COMPUTER NETWORKS (CS-331)

ASSIGNMENT 1

- CHAUDHARI MESHVA (23110075)

- BANAVATH DIRAJ (22110044)

Introduction

This assignment consists of two major tasks:

1. Implementing a **custom DNS resolver** using packet parsing and socket programming.
    - The client extracts DNS queries from a PCAP file, appends a custom header, and sends them to the server.
    - The server applies **time-based load-balancing rules** to resolve queries into IP addresses.
2. Understanding the behavior of the **traceroute utility** on different operating systems.
    - Observing the underlying protocols.
    - Capturing traffic with Wireshark/tcpdump.
    - Explaining why outputs differ across systems.

Task-1: DNS Resolver

Methodology

The DNS resolver was implemented in Python using the **Scapy** library and the built-in **socket** module. The workflow was:

Client side:

In the client side the process begins by loading the provided PCAP file and filtering out DNS queries. In this case, the file selected is named *9.pcap* (derived from the calculation (075+044)/10 = 9).

For each DNS query, a custom header is generated in the format **HHMMSSID**, where

- HH = hour in 24-hour format

- `MM` = minute
- `SS` = second
- `ID` = sequence number of query (starting from 00)

This custom header is then prepended to the **DNS query packet**, and the combined packet is transmitted to the server over a **TCP connection**.

Server side:

The received packet is processed by first extracting both the custom header and the DNS query.The header is parsed to determine key details such as the hour of the request and the query ID. Based on the hour, the server applies time-slot rules:

- Morning (04:00–11:59) → use IPs 192.168.1.1 to 192.168.1.5
- Afternoon (12:00–19:59) → use IPs 192.168.1.6 to 192.168.1.10
- Night (20:00–03:59) → use IPs 192.168.1.11 to 192.168.1.15

The exact IP address is chosen by computing **(ID % 5)**, ensuring a consistent mapping within each time slot. The resolved IP is then returned to the client.

Finally, on the client side, all results are logged systematically. For each query, the client records the generated header, the domain name extracted from the DNS query, and the resolved IP address. These results are saved into a CSV file named *dns_results.csv*, which provides a structured format suitable for analysis and direct inclusion in the final report.

Results

The Client-server program was executed successfully. After running the server file. It will wait to connect the client. After we run the client it will take some time to connect. It connected to the server on 127.0.0.1 with an assigned port.

```
PS C:\Users\chaud\OneDrive\Desktop\CN_Assignment1> python server.py
Server running... waiting for client
Client connected: ('127.0.0.1', 63329)
Received header: 22414200 -> Resolved: 192.168.1.11
Received header: 22414201 -> Resolved: 192.168.1.12
Received header: 22414202 -> Resolved: 192.168.1.13
Received header: 22414203 -> Resolved: 192.168.1.14
Received header: 22414204 -> Resolved: 192.168.1.15
Received header: 22414205 -> Resolved: 192.168.1.11
Received header: 22414206 -> Resolved: 192.168.1.12
Received header: 22414207 -> Resolved: 192.168.1.13
Received header: 22414208 -> Resolved: 192.168.1.14
Received header: 22414209 -> Resolved: 192.168.1.15
Received header: 22414210 -> Resolved: 192.168.1.11
Received header: 22414211 -> Resolved: 192.168.1.12
Received header: 22414212 -> Resolved: 192.168.1.13
Received header: 22414213 -> Resolved: 192.168.1.14
Received header: 22414214 -> Resolved: 192.168.1.15
Received header: 22414215 -> Resolved: 192.168.1.11
Received header: 22414216 -> Resolved: 192.168.1.12
Received header: 22414217 -> Resolved: 192.168.1.13
Received header: 22414218 -> Resolved: 192.168.1.14
Received header: 22414219 -> Resolved: 192.168.1.15
Received header: 22414220 -> Resolved: 192.168.1.11
Received header: 22414221 -> Resolved: 192.168.1.12
Received header: 22414222 -> Resolved: 192.168.1.13
```

