Assignment 1 Report
Computer Networks

Members:
Soham Shrivastava (23110315)
Arin Mehta (23110038)

Github Repository: https://github.com/SohamShrivastava/CN_Assignment_1

Task 1

In this task, we implemented the DNS-Client server connection where the client reads DNS server query from reading pcap file (we used 3.pcap), extracts the domain name and attaches a custom header which contains the current timestamp and query ID in the format of "HHMMSSID". This combined packet is then sent over a TCP connection to the server.

On the server side, the packet is received and parsed. It separates the custom header from the DNS query and then applies certain rules to generate the response IP from the IP pool of 15 IP addresses.
Then this response IP is sent back to the client. And then the client receives this response IP and stores the result in csv file.

```
(env) (base) sohamshrivastava@Sohams-MacBook-Air-10 CN_Assignment_1 % python3 server.py
server listening on 127.0.0.1:12345
connected by ('127.0.0.1', 58808)
received query for: netflix.com., with header: 23112700
resolved ip: 192.168.1.11
response sent back to client

connected by ('127.0.0.1', 58809)
received query for: linkedin.com., with header: 23112801
resolved ip: 192.168.1.12
response sent back to client

connected by ('127.0.0.1', 58810)
received query for: example.com., with header: 23112902
resolved ip: 192.168.1.13
response sent back to client

connected by ('127.0.0.1', 58811)
received query for: google.com., with header: 23112903
resolved ip: 192.168.1.14
response sent back to client

connected by ('127.0.0.1', 58812)
received query for: facebook.com., with header: 23113004
resolved ip: 192.168.1.15
response sent back to client

connected by ('127.0.0.1', 58813)
received query for: amazon.com., with header: 23113005
resolved ip: 192.168.1.11
response sent back to client
```

Fig : output from server.py

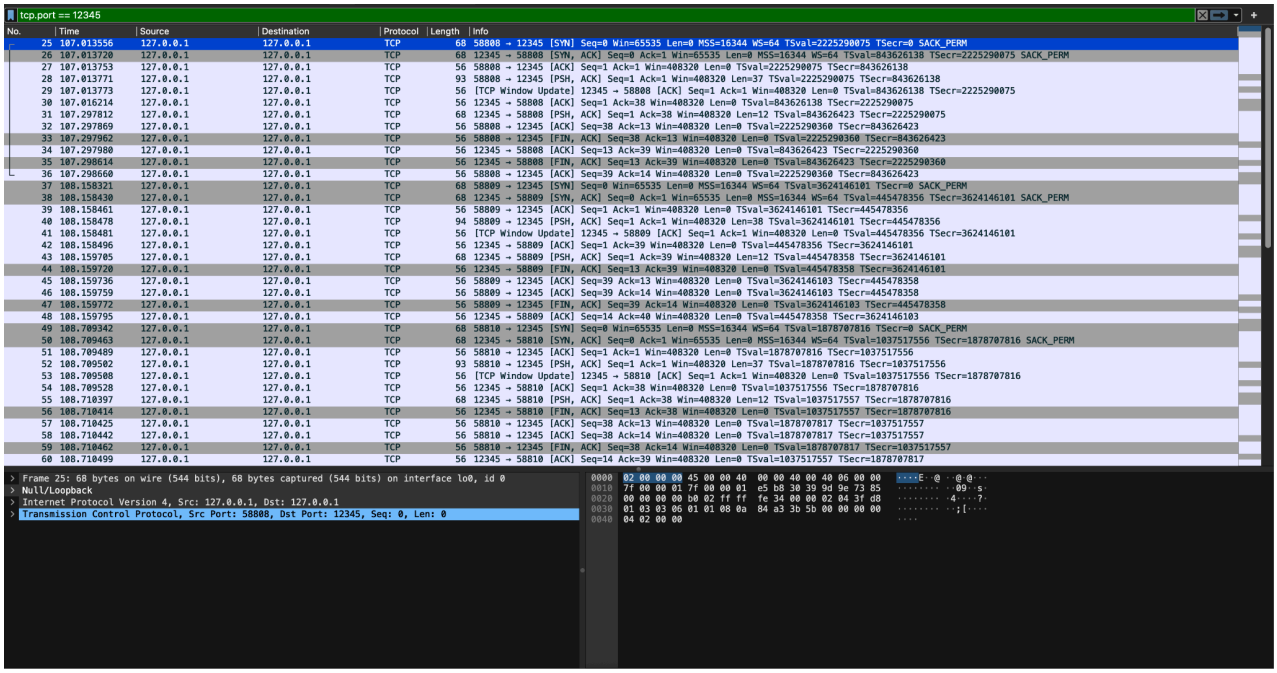
```
(env) (base) sohamshrivastava@Sohams-MacBook-Air-10 CN_Assignment_1 % python3 client.py
Starting to process packets:


Query ID: 0, Domain: netflix.com., Resolved IP: 192.168.1.11
Query ID: 1, Domain: linkedin.com., Resolved IP: 192.168.1.12
Query ID: 2, Domain: example.com., Resolved IP: 192.168.1.13
Query ID: 3, Domain: google.com., Resolved IP: 192.168.1.14
Query ID: 4, Domain: facebook.com., Resolved IP: 192.168.1.15
Query ID: 5, Domain: amazon.com., Resolved IP: 192.168.1.11
summary of queries:

   Custom Header value (HHMMSSID)    Domain name Resolved IP address
0                        23112700   netflix.com.        192.168.1.11
1                        23112801  linkedin.com.        192.168.1.12
2                        23112902   example.com.        192.168.1.13
3                        23112903    google.com.        192.168.1.14
4                        23113004  facebook.com.        192.168.1.15
5                        23113005    amazon.com.        192.168.1.11
report saved as 'dns_report.csv'
```

