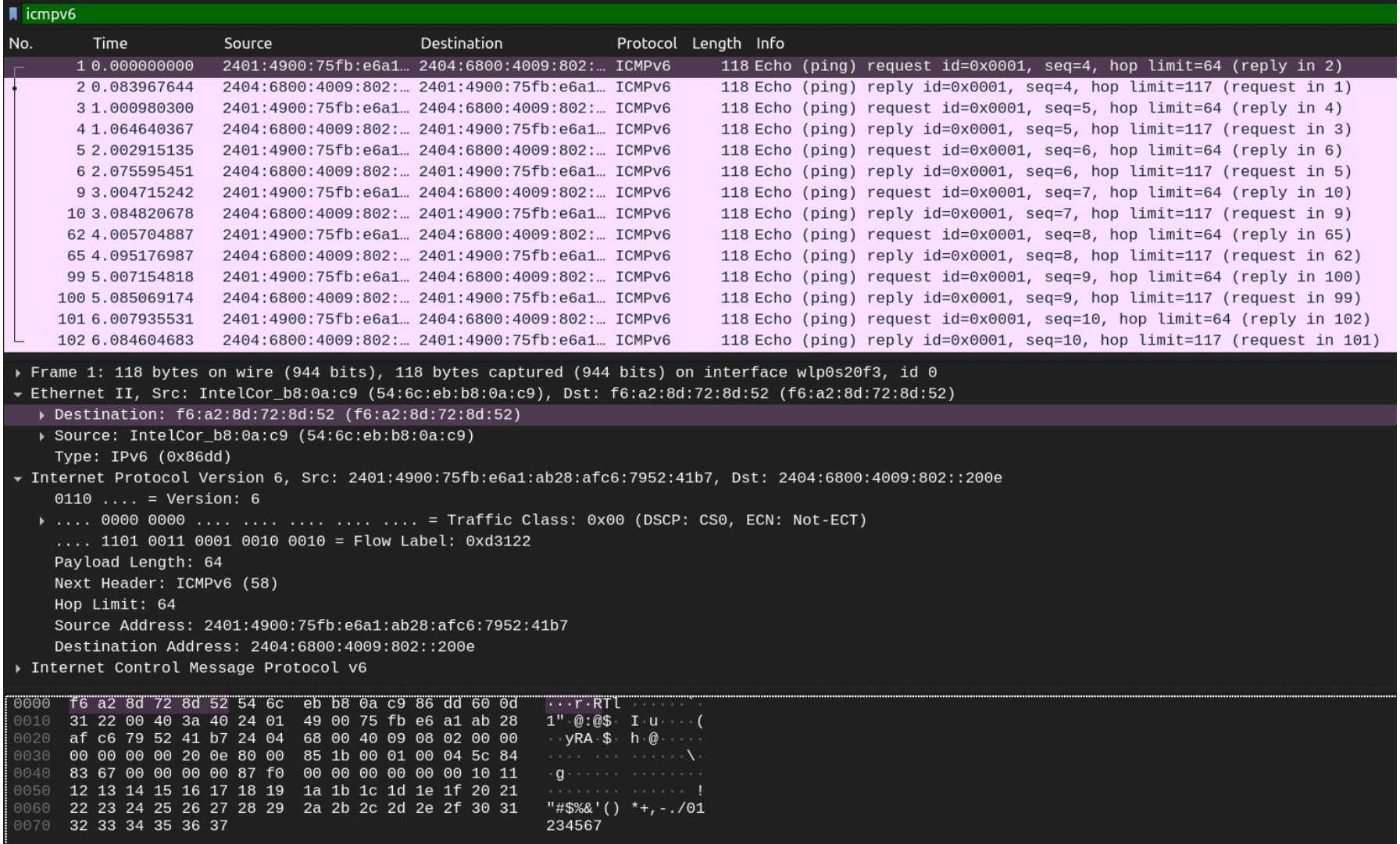


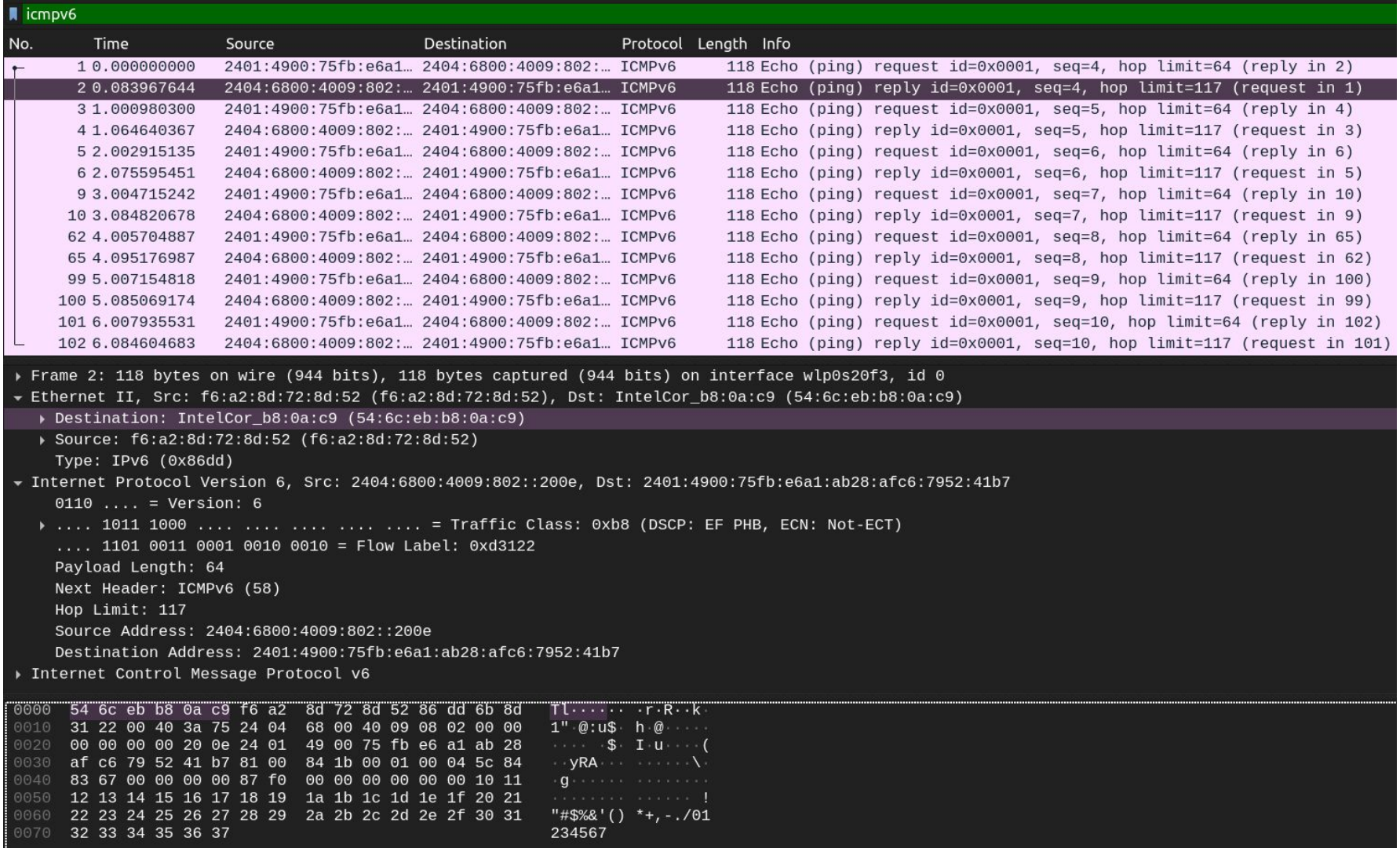
Command	Purpose	Layers Involved	Protocols and Key Packets
`ping`	Tests connectivity and measures round-trip time.	Layer 3 (Network), Layer 2 (Data Link), Layer 1 (Physical)	- ICMP Echo Request (Type 8) - ICMP Echo Reply (Type 0) - Encapsulated in Ethernet frames.
`traceroute`	Traces the path packets take to the destination.	Layers 3, 4, 2, and 1	- ICMP (Time Exceeded, Echo Reply) or UDP packets - TTL field incremented per hop - Encapsulated in IP and Ethernet.
`dig`	Resolves domain names to IP addresses.	Layers 7 (Application), 4 (Transport), 3 (Network), 2 (Data Link)	- DNS Query/Response - UDP (Port 53) or TCP for large queries - Encapsulated in IP and Ethernet frames.
`arp`	Resolves IP addresses to MAC addresses locally.	Layers 3, 2, and 1	- ARP Request: Broadcast (`FF:FF:FF:FF:FF:FF`) - ARP Reply: Unicast with resolved MAC - Works at Layer 2.
`wget`	Downloads files from URLs via HTTP/HTTPS.	Layers 7, 4, 3, 2, and 1	- HTTP GET/HTTPS GET request - TCP (Port 80/443) - IP Packets and Ethernet frames - Three-way handshake in TCP.

What is ICMP?