The client generated a sequence of headers and transmitted domain names and service discovery queries to the server. The server received each header and resolved the corresponding domain/service into IP addresses.

```
C:\Users\chaud\OneDrive\Desktop\CN_Assignment1>python client.py
22414200 | _apple-mobdev._tcp.local | 192.168.1.11
22414201 | _apple-mobdev._tcp.local | 192.168.1.12
22414202 | twitter.com | 192.168.1.13
22414203 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.14
22414204 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.15
22414205 | example.com | 192.168.1.11
22414206 | netflix.com | 192.168.1.12
22414207 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.13
22414208 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.14
22414209 | linkedin.com | 192.168.1.15
22414210 | _apple-mobdev._tcp.local | 192.168.1.11
22414211 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.12
22414212 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.13
22414213 | reddit.com | 192.168.1.14
22414214 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.15
22414215 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.11
22414216 | _apple-mobdev._tcp.local | 192.168.1.12
22414217 | _apple-mobdev._tcp.local | 192.168.1.13
22414218 | openai.com | 192.168.1.14
22414219 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.15
22414220 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.11
22414221 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.12
22414222 | Brother MFC-7860DW._pdl-datastream._tcp.local | 192.168.1.13
```

From the result we get, we can summarize as follows:

Domain Name Resolution

- twitter.com → 192.168.1.13
- netflix.com → 192.168.1.12
- linkedin.com → 192.168.1.15
- reddit.com → 192.168.1.14
- openai.com → 192.168.1.14

Local Network Service Discovery

- _apple-mobdev._tcp.local → 192.168.1.11, 192.168.1.12, 192.168.1.13
- Brother MFC-7860DW._pdl-datastream._tcp.local → 192.168.1.14, 192.168.1.15

We converted it into a CSV file containing Custom Header, Domain, and Resolved IP columns.

	A	B	C
1	Custom Header	Domain	Resolved IP
2	22414200	_apple-mobdev._tcp.local	192.168.1.11
3	22414201	_apple-mobdev._tcp.local	192.168.1.12
4	22414202	twitter.com	192.168.1.13
5	22414203	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.14
6	22414204	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.15
7	22414205	example.com	192.168.1.11
8	22414206	netflix.com	192.168.1.12
9	22414207	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.13
10	22414208	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.14
11	22414209	linkedin.com	192.168.1.15
12	22414210	_apple-mobdev._tcp.local	192.168.1.11
13	22414211	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.12
14	22414212	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.13
15	22414213	reddit.com	192.168.1.14
16	22414214	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.15
17	22414215	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.11
18	22414216	_apple-mobdev._tcp.local	192.168.1.12
19	22414217	_apple-mobdev._tcp.local	192.168.1.13
20	22414218	openai.com	192.168.1.14
21	22414219	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.15
22	22414220	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.11
23	22414221	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.12
24	22414222	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.13

Thus, the program correctly simulated DNS-like resolution and service discovery. The server successfully received client queries, resolved them into IP addresses, and sent the responses back to the client. This validates the correctness of the client–server communication and the domain/service resolution mechanism.

Task-2: Traceroute Protocol Behavior

Methodology

We will use two different operating systems to trace the same address.

- In **Windows**, we will run the `tracert` command.
- In **Linux**, we will run the `traceroute` command.