Fig: output from client.py

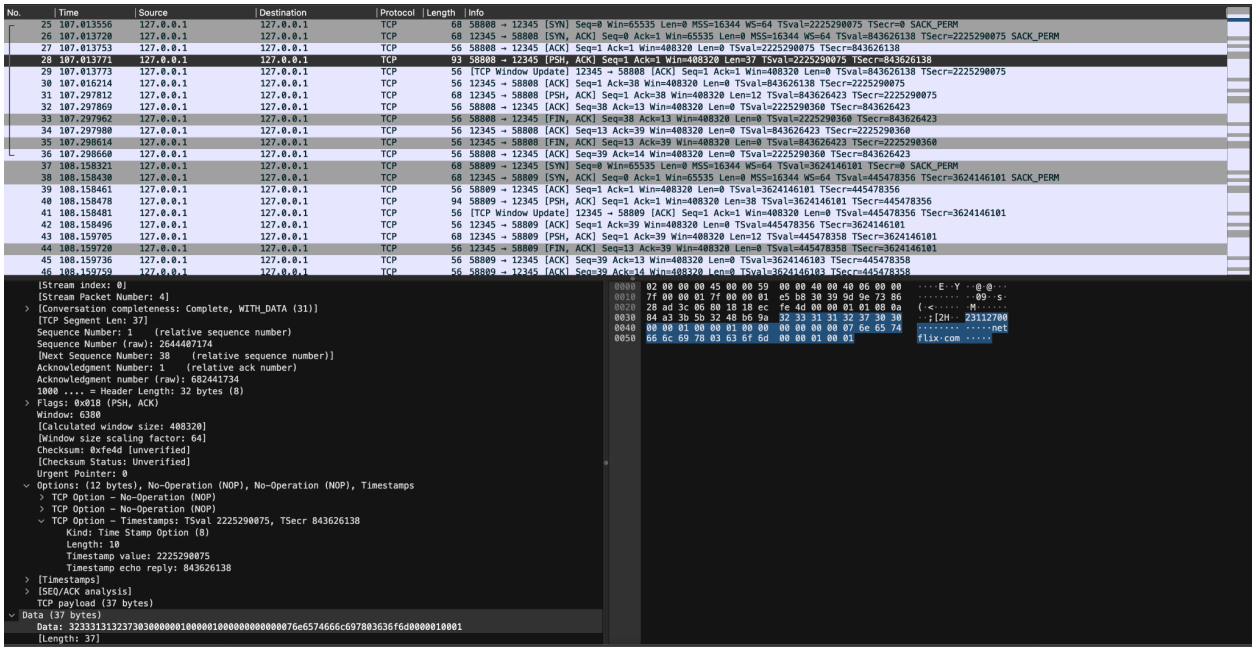
To analyse the background workflow, we analysed the packets using Wireshark on the loopback interface (lo0) (for local capturing) with the filter "tcp.port == 12345".