- ICMP is a network layer protocol used for diagnostics, error reporting, and operational information in IP networks.
- It is defined in **RFC 792**.
- Unlike TCP and UDP, ICMP is not used for data transfer between systems but supports the IP protocol's functionality.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ ping google.com
PING google.com(bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e)) 56 data bytes
64 bytes from pnbomb-aa-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=1 ttl=117 time=150 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=2 ttl=117 time=67.2 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=3 ttl=117 time=150 ms
64 bytes from pnbomb-aa-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=4 ttl=117 time=84.0 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=5 ttl=117 time=63.7 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=6 ttl=117 time=72.7 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=7 ttl=117 time=80.1 ms
64 bytes from pnbomb-aa-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=8 ttl=117 time=89.5 ms
64 bytes from bom05s10-in-x0e.1e100.net (2404:6800:4009:802::200e): icmp_seq=9 ttl=117 time=77.9 ms
^C
--- google.com ping statistics ---
10 packets transmitted, 9 received, 10% packet loss, time 9012ms
rtt min/avg/max/mdev = 63.681/92.783/150.072/31.449 ms
```


icmpv6
No. Time Source Destination Protocol Length Info
1 0.000000000 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=4, hop limit=64 (reply in 2)
2 0.083967644 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=4, hop limit=117 (request in 1)
3 1.000980300 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=5, hop limit=64 (reply in 4)
4 1.064640367 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=5, hop limit=117 (request in 3)
5 2.002915135 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=6, hop limit=64 (reply in 6)
6 2.075595451 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=6, hop limit=117 (request in 5)
9 3.004715242 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=7, hop limit=64 (reply in 10)
10 3.084820678 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=7, hop limit=117 (request in 9)
62 4.005704887 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=8, hop limit=64 (reply in 65)
65 4.095176987 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=8, hop limit=117 (request in 62)
99 5.007154818 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=9, hop limit=64 (reply in 100)
100 5.085069174 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=9, hop limit=117 (request in 99)
101 6.007935531 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=10, hop limit=64 (reply in 102)
102 6.084604683 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=10, hop limit=117 (request in 101)
Frame 1: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9), Dst: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
Destination: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
Source: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
Type: IPv6 (0x86dd)
Internet Protocol Version 6, Src: 2401:4900:75fb:e6a1:ab28:afc6:7952:41b7, Dst: 2404:6800:4009:802::200e
0110 .... = Version: 6
.... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
.... 1101 0011 0001 0010 0010 = Flow Label: 0xd3122
Payload Length: 64
Next Header: ICMPv6 (58)
Hop Limit: 64
Source Address: 2401:4900:75fb:e6a1:ab28:afc6:7952:41b7
Destination Address: 2404:6800:4009:802::200e
Internet Control Message Protocol v6
0000 f6 a2 8d 72 8d 52 54 6c eb b8 0a c9 86 dd 60 0d ···r·RTl ····· ·
0010 31 22 00 40 3a 40 24 01 49 00 75 fb e6 a1 ab 28 1" ·@:@$· I·u ···(
0020 af c6 79 52 41 b7 24 04 68 00 40 09 08 02 00 00 ··yRA ·$· h·@ ····
0030 00 00 00 00 20 0e 80 00 85 1b 00 01 00 04 5c 84 ···· ··· ·····\·
0040 83 67 00 00 00 00 87 f0 00 00 00 00 00 00 10 11 ·g····· ·····
0050 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 ····· ····· !
0060 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0070 32 33 34 35 36 37 234567


icmpv6
No. Time Source Destination Protocol Length Info
1 0.000000000 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=4, hop limit=64 (reply in 2)
2 0.083967644 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=4, hop limit=117 (request in 1)
3 1.000980300 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=5, hop limit=64 (reply in 4)
4 1.064640367 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=5, hop limit=117 (request in 3)
5 2.002915135 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=6, hop limit=64 (reply in 6)
6 2.075595451 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=6, hop limit=117 (request in 5)
9 3.004715242 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=7, hop limit=64 (reply in 10)
10 3.084820678 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=7, hop limit=117 (request in 9)
62 4.005704887 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=8, hop limit=64 (reply in 65)
65 4.095176987 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=8, hop limit=117 (request in 62)
99 5.007154818 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=9, hop limit=64 (reply in 100)
100 5.085069174 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=9, hop limit=117 (request in 99)
101 6.007935531 2401:4900:75fb:e6a1… 2404:6800:4009:802:… ICMPv6 118 Echo (ping) request id=0x0001, seq=10, hop limit=64 (reply in 102)
102 6.084604683 2404:6800:4009:802:… 2401:4900:75fb:e6a1… ICMPv6 118 Echo (ping) reply id=0x0001, seq=10, hop limit=117 (request in 101)
Frame 2: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52), Dst: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
Destination: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
Source: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
Type: IPv6 (0x86dd)
Internet Protocol Version 6, Src: 2404:6800:4009:802::200e, Dst: 2401:4900:75fb:e6a1:ab28:afc6:7952:41b7
0110 .... = Version: 6
.... 1011 1000 .... = Traffic Class: 0xb8 (DSCP: EF PHB, ECN: Not-ECT)
.... 1101 0011 0001 0010 0010 = Flow Label: 0xd3122
Payload Length: 64
Next Header: ICMPv6 (58)
Hop Limit: 117
Source Address: 2404:6800:4009:802::200e
Destination Address: 2401:4900:75fb:e6a1:ab28:afc6:7952:41b7
Internet Control Message Protocol v6
0000 54 6c eb b8 0a c9 f6 a2 8d 72 8d 52 86 dd 6b 8d Tl····· ·r·R··k·
0010 31 22 00 40 3a 75 24 04 68 00 40 09 08 02 00 00 1" @:u$ h @·····
0020 00 00 00 00 20 0e 24 01 49 00 75 fb e6 a1 ab 28 ···· $ I u ···(
0030 af c6 79 52 41 b7 81 00 84 1b 00 01 00 04 5c 84 ··yRA··· ·····\
0040 83 67 00 00 00 00 87 f0 00 00 00 00 00 00 10 11 ·g····· ·····
0050 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 ····· ····· !
0060 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0070 32 33 34 35 36 37 234567

Report: ICMPv6 Packet Capture Analysis

Overview

The provided packet capture shows ICMPv6 (Internet Control Message Protocol for IPv6) traffic. The primary activity includes Echo (ping) requests and replies, which are used for testing connectivity and measuring round-trip time between devices in an IPv6 network.

1. **Packet Details:**
   - Protocol: ICMPv6
   - Packet Type: Echo Request and Echo Reply
   - Payload Length: 64 bytes
3. **Ethernet Frame Details:**
   - Source MAC: `f6:a2:8d:72:8d:52`
   - Destination MAC: `54:6c:eb:b8:0a:c9`
2. **Source and Destination Addresses:**
   - Source Address: `2404:6800:4009:802::200e`
   - Destination Address: `2401:4900:75fb:e6a1:ab28:afc6:7952:41b7`
4. **Traffic Flow:**
   - **Echo Requests:** Generated from `2404:6800:4009:802::200e`.
   - **Echo Replies:** Received back from the destination (`2401:4900:75fb:e6a1:ab28:afc6:7952:41b7`).

5. **Hop Limit:**
    - Echo requests and replies include varying hop limits, which indicate the remaining "lifetime" of the packet as it travels across networks.
6. **Latency Observations:**
    - Packets include round-trip times (e.g., reply times in milliseconds) for each request.

Potential Security Insights

- The ICMPv6 traffic is straightforward; however, excessive ping traffic could indicate a Denial of Service (DoS) attack or reconnaissance activity.
- If this traffic is unexpected, further investigation of the source is recommended.

Note:- The IPv6 addresses suggest global-scope unicast addresses.

ICMP-Based Traceroute Analysis

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ traceroute google.com
traceroute to google.com (142.250.77.110), 30 hops max, 60 byte packets
 1  _gateway (192.168.248.227)  1.912 ms  2.019 ms  2.522 ms
 2  * * *
 3  192.168.182.33 (192.168.182.33)  37.375 ms 192.168.182.65 (192.168.182.65)  79.780 ms 192.168.182.33 (192.168.182.33)  79.779 ms
 4  152.52.62.30 (152.52.62.30)  79.761 ms 152.52.62.26 (152.52.62.26)  79.795 ms  79.782 ms
 5  152.52.62.29 (152.52.62.29)  79.724 ms 152.52.62.25 (152.52.62.25)  79.724 ms 152.52.62.29 (152.52.62.29)  79.703 ms
 6  182.79.177.69 (182.79.177.69)  79.695 ms  76.093 ms *
 7  142.250.169.206 (142.250.169.206)  76.056 ms  76.044 ms  78.249 ms
 8  * * *
 9  142.251.55.224 (142.251.55.224)  78.199 ms 142.251.55.226 (142.251.55.226)  79.370 ms 142.250.233.142 (142.250.233.142)  78.021 ms
10  142.251.55.231 (142.251.55.231)  78.561 ms 142.251.51.118 (142.251.51.118)  92.399 ms 142.251.55.231 (142.251.55.231)  84.843 ms
11  142.251.230.53 (142.251.230.53)  86.021 ms 172.253.70.167 (172.253.70.167)  85.986 ms 142.251.229.251 (142.251.229.251)  83.952 ms
12  142.251.55.231 (142.251.55.231)  82.653 ms maa05s15-in-f14.1e100.net (142.250.77.110)  83.873 ms  83.371 ms
```

Mechanism:

- Sends ICMP Echo Requests with increasing TTL values.
- Each router sends back "Time-to-live exceeded" (ICMP Type 11) messages when TTL reaches zero.

How It Works:

- **Increasing TTL:** Identifies each hop along the path.
- **Routers in Path:** Revealed via the "Source" IP address of the ICMP replies.

No.	Time	Source	Destination	Protocol	Length	Info
15	2.008071643	2401:4900:75fb:e6a1…	2401:4900:75fb:e6a1…	DNS	101	Standard query 0x868d A google.com OPT
16	2.008211544	2401:4900:75fb:e6a1…	2401:4900:75fb:e6a1…	DNS	101	Standard query 0x079a AAAA google.com OPT
17	2.097195322	2401:4900:75fb:e6a1…	2401:4900:75fb:e6a1…	DNS	117	Standard query response 0x868d A google.com A 142.250.77.110 OPT
18	2.097195191	2401:4900:75fb:e6a1…	2401:4900:75fb:e6a1…	DNS	129	Standard query response 0x079a AAAA google.com AAAA 2404:6800:4009:802::200e OPT
19	2.097994320	192.168.248.54	142.250.77.110	UDP	74	40389 → 33434 Len=32
20	2.098021914	192.168.248.54	142.250.77.110	UDP	74	56580 → 33435 Len=32
21	2.098034014	192.168.248.54	142.250.77.110	UDP	74	46699 → 33436 Len=32
22	2.098044909	192.168.248.54	142.250.77.110	UDP	74	47170 → 33437 Len=32
23	2.098056163	192.168.248.54	142.250.77.110	UDP	74	57879 → 33438 Len=32
24	2.098066866	192.168.248.54	142.250.77.110	UDP	74	35973 → 33439 Len=32
25	2.098077874	192.168.248.54	142.250.77.110	UDP	74	33795 → 33440 Len=32
26	2.098088697	192.168.248.54	142.250.77.110	UDP	74	51629 → 33441 Len=32
27	2.098100152	192.168.248.54	142.250.77.110	UDP	74	47749 → 33442 Len=32

```
▸ Frame 19: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9), Dst: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
▾ Internet Protocol Version 4, Src: 192.168.248.54, Dst: 142.250.77.110
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x7c61 (31841)
  ▸ Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▸ Time to Live: 1
    Protocol: UDP (17)
    Header Checksum: 0xa808 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.248.54
    Destination Address: 142.250.77.110
▾ User Datagram Protocol, Src Port: 40389, Dst Port: 33434
    Source Port: 40389
    Destination Port: 33434
    Length: 40
    Checksum: 0x9581 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 3]
```

icmp

No.	Time	Source	Destination	Protocol	Length	Info
35	2.099860934	192.168.248.227	192.168.248.54	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
36	2.100037177	192.168.248.227	192.168.248.54	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
38	2.100552512	192.168.248.227	192.168.248.54	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
39	2.135449391	192.168.182.33	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
40	2.177865493	192.168.182.65	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
41	2.177865686	152.52.62.29	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
42	2.177865732	152.52.62.29	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
43	2.177868050	152.52.62.30	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
44	2.177868450	182.79.177.69	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
45	2.177876023	152.52.62.25	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
46	2.177876214	192.168.182.33	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
47	2.177913365	152.52.62.26	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
48	2.177913479	152.52.62.26	192.168.248.54	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

```
▸ Frame 35: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface wlp0s20f3, id 0
▾ Ethernet II, Src: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52), Dst: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
  ▸ Destination: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
  ▾ Source: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
      Address: f6:a2:8d:72:8d:52 (f6:a2:8d:72:8d:52)
      .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.248.227, Dst: 192.168.248.54
▾ Internet Control Message Protocol
    Type: 11 (Time-to-live exceeded)
    Code: 0 (Time to live exceeded in transit)
    Checksum: 0x8a81 [correct]
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 192.168.248.54, Dst: 142.250.77.110
  ▸ User Datagram Protocol, Src Port: 40389, Dst Port: 33434
▾ Data (32 bytes)
    Data: 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f
    [Length: 32]
```

1. **ICMP Echo Requests and Replies:**
   - **Sequence:** Each packet with incremented TTL corresponds to a router hop.
   - **Destination:** Final target IP remains consistent (e.g., `192.168.248.54`).