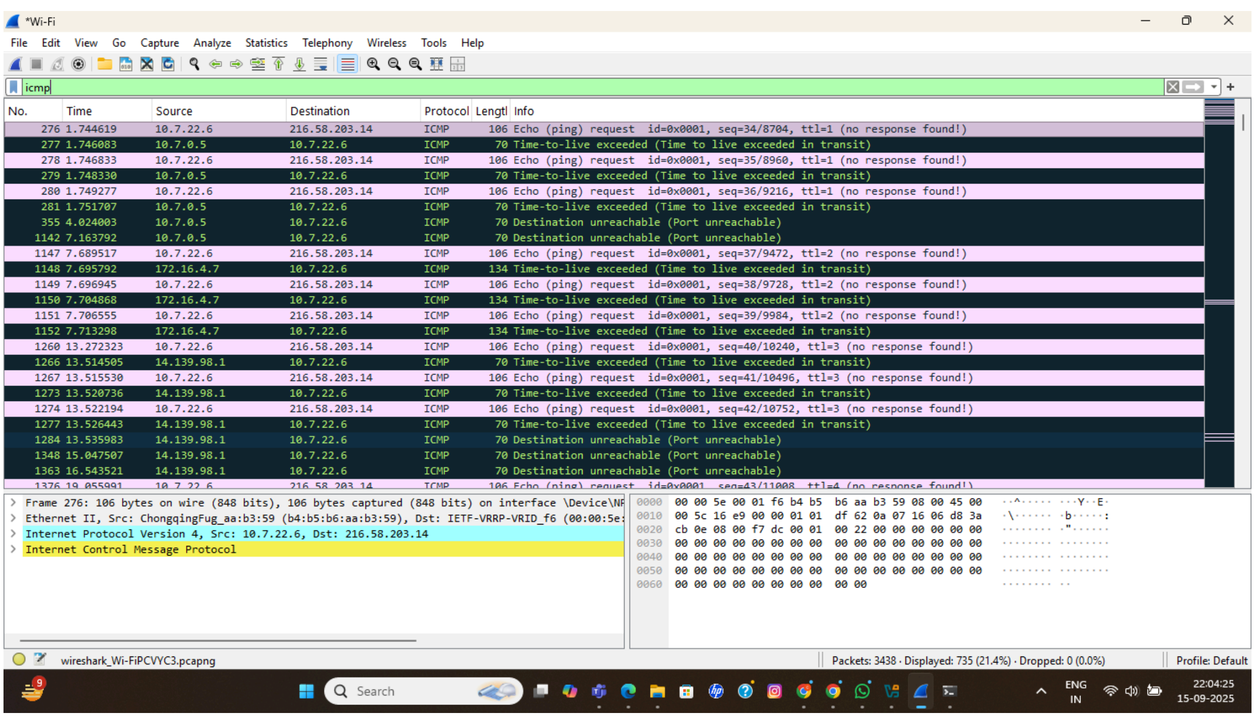
For this task, we are using www.youtube.com as the destination. In both cases, we will capture the network traffic using **Wireshark** (on Windows).

