

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.

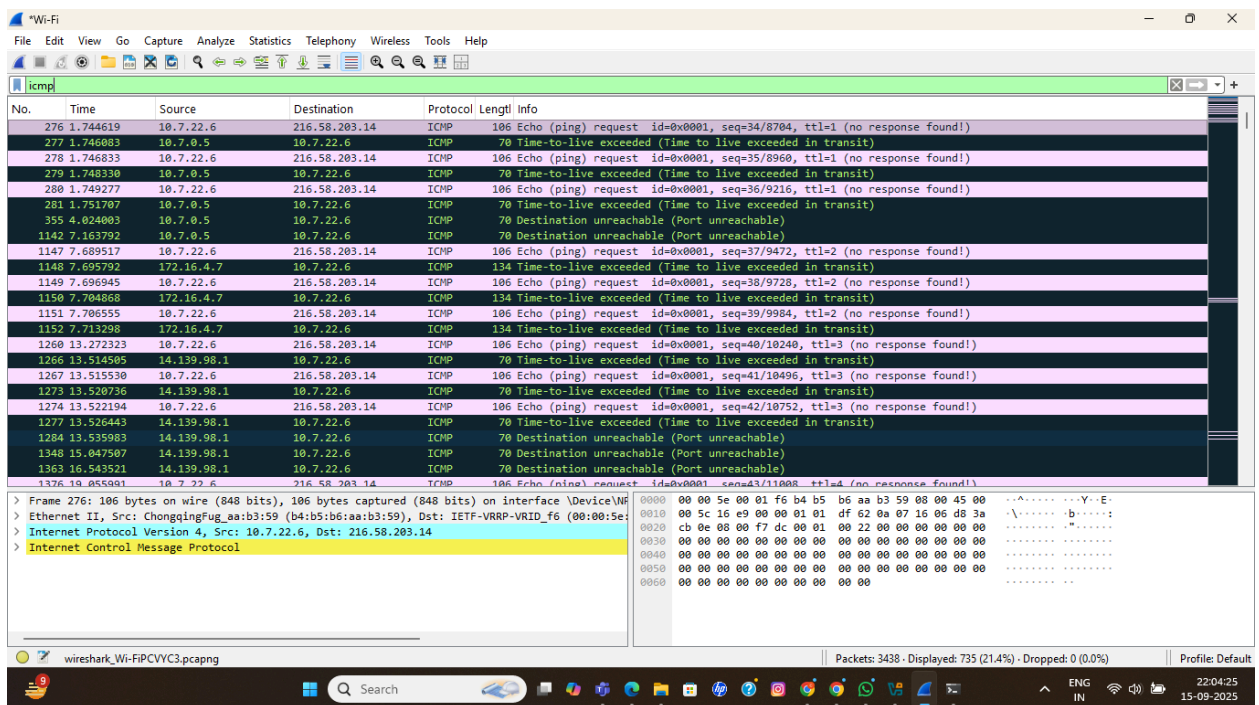
For this task, we are using www.youtube.com as the destination. In both cases, we will capture the network traffic using **Wireshark** (on Windows).

By the Observations

I have provided the answer below,

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**.

No.	Time	Source	Destination	Protocol	Length	Info
14	0.109633	172.16.4.7	10.7.43.229	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
15	0.109792	10.7.43.229	142.250.70.46	UDP	54	40323 → 33440 Len=12
16	0.114039	172.16.4.7	10.7.43.229	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
17	0.114246	10.7.43.229	142.250.70.46	UDP	54	40323 → 33441 Len=12
18	0.120099	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	0.122669	10.7.43.229	142.250.70.46	UDP	54	40323 → 33442 Len=12
20	0.128375	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.128536	10.7.43.229	142.250.70.46	UDP	54	40323 → 33443 Len=12
22	0.134812	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	0.134966	10.7.43.229	142.250.70.46	UDP	54	40323 → 33444 Len=12
24	0.139295	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
25	0.140366	10.7.43.229	142.250.70.46	UDP	54	40323 → 33445 Len=12
26	0.144705	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
27	0.144987	10.7.43.229	142.250.70.46	UDP	54	40323 → 33446 Len=12
28	0.148959	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
29	0.149145	10.7.43.229	142.250.70.46	UDP	54	40323 → 33447 Len=12
30	0.162032	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
31	0.164156	10.7.43.229	142.250.70.46	UDP	54	40323 → 33448 Len=12
32	0.176458	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
33	0.176820	10.7.43.229	142.250.70.46	UDP	54	40323 → 33449 Len=12
34	0.189351	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)

```
> Frame 27: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface en0, id 0000 00 00 5e 00 01 f6 86 50 c2 7f cc 8e 08 00 45 00 ...P...E
> Ethernet II, Src: 86:50:c2:7f:cc:8e (86:50:c2:7f:cc:8e), Dst: IETF-VRRP-VRID_f6 (00:00:5e:00:00:00) 0010 00 28 9d 8f 00 00 04 11 0e 22 0a 07 2b e5 8e fa ...+...+...
> Internet Protocol Version 4, Src: 10.7.43.229, Dst: 142.250.70.46 0020 46 2e 9d 83 82 a6 00 14 d4 87 00 00 00 00 00 ...F.....
> 0100 .... = Version: 4
> .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
> Total Length: 40
> Identification: 0x9d8f (40335)
> 000. .... = Flags: 0x0
> ...0 0000 0000 0000 = Fragment Offset: 0
> Time to Live: 4
> Protocol: UDP (17)
> Header Checksum: 0x0e22 [validation disabled]
> [Header checksum status: Unverified]
> Source Address: 10.7.43.229
> Destination Address: 142.250.70.46
> [Stream index: 1]
> User Datagram Protocol, Src Port: 40323, Dst Port: 33446
> Data (12 bytes)
```