2. **Frame 35 Example Analysis:**

User Datagram Protocol (UDP): The encapsulated data from the original ICMP echo request packet that was sent with a TTL of 1 is included. This data is not typically used in ICMP traceroute but might be present due to variations in implementations.

   - **Ethernet II:**
     - Source MAC: Initiating machine (e.g., `f6:a2:8d:72:8d:52`).
     - Destination MAC: First router (e.g., `54:6c:eb:b8:0a:c9`).
   - **IPv4:** Source = Router IP; Destination = Final target IP.
   - **ICMP:** Type 11 (TTL Exceeded), Code 0 (In Transit).

3. **Traceroute Insights:**
   - Identifies each hop using "Time-to-live exceeded" responses.
   - Increasing TTL reveals path to destination.

DNS Query Analysis for `google.com`

Captured Details:

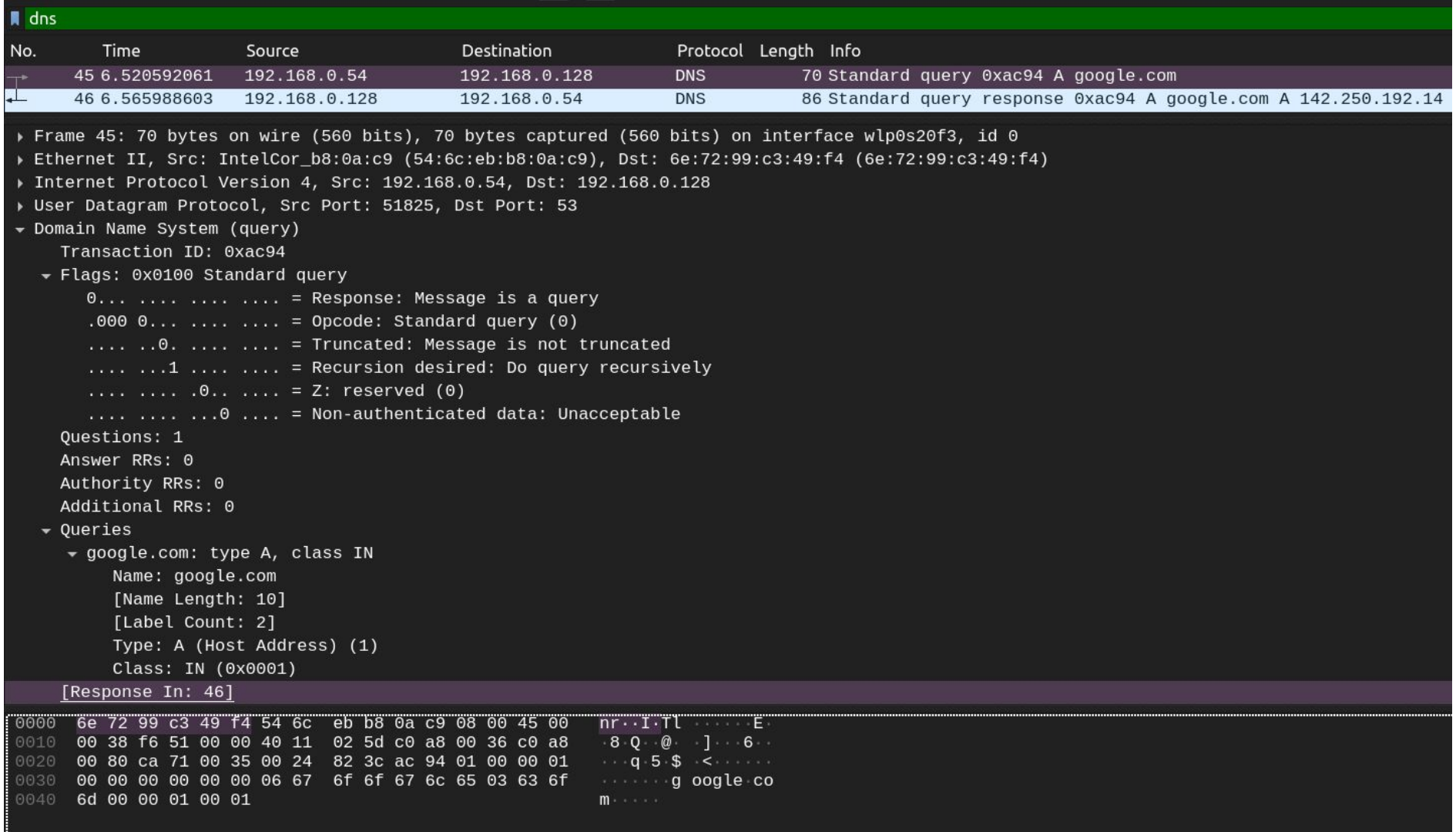
- **Query Details**:
  - **Transaction ID**: 0xac94
  - **Query Name**: `google.com`
  - **Query Type**: A (IPv4 Address)
- **Response Details**:
  - **Transaction ID**: 0xac94
  - **Resolved IP Address**: `142.250.192.14`
  - **Response Time**: 0.045396542 seconds

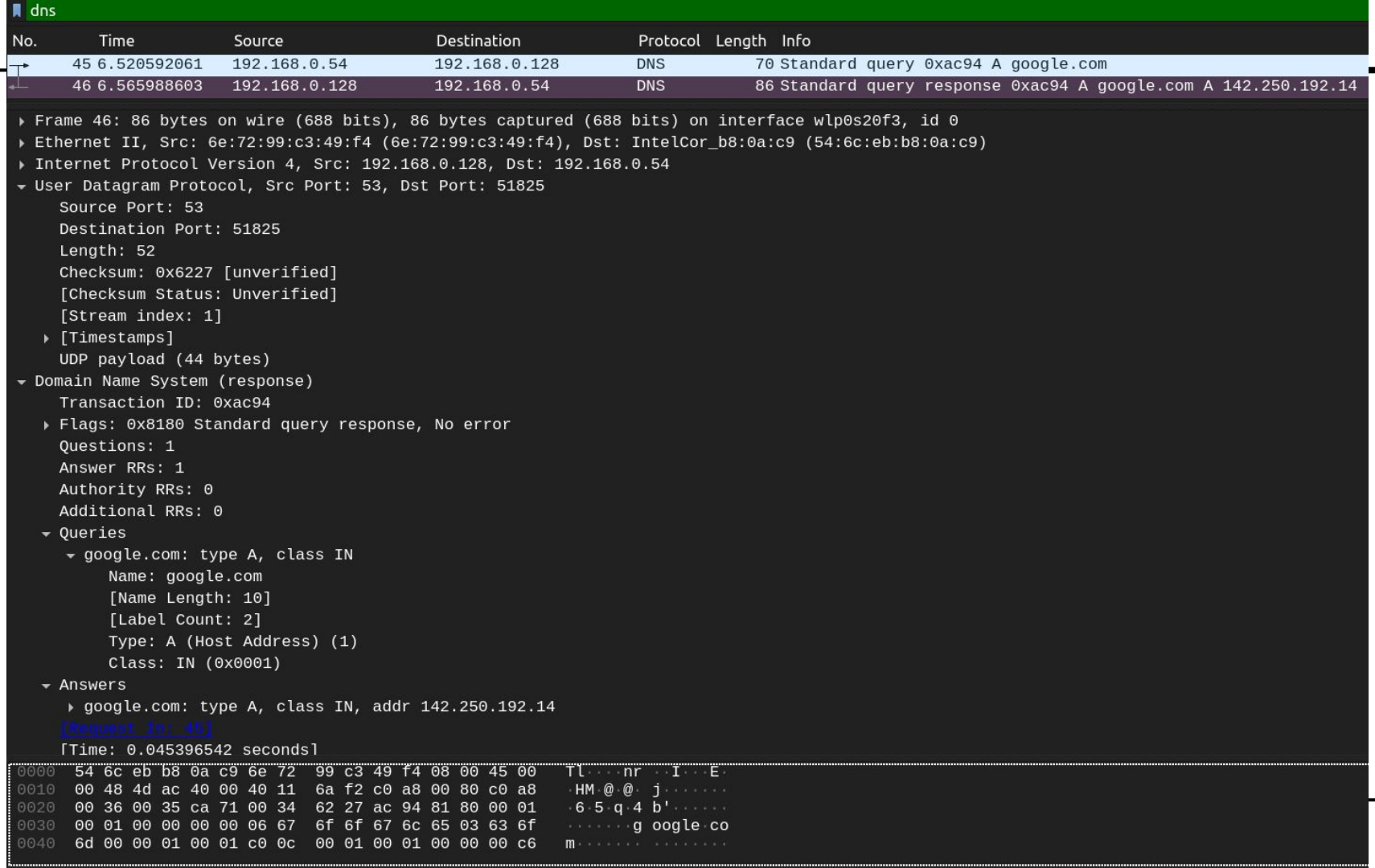
Protocol Overview:

- **Protocol Used**: DNS over UDP
- **Ports**:
  - Source: 53
  - Destination: 51825

Conclusion:

- The DNS query successfully resolved google.com to the IP 142.250.192.14.
- No errors occurred during the transaction.