First, the connection setup is done using the 3-way handshake:
The client sends a SYN packet to request a connection, the server replies with SYN, ACK to acknowledge, and finally, the client responds with ACK to confirm. This establishes the connection between client and server.

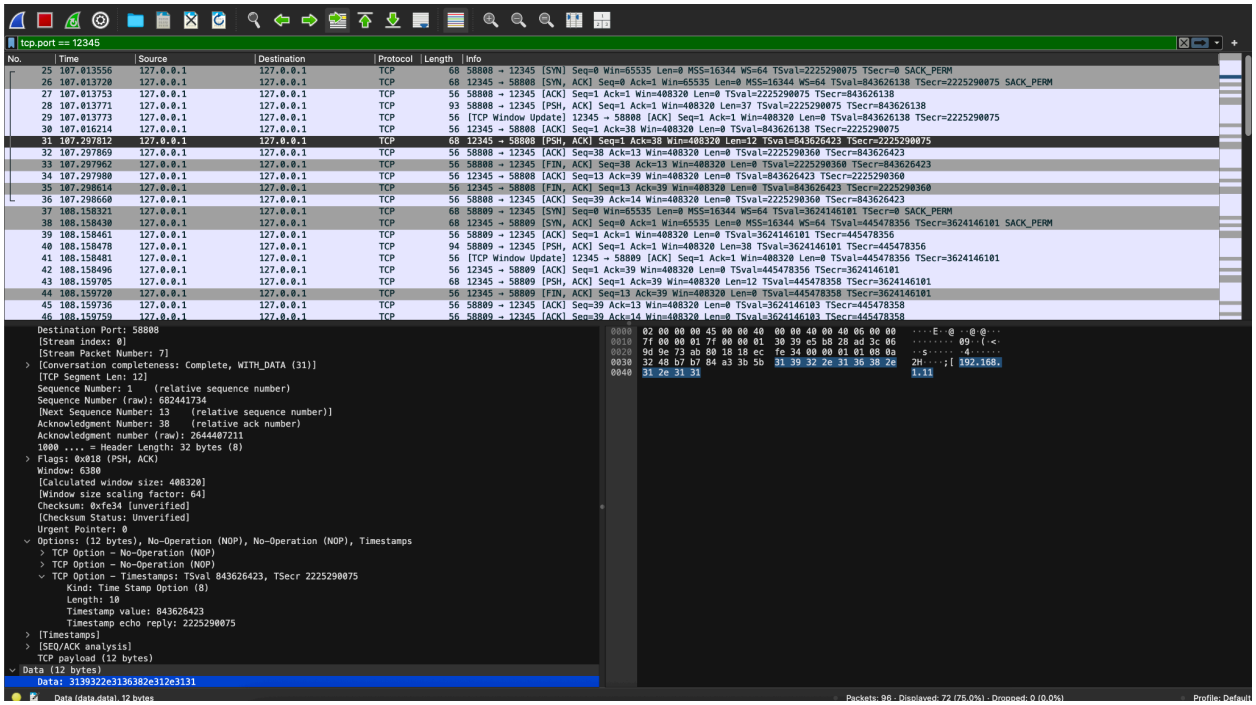
Now, during data transfer, PSH and ACK flags were used. The PSH flag indicates that the client is sending data (pushing) to the server, while ACK confirms receipt of previous data.

We can also see the data that is sent and received by the client and server.



Here on the right side, the highlighted text shows the Custom Header (23112700) and website name (netflix.com), which is sent by the client.

Now, after acknowledging the received data server will send the resolved IP to the client.



Using the same PSH, ACK flags, the server sends the resolved IP (here, 192.168.1.11) back to the client.

Now, after acknowledging the received IP address, our TCP connection gets closed, which can be observed using FIN, ACK flags.

For every response a new TCP connection has been made and get closed after the client gets the response IP address. This shows our non-persistent implementation of our TCP connection.