By the Observations

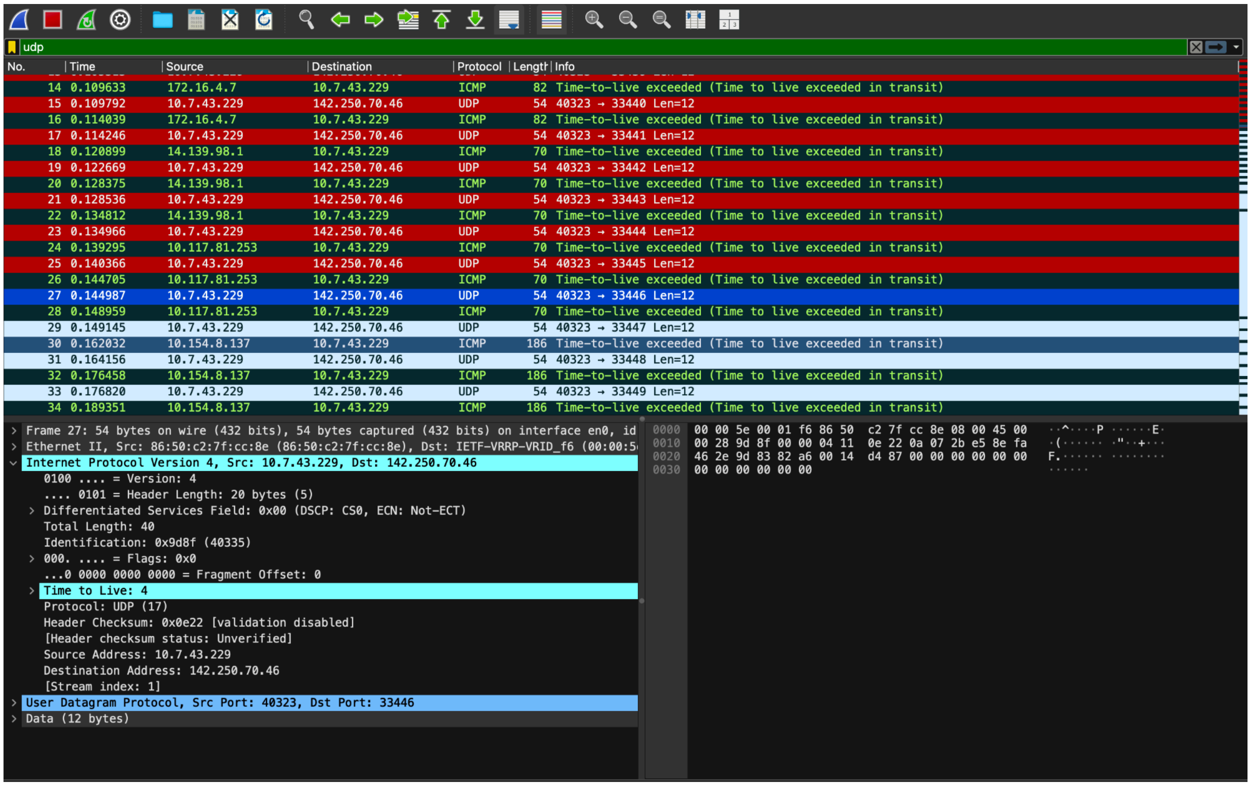
I have provided the answer belove,

1. **What protocol does windows tracert use by default, and what protocol does Linux traceroute use by default?**