dns
No. Time Source Destination Protocol Length Info
45 6.520592061 192.168.0.54 192.168.0.128 DNS 70 Standard query 0xac94 A google.com
46 6.565988603 192.168.0.128 192.168.0.54 DNS 86 Standard query response 0xac94 A google.com A 142.250.192.14
Frame 45: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9), Dst: 6e:72:99:c3:49:f4 (6e:72:99:c3:49:f4)
Internet Protocol Version 4, Src: 192.168.0.54, Dst: 192.168.0.128
User Datagram Protocol, Src Port: 51825, Dst Port: 53
Domain Name System (query)
Transaction ID: 0xac94
Flags: 0x0100 Standard query
0... .... = Response: Message is a query
.000 0... .... = Opcode: Standard query (0)
.... ..0. .... = Truncated: Message is not truncated
....1 .... = Recursion desired: Do query recursively
.... .0.. .... = Z: reserved (0)
....0 .... = Non-authenticated data: Unacceptable
Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0
Queries
google.com: type A, class IN
Name: google.com
[Name Length: 10]
[Label Count: 2]
Type: A (Host Address) (1)
Class: IN (0x0001)
[Response In: 46]
0000 6e 72 99 c3 49 f4 54 6c eb b8 0a c9 08 00 45 00
0010 00 38 f6 51 00 00 40 11 02 5d c0 a8 00 36 c0 a8
0020 00 80 ca 71 00 35 00 24 82 3c ac 94 01 00 00 01
0030 00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f
0040 6d 00 00 01 00 01


dns
No. Time Source Destination Protocol Length Info
45 6.520592061 192.168.0.54 192.168.0.128 DNS 70 Standard query 0xac94 A google.com
46 6.565988603 192.168.0.128 192.168.0.54 DNS 86 Standard query response 0xac94 A google.com A 142.250.192.14
Frame 46: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: 6e:72:99:c3:49:f4 (6e:72:99:c3:49:f4), Dst: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
Internet Protocol Version 4, Src: 192.168.0.128, Dst: 192.168.0.54
User Datagram Protocol, Src Port: 53, Dst Port: 51825
Source Port: 53
Destination Port: 51825
Length: 52
Checksum: 0x6227 [unverified]
[Checksum Status: Unverified]
[Stream index: 1]
[Timestamps]
UDP payload (44 bytes)
Domain Name System (response)
Transaction ID: 0xac94
Flags: 0x8180 Standard query response, No error
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
Queries
google.com: type A, class IN
Name: google.com
[Name Length: 10]
[Label Count: 2]
Type: A (Host Address) (1)
Class: IN (0x0001)
Answers
google.com: type A, class IN, addr 142.250.192.14
[Request In: 45]
[Time: 0.045396542 seconds]
0000 54 6c eb b8 0a c9 6e 72 99 c3 49 f4 08 00 45 00 Tl····nr ··I···E·
0010 00 48 4d ac 40 00 40 11 6a f2 c0 a8 00 80 c0 a8 ·HM·@·@· j·····
0020 00 36 00 35 ca 71 00 34 62 27 ac 94 81 80 00 01 ·6·5·q·4 b'····
0030 00 01 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f ·····g oogle·co
0040 6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 c6 m····· ·····

ARP Packet Analysis Report

Objective: To analyze Address Resolution Protocol (ARP) traffic and understand how IP-to-MAC address resolution occurs.

No, you **should not** use `ping google.com` to generate ARP traffic. Here's why:

- **ARP works only within a local network (LAN)**, so it won't resolve `google.com` (a public IP address on the internet). ARP is used to resolve IP addresses to MAC addresses within the same subnet.

Captured ARP Details

1. **ARP Request Packets**:
   - **Source MAC**: `54:6c:eb:b8:0a:c9`
   - **Source IP**: `192.168.0.54`
   - **Target IP**: `192.168.0.1`
   - **Broadcast Destination**: `ff:ff:ff:ff:ff:ff`
   - **Opcode**: Request (1)
     - Message: "Who has `192.168.0.1`? Tell `192.168.0.54`."

Protocol Overview

- **Layer**: Data Link Layer (OSI Layer 2)
- **Protocol Type**: ARP
- **Hardware Type**: Ethernet (1)
- **Protocol Type**: IPv4 (0x0800)

arp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
2	1.024079705	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
5	2.048038051	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
8	3.072085953	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
9	4.096269081	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
18	5.119953282	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
19	6.143998525	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
22	7.168190623	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
27	8.192031713	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54
28	9.215987174	IntelCor_b8:0a:c9	Broadcast	ARP	42	Who has 192.168.0.1? Tell 192.168.0.54

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
    Sender IP address: 192.168.0.54
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.0.1
```

```
0000  ff ff ff ff ff ff 54 6c  eb b8 0a c9 08 06 00 01   ......Tl ........
0010  08 00 06 04 00 01 54 6c  eb b8 0a c9 c0 a8 00 36   ......Tl ....···6
0020  00 00 00 00 00 00 c0 a8  00 01                     ........ ..
```

Observations

- Multiple ARP requests were sent by the source 192.168.0.54 to resolve the MAC address for the target IP 192.168.0.1.
- The ARP request was broadcasted to all devices on the local network (ff:ff:ff:ff:ff:ff).

Conclusion

- ARP successfully initiated a broadcast to resolve the IP address `192.168.0.1` to its MAC address.
- No ARP replies were captured in this screenshot, indicating a potential response might follow or an issue in communication.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
From 192.168.0.54 icmp_seq=1 Destination Host Unreachable
From 192.168.0.54 icmp_seq=2 Destination Host Unreachable
From 192.168.0.54 icmp_seq=3 Destination Host Unreachable
From 192.168.0.54 icmp_seq=4 Destination Host Unreachable
From 192.168.0.54 icmp_seq=5 Destination Host Unreachable
From 192.168.0.54 icmp_seq=6 Destination Host Unreachable
From 192.168.0.54 icmp_seq=7 Destination Host Unreachable
From 192.168.0.54 icmp_seq=8 Destination Host Unreachable
```

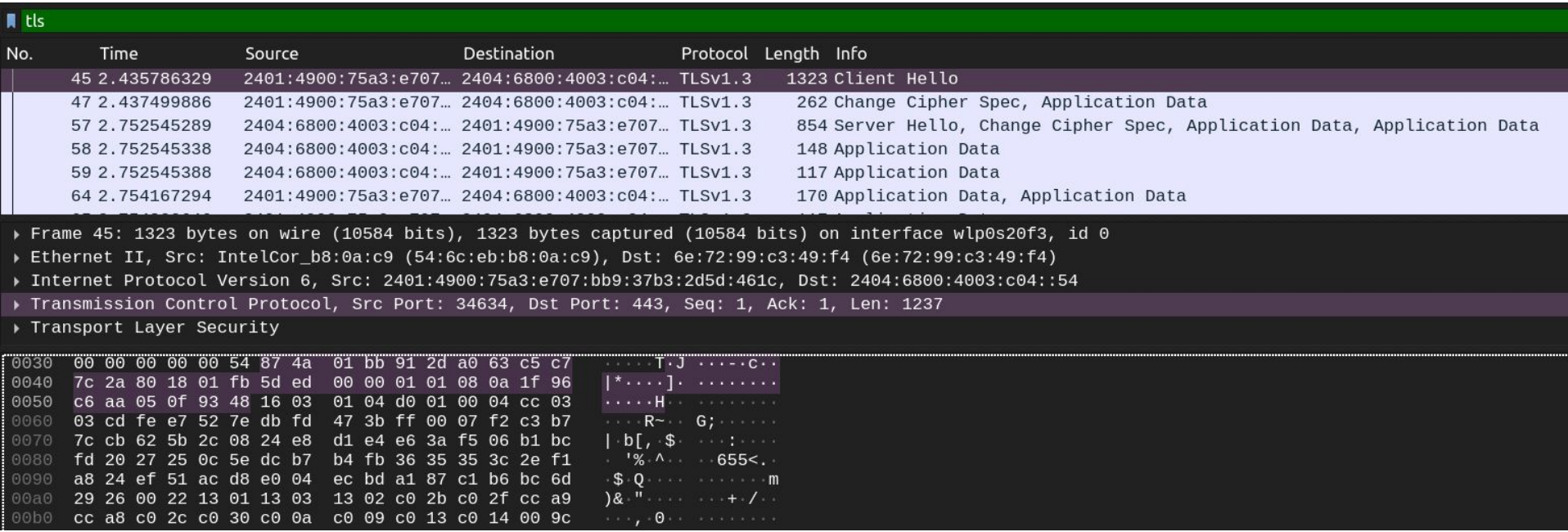
wget is a command-line utility for non-interactive downloading of files from the web using HTTP, HTTPS, and FTP protocols.

```
ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~$ wget https://www.learningcontainer.com/wp-content/uploads/2020/05/sample-mp4-file.mp4
--2025-01-13 10:36:59--  https://www.learningcontainer.com/wp-content/uploads/2020/05/sample-mp4-file.mp4
Resolving www.learningcontainer.com (www.learningcontainer.com)... 2606:4700:3030::6815:5001, 2606:4700:3030::6815:1001, 2606:4700:3030::6815:3001, ...
Connecting to www.learningcontainer.com (www.learningcontainer.com)|2606:4700:3030::6815:5001|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10546620 (10M) [video/mp4]
Saving to: ‘sample-mp4-file.mp4.1’

sample-mp4-file.mp4 100%[===================>]  10.06M  3.64MB/s    in 2.8s

2025-01-13 10:37:03 (3.64 MB/s) - ‘sample-mp4-file.mp4.1’ saved [10546620/10546620]
```

Key Features:

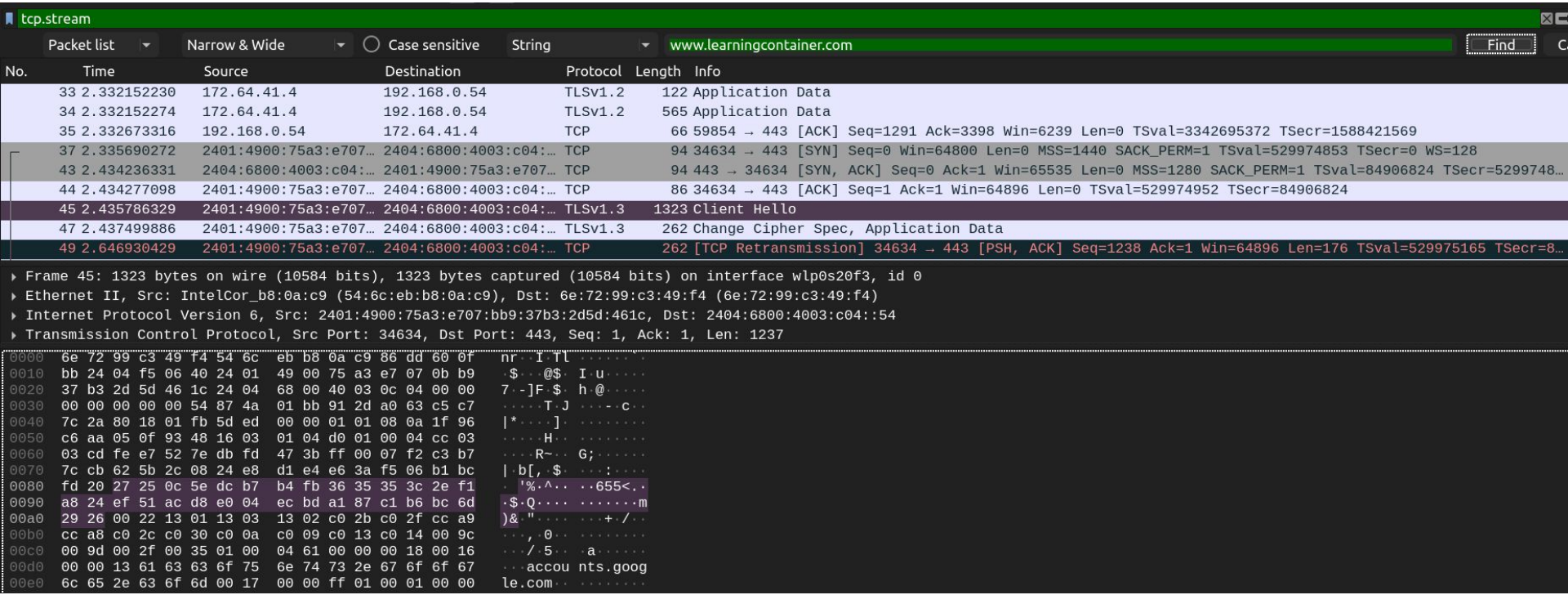
- Supports recursive downloads
- Handles interruptions gracefully (resuming downloads)
- Capable of mirroring websites

wget has successfully connected to the server at the IP address 2606:4700:3030::6815:7001 (which is an IPv6 address).
The server has responded with a 200 OK status and served the file sample-mp4-file.mp4.4.