Custom Header value (HHMMSSID)	Domain name	Resolved IP address
23112700	netflix.com.	192.168.1.11
23112801	linkedin.com.	192.168.1.12
23112902	example.com.	192.168.1.13
23112903	google.com.	192.168.1.14
23113004	facebook.com.	192.168.1.15
23113005	amazon.com.	192.168.1.11

Fig: Final DNS report from our 3.Pcap File

Task 2

```
arinmehta@Arins-MacBook-Pro-4 ~ % traceroute www.google.com
traceroute to www.google.com (142.251.222.100), 64 hops max, 40 byte packets
 1  10.7.0.5 (10.7.0.5)  15.381 ms  3.591 ms  3.033 ms
 2  172.16.4.7 (172.16.4.7)  3.544 ms  3.379 ms  3.583 ms
 3  14.139.98.1 (14.139.98.1)  5.134 ms  4.776 ms  5.327 ms
 4  10.117.81.253 (10.117.81.253)  3.170 ms  3.790 ms  3.147 ms
 5  10.154.8.137 (10.154.8.137)  11.866 ms  11.200 ms  11.896 ms
 6  10.255.239.170 (10.255.239.170)  12.324 ms  10.768 ms  11.439 ms
 7  10.152.7.214 (10.152.7.214)  11.695 ms  11.198 ms  11.772 ms
 8  72.14.204.62 (72.14.204.62)  11.665 ms  12.880 ms  11.617 ms
 9  * * *
10  142.250.227.74 (142.250.227.74)  28.184 ms
    192.178.86.246 (192.178.86.246)  13.738 ms
    142.250.227.74 (142.250.227.74)  22.251 ms
11  192.178.110.106 (192.178.110.106)  12.535 ms
    192.178.110.104 (192.178.110.104)  12.644 ms
    192.178.110.206 (192.178.110.206)  22.134 ms
12  pnbomb-az-in-f4.1e100.net (142.251.222.100)  12.503 ms
    142.250.209.71 (142.250.209.71)  13.711 ms
    pnbomb-az-in-f4.1e100.net (142.251.222.100)  12.261 ms
arinmehta@Arins-MacBook-Pro-4 ~ %
```