   By default, the **Windows `tracert` command** uses the **ICMP Echo Request protocol** to trace the route to the destination.



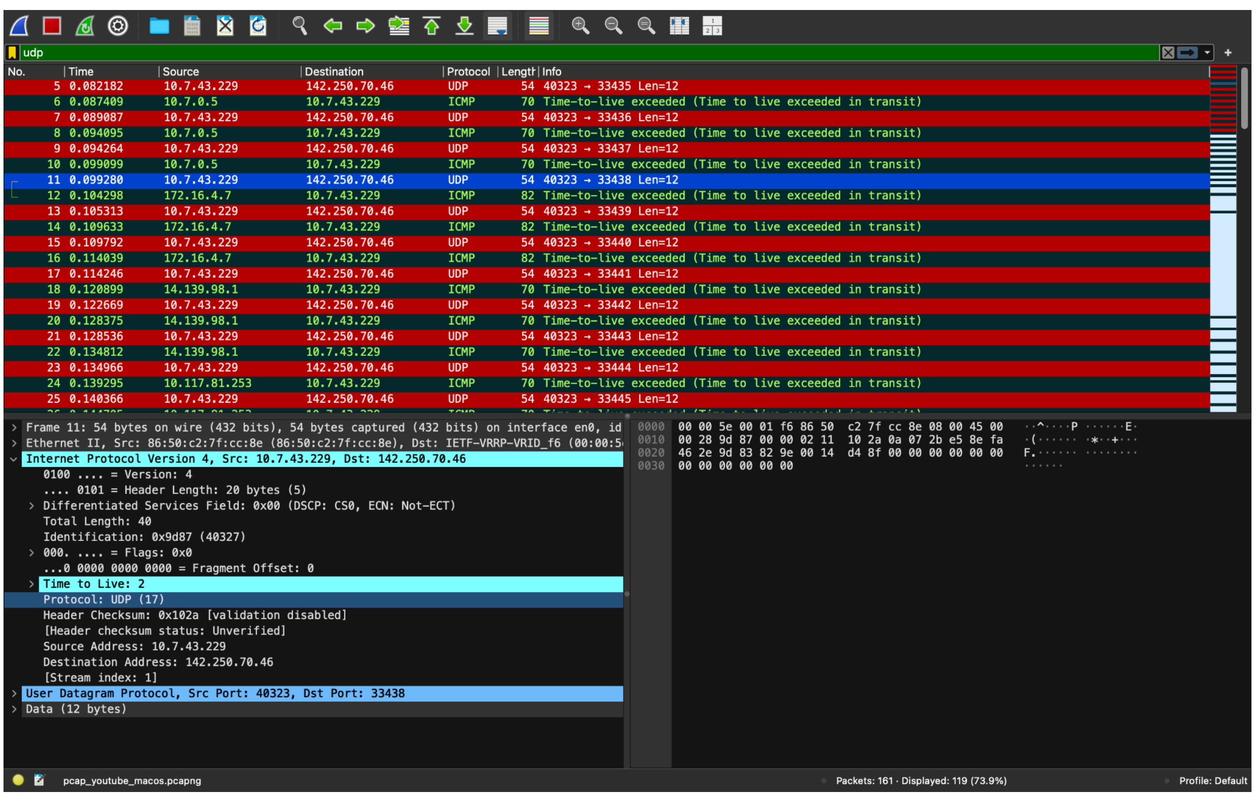
On the other hand, the **Linux `traceroute` command** by default uses **UDP probe packets.**