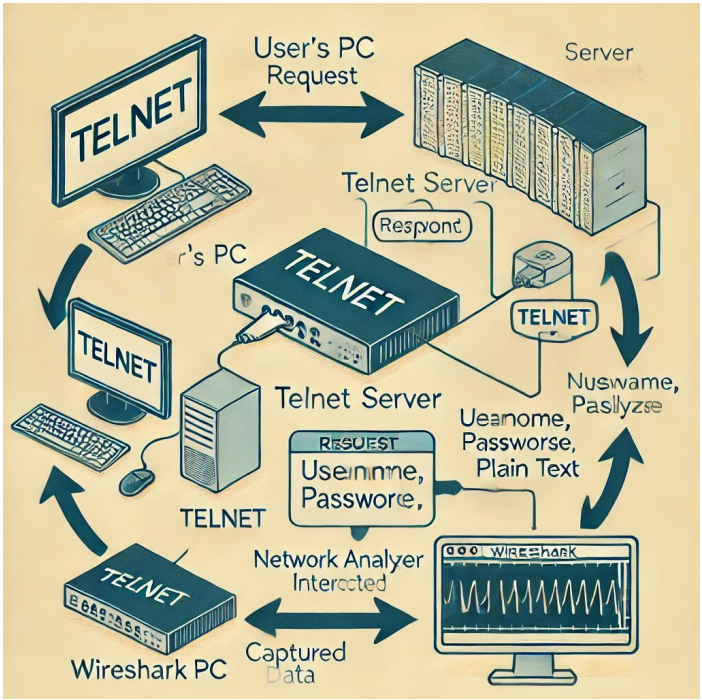


Transport Layer Security (TLS) is a cryptographic protocol designed to provide secure communication over a computer network. It is the successor to the earlier **Secure Sockets Layer (SSL)** protocol, and it ensures privacy, integrity, and authenticity between two communicating parties, such as a web browser and a server.

Since this is using HTTPS we can't directly see the data because it's in encrypted form



2. Capture the packets while sending/receiving `telnet` request/response between your computer and a custom server running the `telnet` daemon. What is your observation while analysing the application layer data?



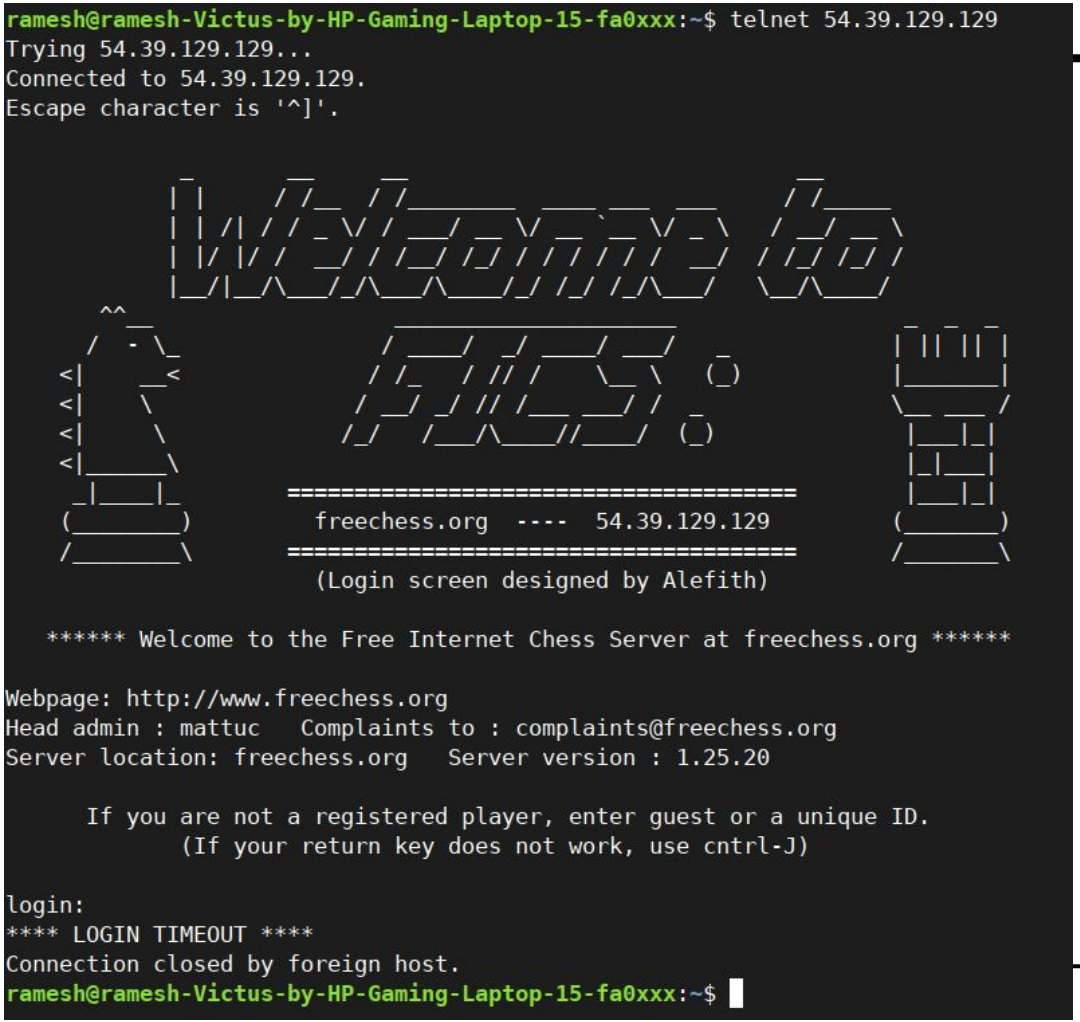
The user's computer connects to the Telnet server, sending a **login request** (username/password).

The server sends a response (e.g., login success or error) and processes user commands.

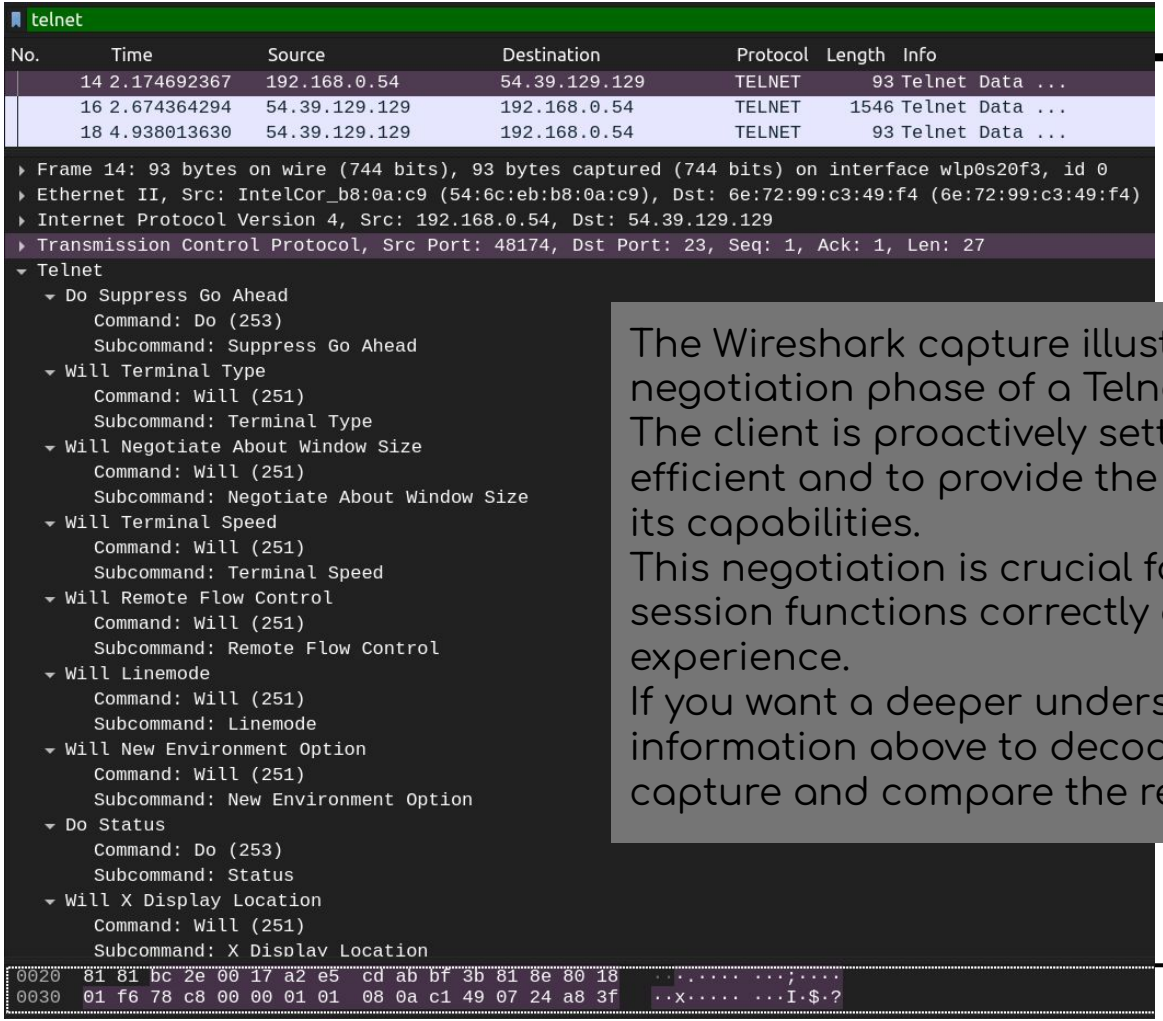
All communication between the user and server (e.g., username, password, commands, responses) is sent as **plain text**.

Anyone monitoring the network can see and steal sensitive details since no encryption is used.

Telnet is insecure for real-world use. Use **SSH**, which encrypts the data, to protect against interception.



