



INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR

Department of Computer Science and Engineering

# CS-331: Computer Networks

---

## Assignment 1

---

**Submitted by:**

Anurag Singh (23110035)

Arjun B Dikshit (23110040)

**Instructor:**

Prof. Sameer Kulkarni

September 15, 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Question 1</b>	<b>2</b>
2.1	Methodology . . . . .	2
2.1.1	Client Implementation . . . . .	2
2.1.2	Server Implementation . . . . .	2
2.2	Results . . . . .	3
<b>3</b>	<b>Question 2</b>	<b>4</b>
3.1	Methodology . . . . .	4
3.2	Observations . . . . .	4
3.2.1	Windows . . . . .	4
3.2.2	Linux . . . . .	6
3.3	Results . . . . .	7
<b>4</b>	<b>Conclusion</b>	<b>9</b>

# 1 Introduction

The purpose of this report is to explore and implement fundamental networking concepts through two practical tasks. In the first task, we designed and deployed a custom DNS resolution system consisting of a client and a server. The system was required to process DNS queries from a packet capture file, augment them with a custom header, and resolve them into predefined IP addresses using server-side rules. This exercise provided practical experience with packet parsing, socket communication, and DNS protocol manipulation.

The second task focused on analyzing the behavior of the `tracert` and `tracert` utilities across two different operating systems: Windows and Linux. By capturing and studying the packets exchanged during route tracing, we investigated the default protocols employed by each tool, the causes of packet loss or missing hops, and the differences in response behavior between intermediate and final nodes. These experiments deepened our understanding of how various tools like Wireshark use different types of filters and rely on core Internet protocols such as ICMP and UDP to discover network paths.

Together, the two questions combined implementation and analysis, allowing us to get a deeper understanding of sockets and packet transfer protocols.

## 2 Question 1

### 2.1 Methodology

In this question, we were required to design and implement a custom DNS resolution system comprising a client and a server. We build a client that processes an input pcap file to get DNS query packets, appends a custom header, and sends the new packet to the server which resolves them to specific IP addresses according to a predefined set of rules.

#### 2.1.1 Client Implementation

The client application was developed to perform the following sequential steps:

1. **PCAP File Parsing and Filtering:** The client first reads the "5.pcap" file using the `dpkt` library. It then filters the raw packets to isolate only valid DNS query packets based on a strict set of criteria, including verifying that the packet is a UDP packet with a source or destination port of 53 and that the DNS query flag (`qr`) is set to 0.
2. **Custom Header Addition:** A unique 8-byte custom header was generated for each filtered DNS query packet. This header is formatted as `HHMMSSID`, where `HH` is the current hour in 24-hour format, `MM` is the minute, `SS` is the second, and `ID` is a two-digit sequence ID starting from 00. This custom header was then prepended to the original DNS packet.
3. **Communication with Server:** The client established a TCP connection with the server at `127.0.0.1` on port `23354`. The chosen port was made sure to be between `1024 - 49151`, so that it lies in usable ports range.
4. **Response Logging:** Upon receiving a response, the client parsed the packet to extract the resolved domain and IP address. The custom header, domain, and resolved IP were logged and stored to be included in the final report.

#### 2.1.2 Server Implementation

The server was designed to handle incoming requests as follows:

1. **Socket Initialization:** The server initialized a TCP socket and started listening for at most connections on port `23354`.
2. **Header Extraction and Rule Application:** Upon receiving a packet, the server decoded the payload to extract the custom header. It used the header's timestamp and packet ID to apply a set of predefined rules from a `rules.json` file. These rules determined which IP address from a given pool should be used for resolution.
3. **DNS Response Generation:** The server then constructed a DNS response packet. This involved modifying the original query packet by setting the response flag (`qr`) and adding an answer section containing the resolved IP address determined by the rules.
4. **Packet Transmission:** The newly generated DNS response packet was then sent back to the client.

## 2.2 Results

The client successfully processed **6 DNS query packets** from the PCAP file. The server correctly resolved each query to a designated IP address. The results of the DNS resolution are presented in the table below.