Fig.1: traceroute www.google.com on MacOS

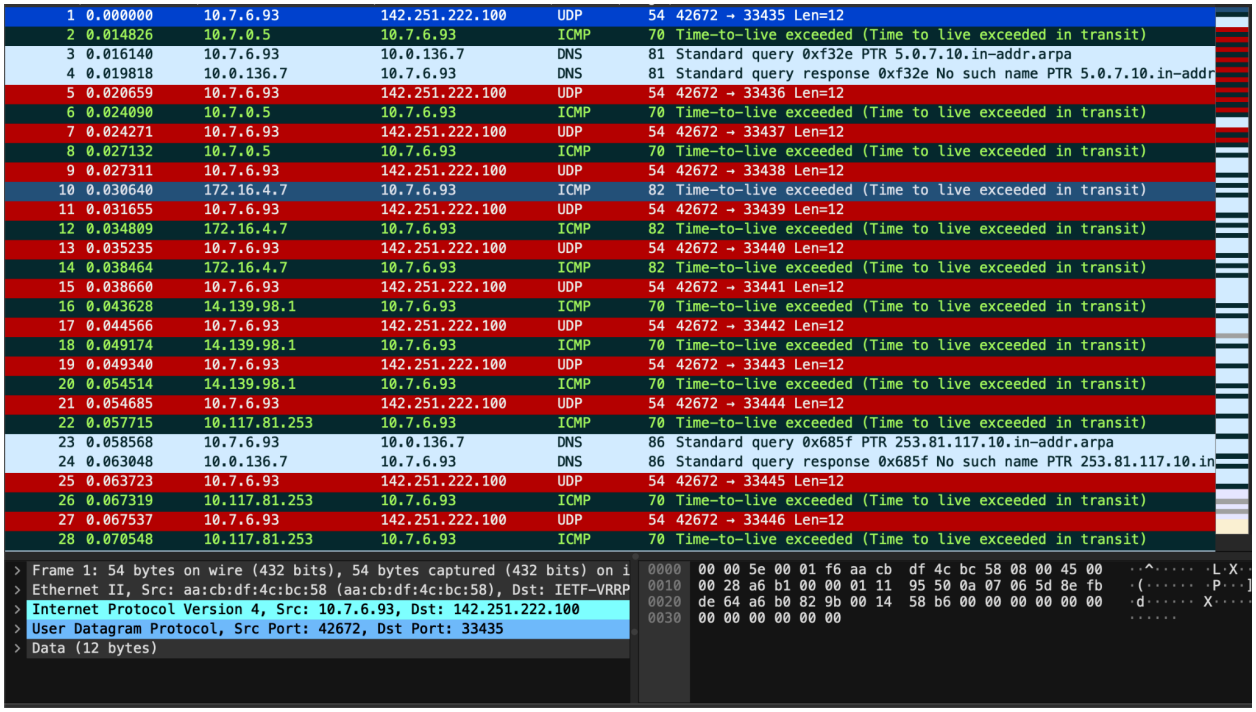


Fig.2: Wireshark screenshot for the corresponding macOS command

```
C:\Users\bhavy>tracert www.google.com

Tracing route to www.google.com [142.251.42.68]
over a maximum of 30 hops:

  1    10 ms     3 ms     3 ms  10.240.0.2
  2    12 ms     2 ms     2 ms  10.3.0.29
  3    17 ms     2 ms     2 ms  10.3.0.5
  4    14 ms     5 ms     2 ms  172.16.4.7
  5    15 ms     5 ms     5 ms  14.139.98.1
  6    13 ms     2 ms     2 ms  10.117.81.253
  7    34 ms    24 ms    22 ms  10.154.8.137
  8    49 ms    41 ms    41 ms  10.255.239.170
  9    58 ms    45 ms    45 ms  10.152.7.214
 10    60 ms    55 ms    58 ms  72.14.204.62
 11    71 ms    69 ms    67 ms  72.14.239.103
 12    23 ms    18 ms    17 ms  142.251.69.105
 13    68 ms    71 ms    73 ms  bom12s21-in-f4.1e100.net [142.251.42.68]

Trace complete.
```