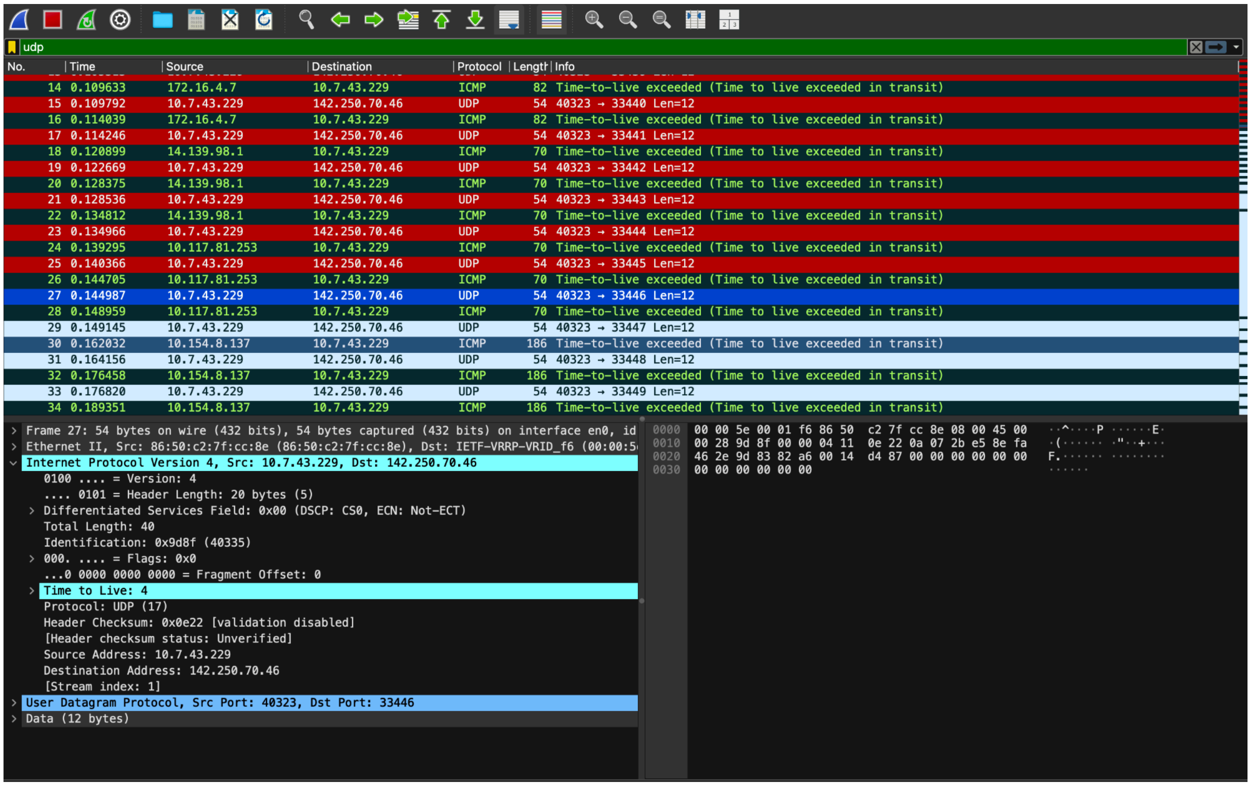


2. **Some hops in your tracert output may show ***. Provide at least reasons why a route might not reply.**
    - The router is configured not to send ICMP Time Exceeded messages for security or policy reasons.
    - A firewall or access control list (ACL) is blocking ICMP responses, preventing the probe packets from returning.
    - The router may be overloaded or dropping probe packets due to congestion.

```
(base) Banavaths-MacBook-Air:~ banavathdirajnaik$ traceroute www.youtube.com
traceroute: Warning: www.youtube.com has multiple addresses; using 142.250.70.46
traceroute to youtube-ui.l.google.com (142.250.70.46), 64 hops max, 40 byte packets
 1  10.7.0.5 (10.7.0.5)  5.805 ms  5.160 ms  5.010 ms
 2  172.16.4.7 (172.16.4.7)  5.152 ms  4.489 ms  4.429 ms
 3  14.139.98.1 (14.139.98.1)  6.815 ms  5.863 ms  6.418 ms
 4  10.117.81.253 (10.117.81.253)  4.417 ms  4.547 ms  4.169 ms
 5  10.154.8.137 (10.154.8.137)  13.233 ms  12.601 ms  12.828 ms
 6  10.255.239.170 (10.255.239.170)  12.427 ms  12.291 ms  13.301 ms
 7  10.152.7.214 (10.152.7.214)  12.129 ms  12.232 ms  11.783 ms
 8  * 72.14.204.62 (72.14.204.62)  22.294 ms  13.096 ms
 9  * * *
10  142.251.77.98 (142.251.77.98)  23.478 ms
    142.250.227.72 (142.250.227.72)  17.413 ms
    142.250.238.196 (142.250.238.196)  14.200 ms
11  192.178.110.104 (192.178.110.104)  14.037 ms
    192.178.86.243 (192.178.86.243)  17.261 ms
    192.178.110.104 (192.178.110.104)  14.177 ms
12  pnbomb-aa-in-f14.1e100.net (142.250.70.46)  13.943 ms
    192.178.110.245 (192.178.110.245)  13.982 ms
    142.250.209.71 (142.250.209.71)  14.400 ms
(base) Banavaths-MacBook-Air:~ banavathdirajnaik$
```

3. **In Linux traceroute, which field in the probe packets changes between successive probes sent to the destination?**