2. Some hops in your traceroute 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.

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

No.	Time	Source	Destination	Protocol	Length	Info
14	0.109633	172.16.4.7	10.7.43.229	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
15	0.109792	10.7.43.229	142.250.70.46	UDP	54	40323 → 33440 Len=12
16	0.114039	172.16.4.7	10.7.43.229	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
17	0.114246	10.7.43.229	142.250.70.46	UDP	54	40323 → 33441 Len=12
18	0.120899	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	0.122669	10.7.43.229	142.250.70.46	UDP	54	40323 → 33442 Len=12
20	0.128375	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.128536	10.7.43.229	142.250.70.46	UDP	54	40323 → 33443 Len=12
22	0.134812	14.139.98.1	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	0.134966	10.7.43.229	142.250.70.46	UDP	54	40323 → 33444 Len=12
24	0.139295	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
25	0.140366	10.7.43.229	142.250.70.46	UDP	54	40323 → 33445 Len=12
26	0.144705	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
27	0.144987	10.7.43.229	142.250.70.46	UDP	54	40323 → 33446 Len=12
28	0.148959	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
29	0.149145	10.7.43.229	142.250.70.46	UDP	54	40323 → 33447 Len=12
30	0.162032	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
31	0.164156	10.7.43.229	142.250.70.46	UDP	54	40323 → 33448 Len=12
32	0.176458	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
33	0.176820	10.7.43.229	142.250.70.46	UDP	54	40323 → 33449 Len=12
34	0.189351	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)

```

> Frame 27: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface en0, id 0000 00 00 5e 00 01 f6 86 50 c2 7f cc 8e 08 00 45 00 ...P .....E:
> Ethernet II, Src: 86:50:c2:7f:cc:8e (86:50:c2:7f:cc:8e), Dst: IETF-VRRP-VRID_f6 (00:00:5:0010 00 28 9d 8f 00 00 04 11 0e 22 0a 07 2b e5 8e fa ...(:.....".+...
0020 46 2e 9d 83 82 a6 00 14 d4 87 00 00 00 00 00 00 F.....
0030 00 00 00 00 00 00
> Internet Protocol Version 4, Src: 10.7.43.229, Dst: 142.250.70.46
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x9d8f (40335)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
  > Time to Live: 4
    Protocol: UDP (17)
    Header Checksum: 0x0e22 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.7.43.229
    Destination Address: 142.250.70.46
    [Stream index: 1]
> User Datagram Protocol, Src Port: 40323, Dst Port: 33446
> Data (12 bytes)

```

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".

The image shows a Wireshark packet capture window titled 'udp'. The packet list pane displays 27 packets. Most are ICMP 'Time-to-live exceeded (Time to live exceeded in transit)' messages from 10.7.43.229 to 142.250.70.46. The final packet (No. 27) is a UDP packet from 10.7.43.229 to 142.250.70.46, port 40323 to 33446, with a length of 54 bytes. The packet details pane shows the structure of this final packet: Ethernet II, Internet Protocol Version 4, and User Datagram Protocol. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
27	0.144987	10.7.43.229	142.250.70.46	UDP	54	40323 → 33446 Len=12
28	0.148959	10.117.81.253	10.7.43.229	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
29	0.149145	10.7.43.229	142.250.70.46	UDP	54	40323 → 33447 Len=12
30	0.162032	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
31	0.164156	10.7.43.229	142.250.70.46	UDP	54	40323 → 33448 Len=12
32	0.176458	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
33	0.176820	10.7.43.229	142.250.70.46	UDP	54	40323 → 33449 Len=12
34	0.189351	10.154.8.137	10.7.43.229	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
35	0.189557	10.7.43.229	142.250.70.46	UDP	54	40323 → 33450 Len=12
36	0.201776	10.255.239.170	10.7.43.229	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
37	0.203370	10.7.43.229	142.250.70.46	UDP	54	40323 → 33451 Len=12
38	0.215426	10.255.239.170	10.7.43.229	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
39	0.215705	10.7.43.229	142.250.70.46	UDP	54	40323 → 33452 Len=12
40	0.228752	10.255.239.170	10.7.43.229	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
41	0.229040	10.7.43.229	142.250.70.46	UDP	54	40323 → 33453 Len=12
42	0.240979	10.152.7.214	10.7.43.229	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
43	0.242891	10.7.43.229	142.250.70.46	UDP	54	40323 → 33454 Len=12
44	0.254929	10.152.7.214	10.7.43.229	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
45	0.255145	10.7.43.229	142.250.70.46	UDP	54	40323 → 33455 Len=12
46	0.266706	10.152.7.214	10.7.43.229	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
47	0.266969	10.7.43.229	142.250.70.46	UDP	54	40323 → 33456 Len=12