Fig.3: traceroute www.google.com on Windows

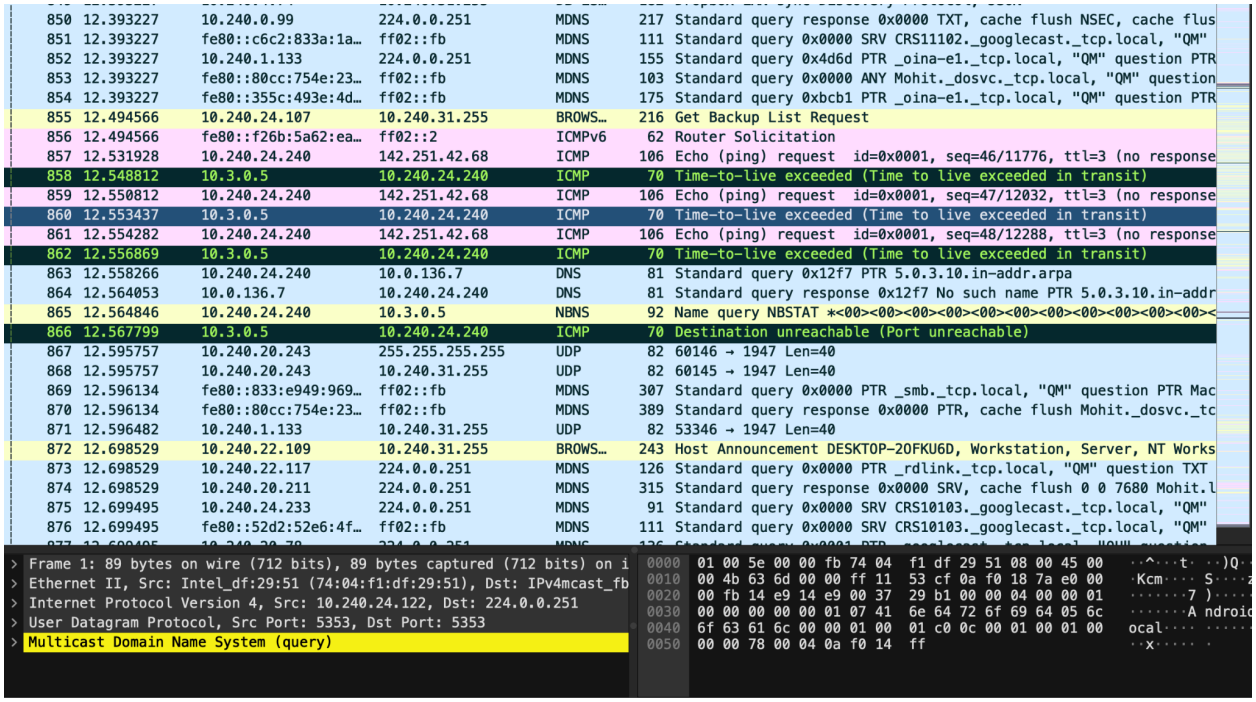


Fig.4: Wireshark screenshot for the corresponding Windows command

Question 1:

From the above screenshots, we can see that macOS(it has a close resemblance to Linux due to its Unix ancestor) uses UDP to send requests to port 33435 and above. The selected frame in the screenshot is the first frame, and the subsequent frames have the destination port as increments of 33435. The ICMP replies that we see are the TTL-exceeded replies.

Windows uses ICMP echo requests, as seen in Fig.4. We see that the replies are ICMP TTL-exceeded replies.

Thus, we can conclude that Linux traceroute uses UDP(User Datagram Protocol) by default and Windows traceroute uses ICMP echo by default.

Question 2:

In Fig.1, we can see that the 9th hop has output ***. Reasons why the router might not reply are:

1. Packets may be dropped by the firewall.
2. The destination server does not respond, either due to the service being unavailable or because it blocks probes.
3. ICMP responses may be disabled by configuration.

Question 3:

As we can see in Fig.2, the destination port increments on each successive port. The first destination port, as seen in frame 1, is 33435.

Question 4:

For macOS, at the final hop, we can see that the ICMP response is "*Destination unreachable (Port unreachable)"*, whereas in the intermediate hops, we get the ICMP "*Time-to-live exceeded (Time to live exceeded in transit)"* response. The screenshot of the same is attached below.