We have found a custom telnet server called, ‘freechess.org’ where we have sent the tell new request.
In the picture we can see the Internet Protocol which tell us the Source and Destinations IP addresses.
In this case the destination IP address is, 54.39.129.129.



The Wireshark capture illustrates the initial option negotiation phase of a Telnet connection.
The client is proactively setting up the session to be more efficient and to provide the server with information about its capabilities.
This negotiation is crucial for ensuring that the Telnet session functions correctly and that the user has a good experience.
If you want a deeper understanding, you can use the information above to decode other packets from your capture and compare the results.

The capture shows the initial part of a Telnet connection where the client (192.168.0.54) and the server (54.39.129.129) are negotiating options.

1. **Initial Telnet Packets**

    - Packet 14: This packet, from the client to the server, initiates option negotiation, which will tell the server how to handle various aspects of the Telnet session.
    - Packets 16 and 18: appear to be ongoing data transfer or further negotiation, possibly responses to earlier requests or commands.

2. Telnet Option Negotiation (Packet 14)

- **Do Suppress Go Ahead (Option 3)**: Client requests to suppress "Go Ahead" signals to improve efficiency.
- **Will Terminal Type (Option 24)**: Client agrees to send its terminal type for server customization.
- **Will NAWS (Option 31)**: Client will negotiate window size for proper formatting.
- **Will Terminal Speed (Option 32)**: Client will report terminal speed for output adjustments.
- **Will Remote Flow Control (Option 33)**: Client supports server-initiated flow control.
- **Will Linemode (Option 34)**: Client buffers and sends input line-by-line for efficiency.
- **Will New Environment (Option 39)**: Client will pass environment variables to the server.
- **Do Status (Option 5)**: Client requests server status information.
- **Will X Display Location (Option 35)**: Client will provide X display location for graphical output redirection.

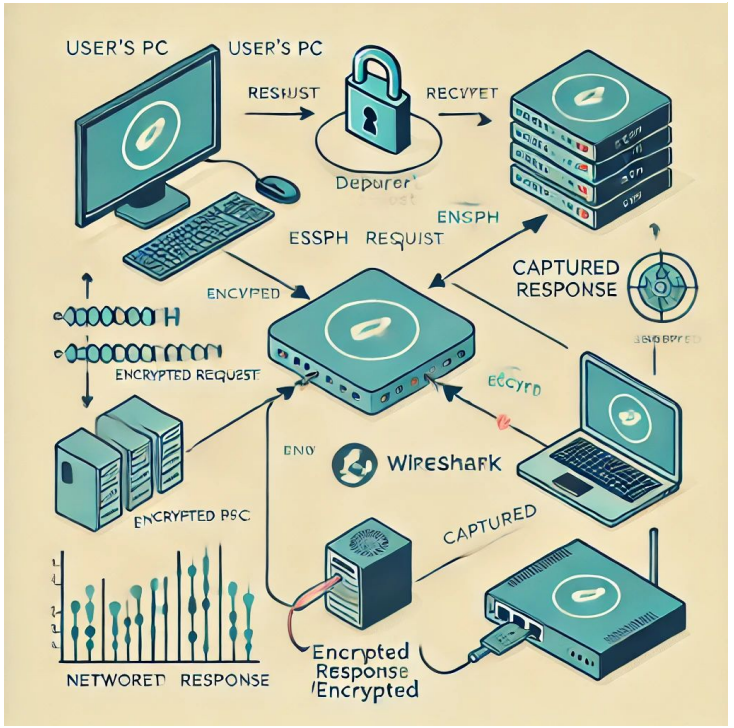
3. Capture the packets while sending/receiving `ssh`request/response between your computer and one of the department servers. What is your observation while analysing the application layer data?

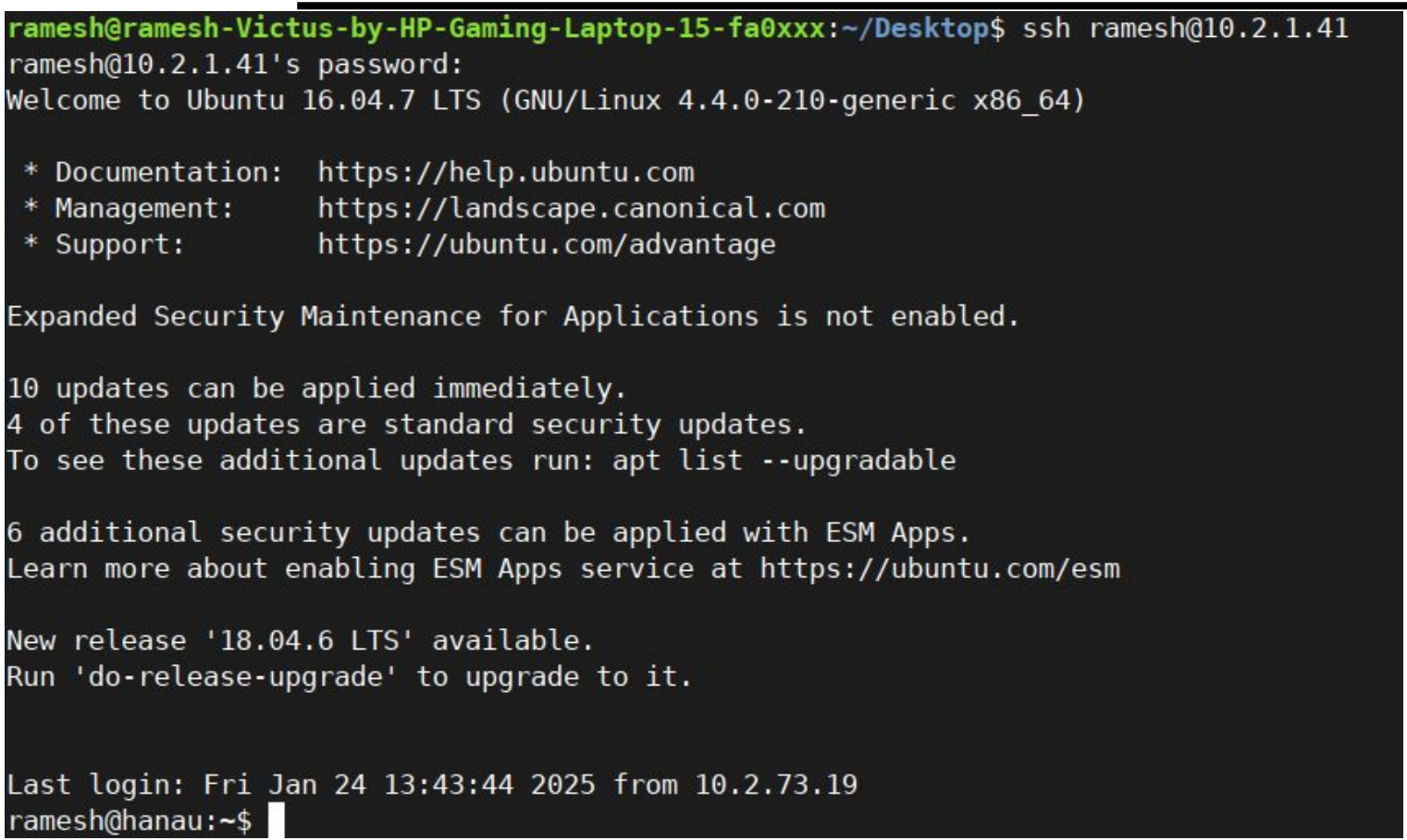
The user's computer initiates a secure connection to the department server.

The department server processes the request and sends encrypted data back.

A tool like Wireshark captures the traffic but shows only **encrypted data**, making it impossible to view sensitive details.

Unlike Telnet, **SSH protects sensitive information** (e.g., credentials and commands) by encrypting the data, ensuring privacy and security.




ramesh@ramesh-Victus-by-HP-Gaming-Laptop-15-fa0xxx:~/Desktop$ ssh ramesh@10.2.1.41
ramesh@10.2.1.41's password:
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.4.0-210-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
Expanded Security Maintenance for Applications is not enabled.
10 updates can be applied immediately.
4 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable
6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm
New release '18.04.6 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
Last login: Fri Jan 24 13:43:44 2025 from 10.2.73.19
ramesh@hanau:~$

ssh

No.	Time	Source	Destination	Protocol	Length	Info
182	3.172765906	10.2.73.19	10.2.1.41	SSHv2	107	Client: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6)
184	3.201571702	10.2.1.41	10.2.73.19	SSHv2	108	Server: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10)
186	3.202530272	10.2.73.19	10.2.1.41	SSHv2	1602	Client: Key Exchange Init
187	3.212451473	10.2.1.41	10.2.73.19	SSHv2	1042	Server: Key Exchange Init
189	3.220242481	10.2.73.19	10.2.1.41	SSHv2	114	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
208	3.285815894	10.2.1.41	10.2.73.19	SSHv2	358	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (len=84)
210	3.294511809	10.2.73.19	10.2.1.41	SSHv2	82	Client: New Keys
239	3.693641782	10.2.73.19	10.2.1.41	SSHv2	110	Client: Encrypted packet (len=44)
242	3.743115345	10.2.1.41	10.2.73.19	SSHv2	110	Server: Encrypted packet (len=44)
243	3.743270388	10.2.73.19	10.2.1.41	SSHv2	134	Client: Encrypted packet (len=68)
244	3.755043274	10.2.1.41	10.2.73.19	SSHv2	118	Server: Encrypted packet (len=52)
329	6.550760326	10.2.73.19	10.2.1.41	SSHv2	214	Client: Encrypted packet (len=148)
341	6.793042124	10.2.1.41	10.2.73.19	SSHv2	96	Server: Encrypted packet (len=28)
343	6.793229313	10.2.73.19	10.2.1.41	SSHv2	178	Client: Encrypted packet (len=112)
994	32.211041170	10.2.1.41	10.2.73.19	SSHv2	1006	Server: Encrypted packet (len=940)
995	32.212212618	10.2.73.19	10.2.1.41	SSHv2	414	Client: Encrypted packet (len=348)
996	32.228560993	10.2.1.41	10.2.73.19	SSHv2	110	Server: Encrypted packet (len=44)
997	32.228836834	10.2.73.19	10.2.1.41	SSHv2	518	Client: Encrypted packet (len=452)
998	32.239513263	10.2.1.41	10.2.73.19	SSHv2	374	Server: Encrypted packet (len=308)
999	32.244338689	10.2.1.41	10.2.73.19	SSHv2	174	Server: Encrypted packet (len=108)
1003	32.253619324	10.2.1.41	10.2.73.19	SSHv2	814	Server: Encrypted packet (len=748)
1005	32.328706134	10.2.1.41	10.2.73.19	SSHv2	118	Server: Encrypted packet (len=52)

Protocol Initialization: The SSH handshake begins with both client and server identifying their protocol versions (e.g., SSH-2.0).

Key Exchange: Secure algorithms (like Elliptic Curve Diffie-Hellman) are used to establish encryption keys.