Table 2.1: DNS Queries and Resolved IP Addresses

Custom Header	Domain	Resolved IP
15290300	apple.com	192.168.1.6
15290301	facebook.com	192.168.1.7
15290302	amazon.com	192.168.1.8
15290303	twitter.com	192.168.1.9
15290304	wikipedia.org	192.168.1.10
15290305	stackoverflow.com	192.168.1.6

The following figure shows the DNS query packets sent by the client and the final DNS packet received by the client containing resolved IP in answers section of DNS packet.

```

Query DNS Packet:

[Queries]
Name: facebook.com, Type: 1, Class: 1
-----
Response DNS Packet:

[Queries]
Name: facebook.com, Type: 1, Class: 1

[Answers]
Name: facebook.com, IP: 192.168.1.7
-----

```

Figure 2.1: DNS packets before and after

The system demonstrated a successful end-to-end flow, from packet capture and filtering to custom header addition, communication, and rule-based DNS resolution. The total number of DNS query packets processed was 6.

## 3 Question 2

### 3.1 Methodology

To analyze the behavior of the `tracert` and `tracertoute` utilities on different operating systems, experiments were conducted on both Windows and Linux environments. The objective was to understand the underlying protocols used, the reasons for potential packet loss, and the differences in responses at intermediate and final hops.

The following steps were performed for both operating systems, targeting the destination `www.discord.com` (with an IP address of `162.159.135.232`):

1. **Command Execution:** The `tracert www.discord.com` command was executed on a Windows machine. Subsequently, the `tracertoute www.discord.com` command was executed on a Linux machine. Both commands were configured to trace the route with a maximum of 30 hops.
2. **Network Traffic Capture:** Wireshark was used to capture all network traffic during the execution of both commands. The captures were then analyzed to inspect the protocols, packet fields, and response types. The captures were saved and labeled for later analysis.
3. **Data Analysis:** The captured packets were examined in detail. Key fields such as the Protocol field, Time-to-Live (TTL) value, and the type of ICMP messages received were specifically analyzed to answer the posed questions.

### 3.2 Observations

#### 3.2.1 Windows

We run the `tracert` routine on a Windows machine with the destination being `discord`. We get the following output:

```
C:\Users\Arjun>tracert www.discord.com

Tracing route to www.discord.com [162.159.135.232]
over a maximum of 30 hops:

  0  3 ms    2 ms    2 ms   10.240.0.2
  1  16 ms    2 ms    3 ms   10.3.0.29
  2   9 ms    5 ms    5 ms   10.3.0.5
  3  12 ms    2 ms    2 ms   172.16.4.7
  4  15 ms    4 ms    5 ms   14.139.98.1
  5  13 ms    2 ms    4 ms   10.117.81.253
  6  *        *        *      Request timed out.
  7  *        *        *      Request timed out.
  8  *        *        *      Request timed out.
  9  22 ms    23 ms   22 ms   10.119.234.162
 10  45 ms    35 ms   35 ms   103.218.244.94
 11  41 ms    36 ms   35 ms   104.23.231.30
 12  42 ms    36 ms   35 ms   162.159.135.232

Trace complete
```

Figure 3.1: Output of running `tracert` on windows

We capture the packets using Wireshark (and filter for **ICMP**)

14206	310.350757	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1213/48388, ttl=1 (no response found!)
14197	310.354568	10.240.8.6	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14208	310.355585	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1214/48644, ttl=1 (no response found!)
14210	310.357798	10.240.8.2	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14211	310.358684	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1215/48900, ttl=1 (no response found!)
14217	310.361145	10.240.8.2	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14248	310.374536	10.240.8.2	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14315	312.120735	10.240.8.2	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14382	313.822115	10.240.8.2	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14494	316.341498	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1216/49156, ttl=2 (no response found!)
14495	316.357229	10.3.0.29	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14496	316.360555	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1217/49412, ttl=2 (no response found!)
14497	316.363384	10.3.0.29	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14498	316.364636	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1218/49668, ttl=2 (no response found!)
14499	316.368410	10.3.0.29	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14503	316.379742	10.3.0.29	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14551	317.884427	10.3.0.29	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14586	319.398122	10.3.0.29	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14644	321.915983	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1219/49924, ttl=3 (no response found!)
14645	321.925108	10.3.0.5	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14646	321.925911	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1220/50189, ttl=3 (no response found!)
14647	321.930898	10.3.0.5	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14648	321.931516	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1221/50436, ttl=3 (no response found!)
14657	321.936529	10.3.0.5	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14661	321.950411	10.3.0.5	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14689	323.460901	10.3.0.5	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14715	324.968154	10.3.0.5	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14755	327.478954	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1222/50692, ttl=4 (no response found!)
14756	327.491209	172.16.4.7	10.240.8.6	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)
14757	327.492696	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1223/50948, ttl=4 (no response found!)
14758	327.495741	10.240.8.6	10.240.8.6	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)
14759	327.495764	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1224/51204, ttl=4 (no response found!)
14768	327.498394	172.16.4.7	10.240.8.6	ICMP	134 Time-to-live exceeded (Time to live exceeded in transit)
14988	333.034271	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1225/51460, ttl=5 (no response found!)
14989	333.045747	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14990	333.050449	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1226/51716, ttl=5 (no response found!)
14991	333.054674	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14992	333.055702	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1227/51972, ttl=5 (no response found!)
14993	333.061415	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14994	333.074569	14.139.98.1	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
14993	333.061415	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14997	333.074568	14.139.98.1	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)