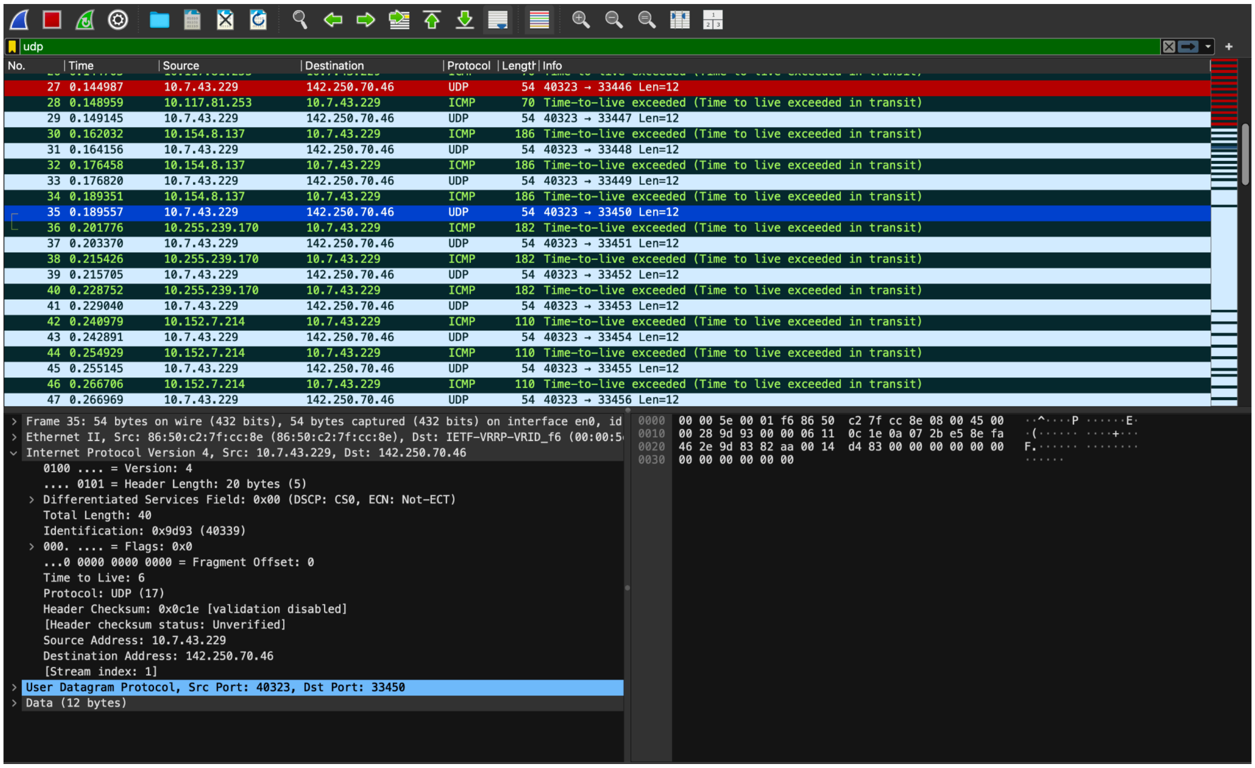
In Linux traceroute, the UDP destination port number changes for each successive probe packet.