Encrypted Communication: Once established, communication is encrypted, indicated by "Encrypted packet" in captures.

```
▸ Frame 182: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface wlp0s20f3, id 0
▾ Ethernet II, Src: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9), Dst: HewlettP_8c:9e:8c (9c:b6:54:8c:9e:8c)
  ▸ Destination: HewlettP_8c:9e:8c (9c:b6:54:8c:9e:8c)
  ▸ Source: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 10.2.73.19, Dst: 10.2.1.41
▸ Transmission Control Protocol, Src Port: 33202, Dst Port: 22, Seq: 1, Ack: 1, Len: 41
▾ SSH Protocol
    Protocol: SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6
    [Direction: client-to-server]
```

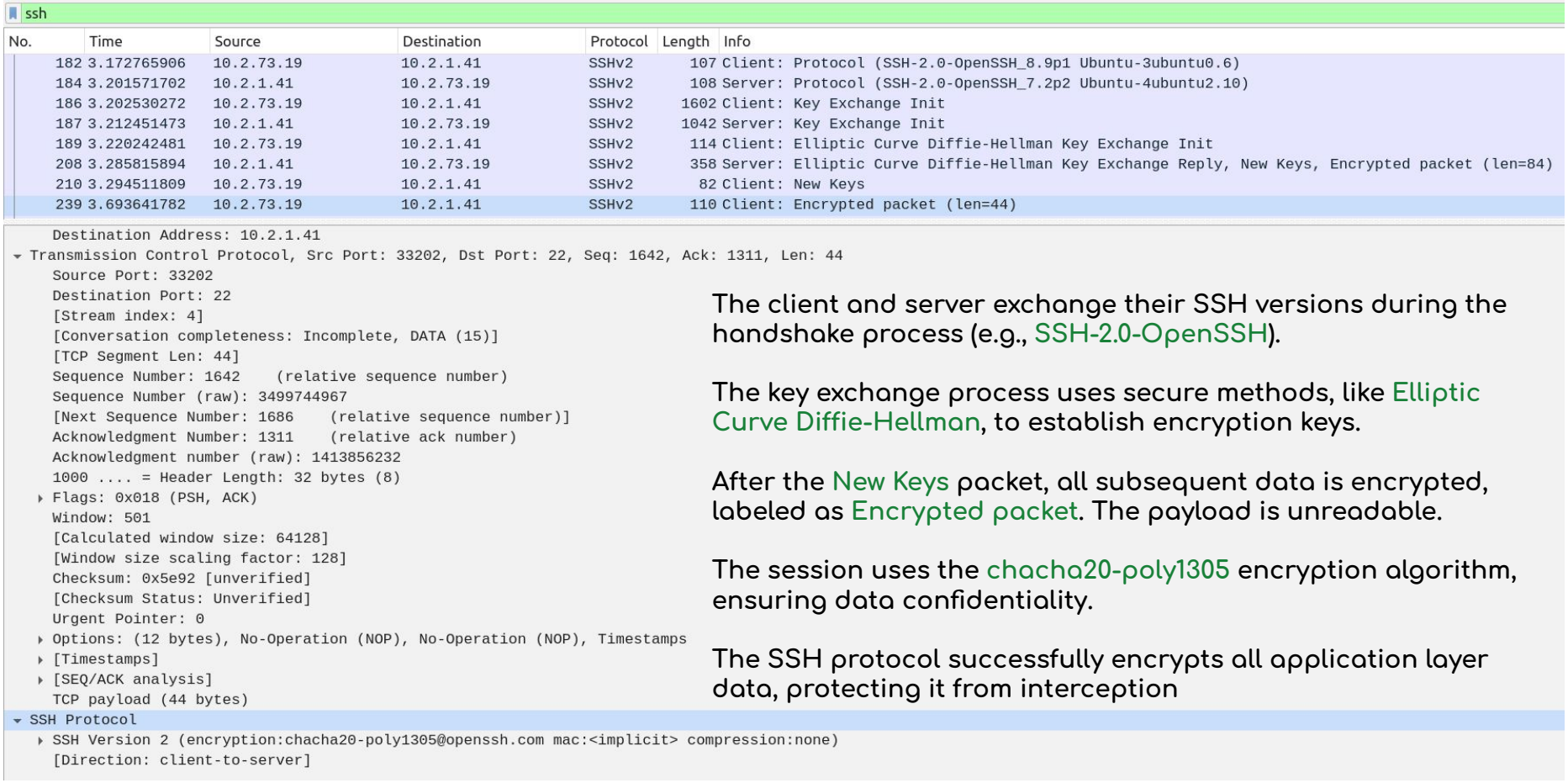
Connection Flow: Source/destination IP addresses and TCP port 22 (default for SSH) reveal the direction of communication between client and server.

Security: Application layer data is unreadable due to encryption, a core security feature of SSH.

```
0000  9c b6 54 8c 9e 8c 54 6c  eb b8 0a c9 08 00 45 10   ··T···Tl ····· E·
0010  00 5d 79 94 40 00 40 06  62 b7 0a 02 49 13 0a 02   ·]y·@·@· b···I···
0020  01 29 81 b2 00 16 d0 99  d8 5e 54 45 b6 ca 80 18   ·)····· ·^TE····
0030  01 f6 5e 8f 00 00 01 01  08 0a ab 3c 74 7b 83 8c   ··^····· ···<t{··
0040  9d 8e 53 53 48 2d 32 2e  30 2d 4f 70 65 6e 53 53   ··SSH-2. 0-OpenSS
```

SSH Protocol: Protocol

Packets: 1119 · Displayed: 22 (2.0%) · Dropped: 0 (0.0%)



The client and server exchange their SSH versions during the handshake process (e.g., SSH-2.0-OpenSSH).

The key exchange process uses secure methods, like Elliptic Curve Diffie-Hellman, to establish encryption keys.

After the New Keys packet, all subsequent data is encrypted, labeled as Encrypted packet. The payload is unreadable.

The session uses the chacha20-poly1305 encryption algorithm, ensuring data confidentiality.

The SSH protocol successfully encrypts all application layer data, protecting it from interception

4. Enter the URL:`http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html` andcapture packets using `Wireshark`. After your browser has displayed the `INTRO-wireshark-file1.html` page (it is a simple one line of congratulations), stop `Wireshark` packet capture.

   Answer the following from the capturedpackets:

   a. How long did it take from when the `HTTP GET` message was sent until the `HTTP OK` reply was received?
   b. What is the Internet address of the `gaia.cs.umass.edu`? What is the Internet address of your computer? Support your answer with an appropriate screenshot from your computer.

Your browser sends an HTTP GET request to gaia.cs.umass.edu to fetch the HTML page.

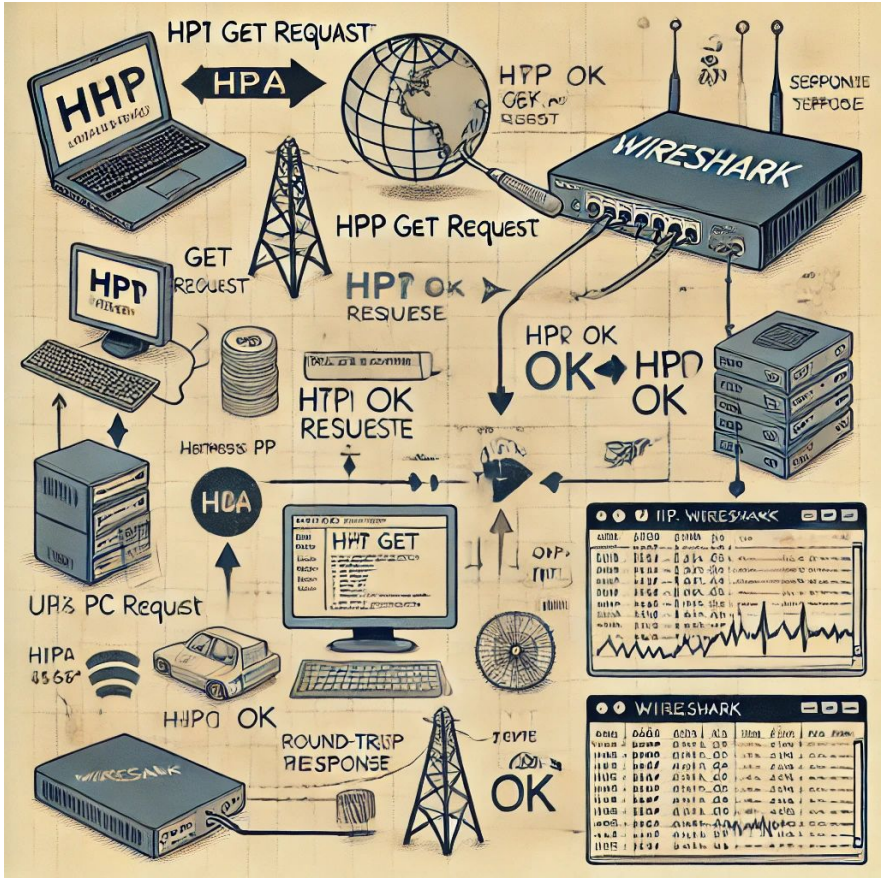
The server responds with an HTTP 200 OK message and the requested content.

Wireshark logs both request and response packets, displaying:

- **Timestamps**: Measure the time between the request and response.
- **Source IP**: The IP address of your computer.
- **Destination IP**: The IP address of gaia.cs.umass.edu.

Use Wireshark interface to determine:

- **Time Taken**: Difference between the GET request and OK reply timestamps.
- **IP Details**: Locate source and destination IP addresses.



http

No.	Time	Source	Destination	Protocol	Length	Info
121	5.757349038	192.168.0.54	128.119.245.12	HTTP	436	GET /wireshark-labs/INTRO-wireshark-file1.html HTTP/1.1
131	6.069433149	128.119.245.12	192.168.0.54	HTTP	492	HTTP/1.1 200 OK (text/html)
151	6.342975581	192.168.0.54	128.119.245.12	HTTP	393	GET /favicon.ico HTTP/1.1
164	6.723602439	128.119.245.12	192.168.0.54	HTTP	538	HTTP/1.1 404 Not Found (text/html)

```
▸ Frame 131: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits) on interface wlp0s20f3, id 0
▸ Ethernet II, Src: 6e:72:99:c3:49:f4 (6e:72:99:c3:49:f4), Dst: IntelCor_b8:0a:c9 (54:6c:eb:b8:0a:c9)
▸ Internet Protocol Version 4, Src: 128.119.245.12, Dst: 192.168.0.54
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 38080, Seq: 1, Ack: 383, Len: 438
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 200 OK\r\n
    ▸ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    Date: Thu, 16 Jan 2025 04:29:20 GMT\r\n
    Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.33 mod_perl/2.0.11 Perl/v5.16.3\r\n
    Last-Modified: Wed, 15 Jan 2025 06:59:01 GMT\r\n
    ETag: "51-62bb9366073a0"\r\n
    Accept-Ranges: bytes\r\n
  ▸ Content-Length: 81\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.312084111 seconds]
    [Request in frame: 121]
    [Next request in frame: 151]
    [Next response in frame: 164]
    [Request URI: http://gaia.cs.umass.edu/favicon.ico]
    File Data: 81 bytes
▾ Line-based text data: text/html (3 lines)
    <html>\n
    Congratulations!  You've downloaded the first Wireshark lab file!\n
    </html>\n
```

In summary, the Wireshark capture indicates that it took about 0.312 seconds for the HTTP OK message to be received after the GET message was sent. gaia.cs.umass.edu has an IP address of 128.119.245.12, and your computer's IP address is 192.168.0.54.

a. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received?