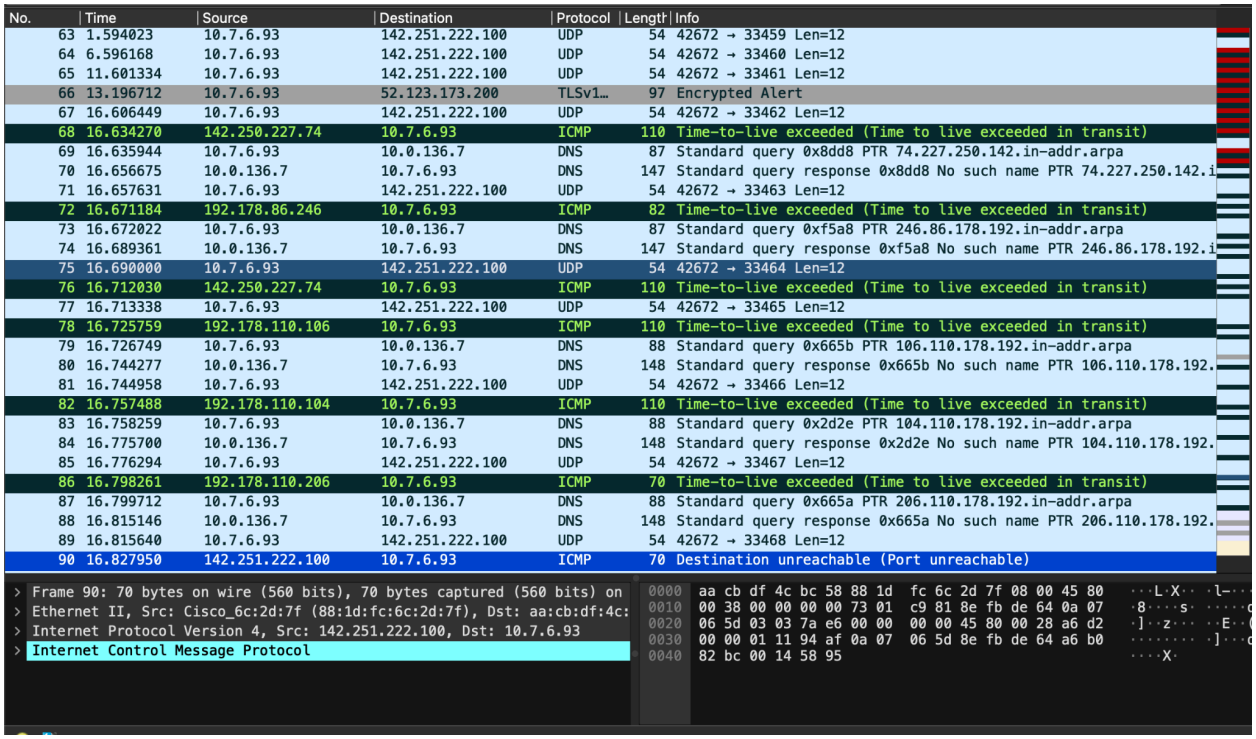


Fig.5: Wireshark screenshot displaying the final hop on macOS

For windows, at the final hop, we can see that at the final hop, the response is ICMP Echo Reply, whereas in intermediate hops, we get the ICMP "*Time-to-live exceeded (Time to live exceeded in transit)"* response. The screenshot of the same is attached below.

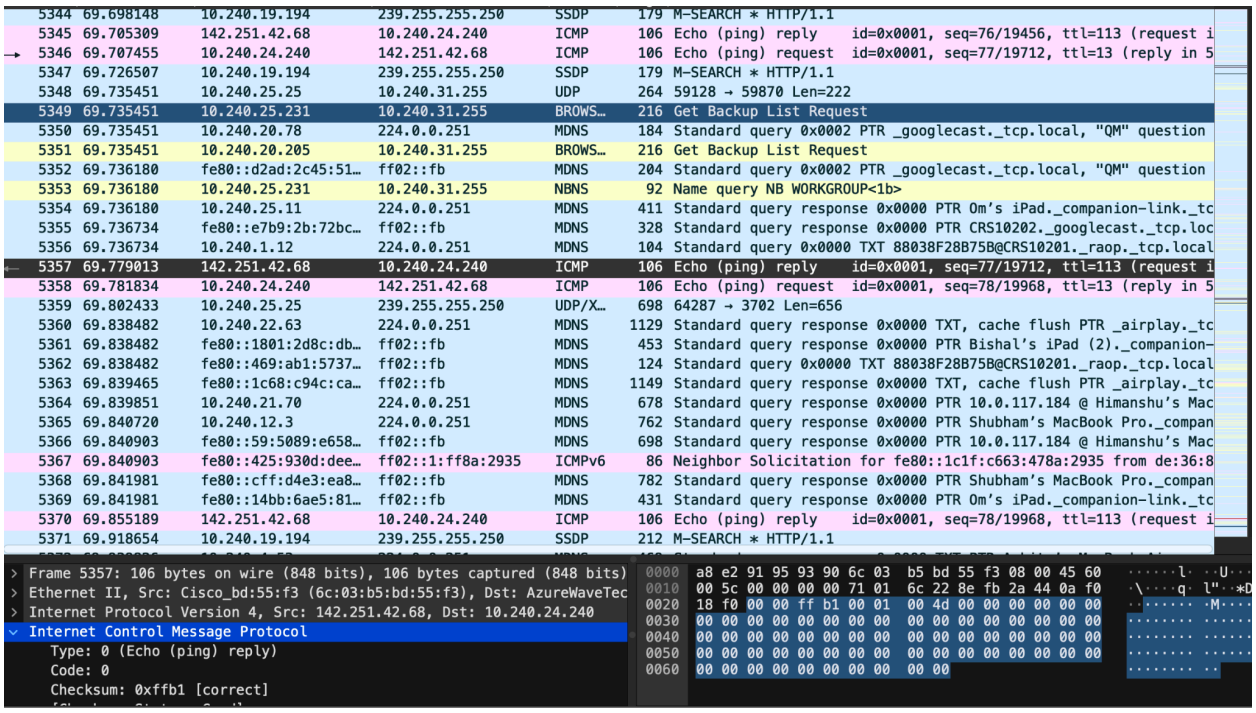


Fig.6: Wireshark screenshot displaying the final hop on Windows

Question 5:

If the firewall blocks UDP but allows ICMP the linux traceroute will not work as it uses UDP. On the other hand, windows traceroute will continue working as it is because it uses ICMP echo.