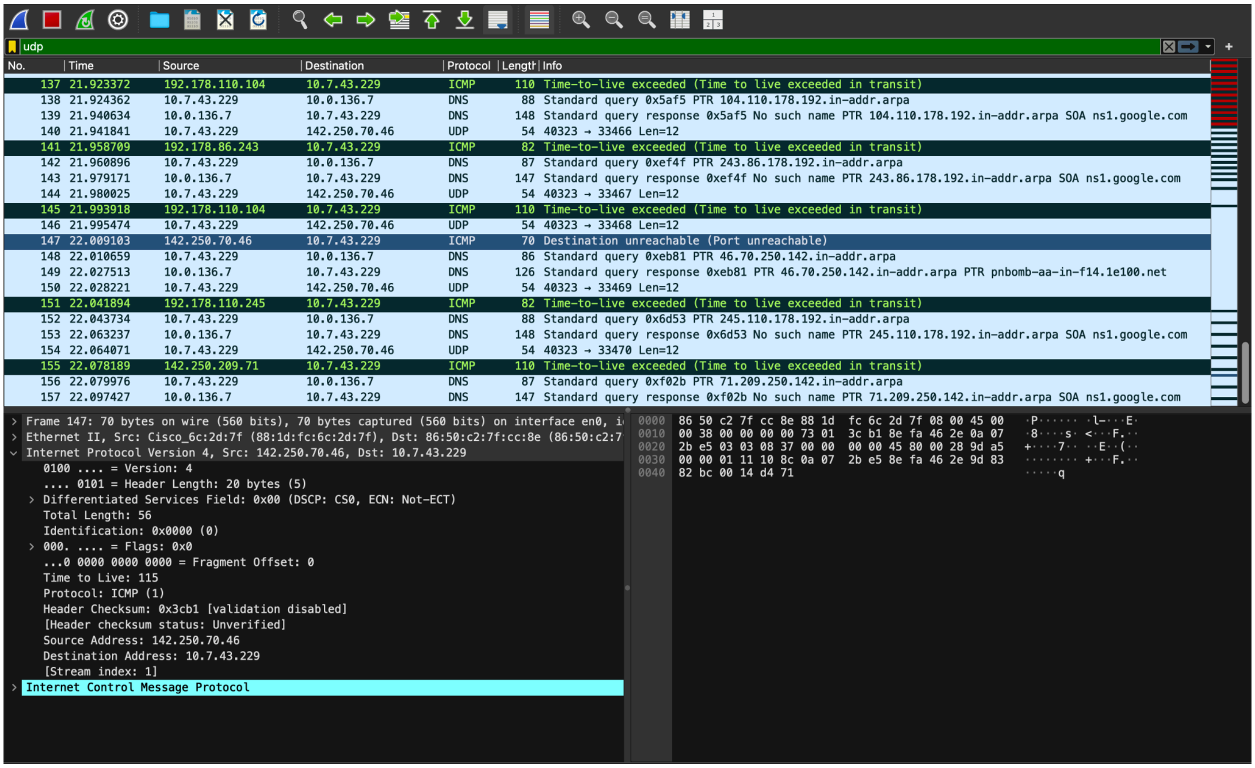
- One probe shows Source Port: 40323, Destination Port: 33438
- Another probe shows Source Port: 40323, Destination Port: 33446
- This confirms that while the source port remains the same, the destination port increments with each probe. This behavior helps the traceroute tool match ICMP responses to the correct probe packet.

4. At the final hop, how is the response different compared to the intermediate hop?

At the intermediate hops, the response you see is ICMP "Time-to-Live exceeded in transit" messages. This happens because each router along the path decrements the TTL field of the packet, and when TTL reaches zero, the router discards the packet and sends back an ICMP Time Exceeded message.

At the final hop, the behavior changes: instead of a "TTL exceeded" message, you get an ICMP "Destination unreachable (Port unreachable)" message. This is because the probe packet finally reaches the destination host, but the host has no application listening on the given UDP port, so it responds with "Port unreachable".





5. **Suppose a firewall blocks UDP traffic but allows ICMP - how would this affect the results of LInux traceroute vs. Windows tracert?**
    - Linux traceroute (which relies on UDP probes by default) would fail, because the UDP packets would be blocked by the firewall. As a result, the output would mostly show * * * for each hop, and the destination would not be reached.
    - Windows tracert (which uses ICMP Echo Requests by default) would still work correctly, because ICMP packets are allowed through the firewall. The hops and the destination would appear normally in the output.

Conclusion

In Task-1, the DNS resolver was successfully built with Python. It demonstrated how DNS queries can be parsed, enhanced with custom headers, and resolved using a set of predefined load-balancing rules. The use of time-based slots and modulo arithmetic for IP selection provided a practical understanding of how servers can distribute traffic intelligently.

In Task-2, the traceroute experiments highlighted how operating systems differ in their implementation of network diagnostic tools. By comparing ICMP-based probing in Windows with UDP-based probing in Linux, we understood why outputs differ and how firewalls or routers influence visibility. Together, both tasks reinforced key concepts of networking at the packet level and the importance of protocol choice.

References

- Windows tracert documentation: https://www.catchpoint.com/network-admin-guide/how-to-read-a-traceroute
- Linux man traceroute: https://linux.die.net/man/8/traceroute
- Wireshark Documentation: https://www.wireshark.org/docs/