Frame 35: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface en0, id 0000 00 00 5e 00 01 f6 86 50 c2 7f cc 8e 08 00 45 00 ...P...E.
> Ethernet II, Src: 86:50:c2:7f:cc:8e (86:50:c2:7f:cc:8e), Dst: IETF-VRRP-VRID_16 (00:00:5e:00:00:00) ..(...+...
> Internet Protocol Version 4, Src: 10.7.43.229, Dst: 142.250.70.46 0020 46 2e 9d 83 82 aa 00 14 d4 83 00 00 00 00 00 F.....
0030 00 00 00 00 00 00 00

0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 40
Identification: 0x9d93 (40339)
> 0000 = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 6
Protocol: UDP (17)
Header Checksum: 0x0c1e [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.7.43.229
Destination Address: 142.250.70.46
[Stream index: 1]
> User Datagram Protocol, Src Port: 40323, Dst Port: 33450
> Data (12 bytes)

The screenshot displays a Wireshark capture of network traffic. The top section shows a list of packets, including ICMP Echo requests and DNS Standard Query messages. The bottom section provides a detailed view of a selected ICMP Echo Request packet, showing its structure and fields.

No.	Time	Source	Destination	Protocol	Length/Info
137	21.923372	192.178.110.104	10.7.43.229	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
138	21.924362	10.7.43.229	10.0.136.7	DNS	88 Standard query 0x5af5 PTR 104.110.178.192.in-addr.arpa
139	21.940634	10.0.136.7	10.7.43.229	DNS	148 Standard query response 0x5af5 No such name PTR 104.110.178.192.in-addr.arpa SOA ns1.google.com
140	21.941841	10.7.43.229	142.250.70.46	UDP	54 40323 → 33466 Len=12
141	21.958709	192.178.86.243	10.7.43.229	ICMP	82 Time-to-live exceeded (Time to live exceeded in transit)
142	21.960896	10.7.43.229	10.0.136.7	DNS	87 Standard query 0xef4f PTR 243.86.178.192.in-addr.arpa
143	21.979171	10.0.136.7	10.7.43.229	DNS	147 Standard query response 0xef4f No such name PTR 243.86.178.192.in-addr.arpa SOA ns1.google.com
144	21.980825	10.7.43.229	142.250.70.46	UDP	54 40323 → 33467 Len=12
145	21.993316	192.178.110.104	10.7.43.229	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
146	21.995474	10.7.43.229	142.250.70.46	UDP	54 40323 → 33468 Len=12
147	22.009103	142.250.70.46	10.7.43.229	ICMP	70 Destination unreachable (Port unreachable)
148	22.010659	10.7.43.229	10.0.136.7	DNS	86 Standard query 0xeb81 PTR 46.70.250.142.in-addr.arpa
149	22.027513	10.0.136.7	10.7.43.229	DNS	126 Standard query response 0xeb81 PTR 46.70.250.142.in-addr.arpa PTR pnbomb-aa-in-f14.1e100.net
150	22.028221	10.7.43.229	142.250.70.46	UDP	54 40323 → 33469 Len=12
151	22.041894	192.178.110.245	10.7.43.229	ICMP	82 Time-to-live exceeded (Time to live exceeded in transit)
152	22.043734	10.7.43.229	10.0.136.7	DNS	88 Standard query 0x6d53 PTR 245.110.178.192.in-addr.arpa
153	22.063237	10.0.136.7	10.7.43.229	DNS	148 Standard query response 0x6d53 No such name PTR 245.110.178.192.in-addr.arpa SOA ns1.google.com
154	22.064071	10.7.43.229	142.250.70.46	UDP	54 40323 → 33470 Len=12
155	22.078189	142.250.209.71	10.7.43.229	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
156	22.079976	10.7.43.229	10.0.136.7	DNS	87 Standard query 0xf02b PTR 71.209.250.142.in-addr.arpa
157	22.097427	10.0.136.7	10.7.43.229	DNS	147 Standard query response 0xf02b No such name PTR 71.209.250.142.in-addr.arpa SOA ns1.google.com

Packet details for Frame 147: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface en0, 1...
 Ethernet II, Src: Cisco_6c:2d:7f (88:1d:fc:6c:2d:7f), Dst: 86:50:c2:7f:cc:8e (86:50:c2:7...
 Internet Protocol Version 4, Src: 142.250.70.46, Dst: 10.7.43.229
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 56
 Identification: 0x0000 (0)
 > 000. = Flags: 0x0
 ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 115
 Protocol: ICMP (1)
 Header checksum: 0x3cb1 [validation disabled]
 [Header checksum status: Unverified]
 Source Address: 142.250.70.46
 Destination Address: 10.7.43.229
 [Stream index: 1]
 > Internet Control Message Protocol

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/>