Based on the “Time” column in the Wireshark capture:

- HTTP GET message (packet #121) was sent at **5.757349038** seconds.
- HTTP OK reply (packet #131) was received at **6.069433149** seconds.

Therefore, the time difference is approximately **0.312084111** seconds (6.069433149 - 5.757349038 = 0.312084111). This is confirmed in the analysis under “[Time since request: 0.312084111 seconds]”.

b. What is the Internet address of the gaia.cs.umass.edu? What is the Internet address of your computer? Support your answer with an appropriate screenshot from your computer.

- **gaia.cs.umass.edu's Internet Address:** The capture shows that the destination IP address for the HTTP GET requests is **128.119.245.12**. This is the Internet address of gaia.cs.umass.edu.

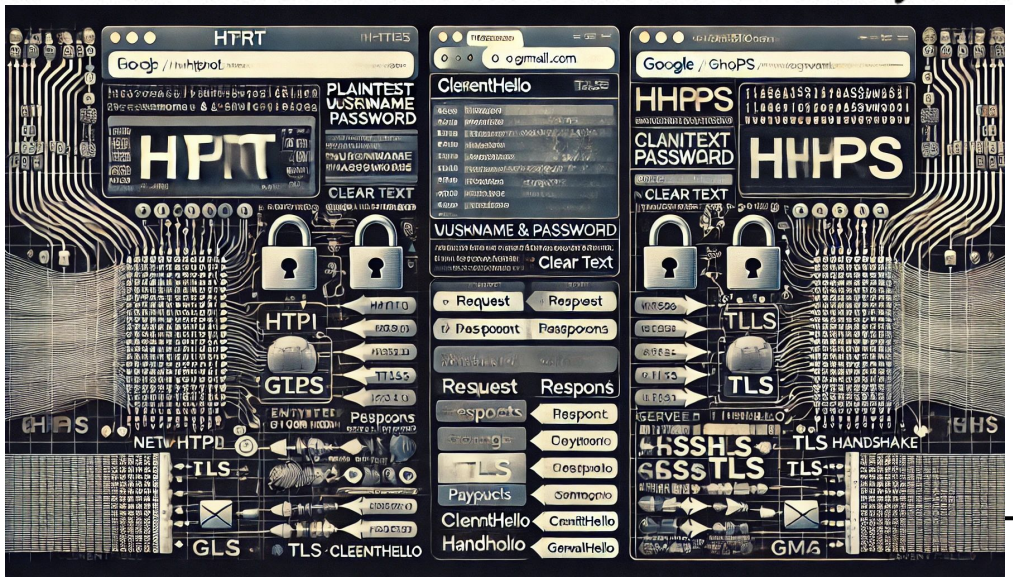
- **Your Computer's Internet Address:** The source IP address for the HTTP GET requests is listed as **192.168.0.54**. This is your computer's internet address.

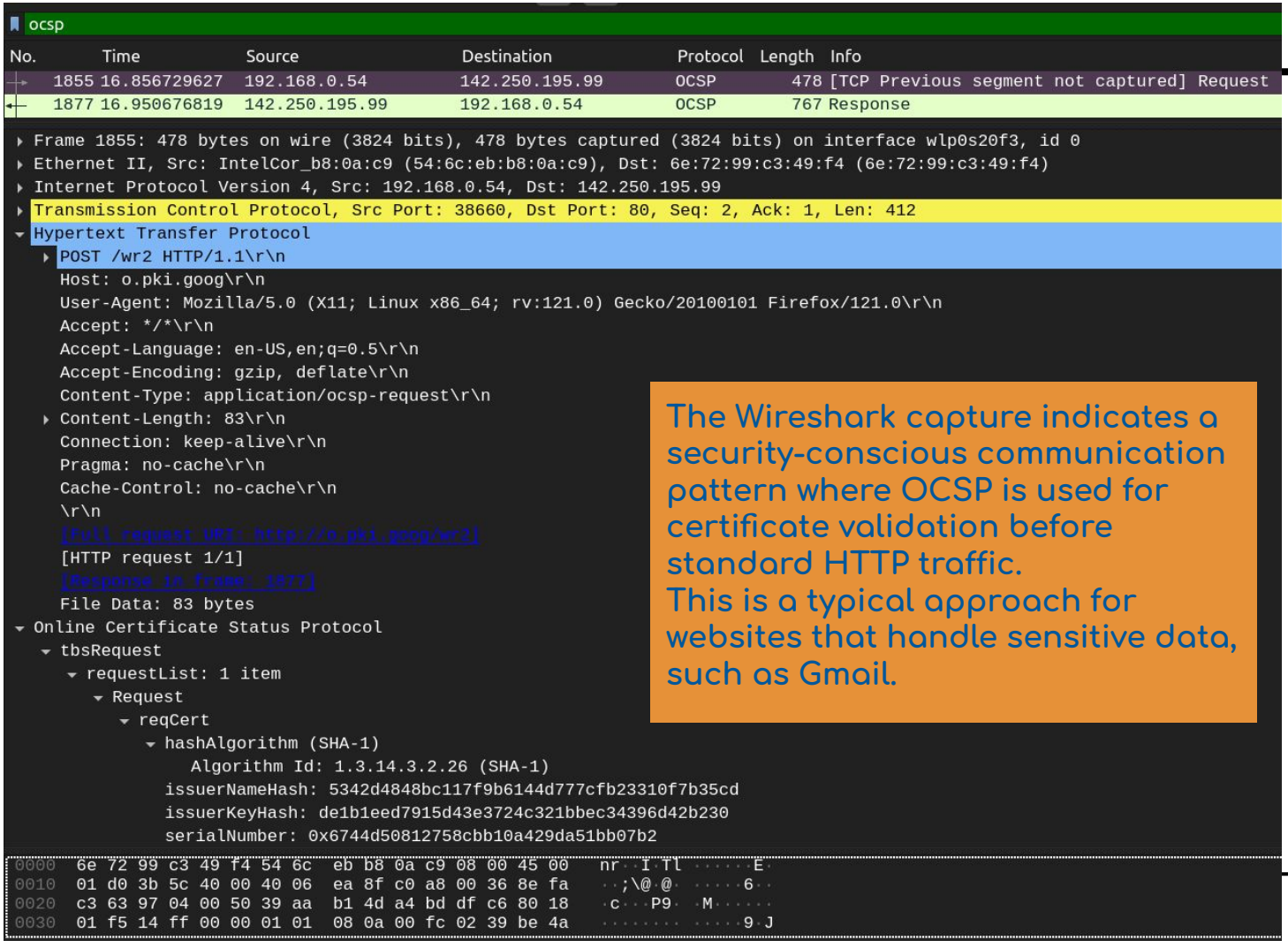
5. Start the `Wireshark` packet capturing service. Enter the URL: `https://www.gmail.com` on your browser and sign-in to your gmail account by providing credentials (Username/Password).

Answer the following from the captured packets:

a. Is there any difference in the application layer protocol?

b. How it is different from the `HTTP` data you analysed in the above problem?





The Wireshark capture indicates a security-conscious communication pattern where OCSP is used for certificate validation before standard HTTP traffic.
This is a typical approach for websites that handle sensitive data, such as Gmail.

Okay, here's an analysis of the Wireshark capture and answers to your questions about the application layer protocol, based on the provided image.

The Wireshark capture shows a communication sequence that includes both OCSP (Online Certificate Status Protocol) and HTTP traffic. Let's break it down:

1. **OCSP Traffic:**
    - **Purpose:** OCSP is used to check the revocation status of digital certificates. It's a security measure to ensure that a website's certificate is still valid and hasn't been compromised.
    - **Observations:**
        - Packets 1855 and 1877 show an OCSP request and response.
        - The request is going to `o.pki.goog`, which is a Google-operated OCSP responder.
        - The request includes details like the issuer's name, issuer key hash, and the serial number of the certificate being checked.

2. **HTTP Traffic:**

- **Purpose:** This is the standard protocol for web communication, used to transmit data between a web client (browser) and a web server.
- **Observations:**
  - The highlighted packet is an HTTP POST request to `/wr2`.
  - The "Host" header indicates it's also going to a Google domain (`o.pki.goog`).
  - The User-Agent header reveals that the client is Firefox 121.0 on Linux.
  - There are typical HTTP headers like Accept, Accept-Language, Accept-Encoding, Content-Type, etc.

Answers to Your Questions

A. Is there any difference in the application layer protocol?

Yes, there is a difference. The capture shows the use of two application layer protocols:

- **OCSP (Online Certificate Status Protocol):** This is seen in the initial exchange where the client verifies a certificate's status.
- **HTTP (Hypertext Transfer Protocol):** This is observed in the later part of the capture for the actual web request.

B. How is it different from the HTTP data you analyzed in the above problem?

Here's how it's different from a typical HTTP exchange.

1. **OCSP Precedes HTTP:** In this case, OCSP traffic is used *before* the primary HTTP communication. The browser is performing a certificate status check before sending the HTTP request. This is a security practice to make sure that the website the user is connecting to is legitimate and its certificate is valid.
2. **Security Focus:** The presence of OCSP indicates that the browser (and the user) is prioritizing security. OCSP is not always used in every HTTP interaction. It's typically employed in situations where enhanced security is required, for example, when dealing with sensitive data, such as during a Gmail login.
3. **Destination:** Although both are to Google servers, the OCSP and HTTP traffic are directed to different resources (likely different servers or different services within Google's infrastructure). OCSP goes to a specialized service that provides certificate revocation information, while the HTTP goes to a web server.
4. **Content:** The OCSP request has content specifically formatted for certificate status verification. The HTTP request contains data related to the web request (`/wr2` in this case). It would also likely contain cookies and other session-related information after the initial connection.

In the Context of Gmail Login

The sequence you've described (visiting `https://www.gmail.com` and logging in) explains the observed traffic:

1. **Initial Connection:** When you connect to `https://www.gmail.com`, your browser needs to establish a secure (HTTPS) connection.
2. **Certificate Verification:** Before the browser trusts the connection, it performs an OCSP check to ensure that Gmail's certificate is valid. This is seen in packets 1855 and 1877.
3. **Secure Communication:** After the certificate is verified, the browser proceeds with the HTTP communication, which includes sending your login credentials in a secure manner (likely encrypted within the HTTPS tunnel).