Figure 3.2: Wireshark Captures with packets (TTL 1 to TTL 5)

In Figure 3.2, we can see that every failure evokes 2 responses from the router where TTL expired

1. TTL exceeded in transit
2. Destination Unreachable

14998	333.090449	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1226/51716, ttl=5 (no response found!)
14991	333.054674	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14992	333.055702	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1227/51972, ttl=5 (no response found!)
14993	333.061415	14.139.98.1	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14994	333.074569	14.139.98.1	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
15012	334.586952	14.139.98.1	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
15041	336.093309	14.139.98.1	10.240.8.6	ICMP	70 Destination unreachable (Port unreachable)
15119	338.602807	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1228/52228, ttl=6 (no response found!)
15120	338.610680	10.117.81.253	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
15121	338.617659	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1229/52484, ttl=6 (no response found!)
15122	338.620393	10.117.81.253	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
15123	338.621470	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1230/52740, ttl=6 (no response found!)
15129	338.626143	10.117.81.253	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
15131	338.637903	172.16.4.7	10.240.8.6	ICMP	120 Time-to-live exceeded (Time to live exceeded in transit)
15141	338.796744	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15142	338.796744	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15145	338.876379	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15146	338.876379	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15147	338.876379	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15148	338.876379	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15154	338.956212	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15155	338.956212	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15156	338.956212	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15157	338.956212	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15163	339.037848	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15164	339.037848	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15165	339.037848	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)
15166	339.037848	10.3.0.9	10.240.8.6	ICMP	70 Redirect (Redirect for host)

Figure 3.3: Wireshark Captures with packets (TTL 5 to TTL 6)

In Figure 3.3, we see some Redirect from Host messages. This is not a property of the routine, but a property of the ISP, (which in this case happens to be IITGN SSO). This is a common optimization on campus/enterprise networks. For the sake of our analysis, we ignore these.

15482	344.151522	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1231/52996, ttl=7 (no response found!)
15592	348.047470	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1232/53252, ttl=7 (no response found!)
15698	352.047964	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1233/53508, ttl=7 (no response found!)
15823	356.049640	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1234/53764, ttl=8 (no response found!)
15937	360.054866	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1235/54020, ttl=8 (no response found!)
16033	364.054860	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1236/54276, ttl=8 (no response found!)
16216	368.048766	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1237/54532, ttl=9 (no response found!)
16357	372.061097	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1238/54788, ttl=9 (no response found!)
16486	376.046982	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1239/55044, ttl=9 (no response found!)
16551	380.047781	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1240/55300, ttl=10 (no response found!)
16552	380.070043	10.119.234.162	10.240.8.6	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
16553	380.071267	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1241/55556, ttl=10 (no response found!)
16554	380.084103	10.119.234.162	10.240.8.6	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
16556	380.095619	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request id=0x0001, seq=1242/55812, ttl=10 (no response found!)
16557	380.117865	10.119.234.162	10.240.8.6	ICMP	110 Time-to-live exceeded (Time to live exceeded in transit)
16561	380.151310	172.16.4.7	10.240.8.6	ICMP	120 Time-to-live exceeded (Time to live exceeded in transit)
16582	381.656750	172.16.4.7	10.240.8.6	ICMP	120 Time-to-live exceeded (Time to live exceeded in transit)
16588	383.162992	172.16.4.7	10.240.8.6	ICMP	120 Time-to-live exceeded (Time to live exceeded in transit)

Figure 3.4: Wireshark Captures with packets (TTL 7 to TTL 10)

In Figure 3.5, we can finally see here that, on the 13th hop, we reached the destination server. The destination server sends back an **ICMP Echo Reply** to the request we have issued.

### 3 Question 2

16756	385.665065	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1243/56068, ttl=11 (no response found!)
16757	385.710351	103.218.244.94	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
16758	385.712122	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1244/56324, ttl=11 (no response found!)
16777	385.747251	103.218.244.94	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
16778	385.748957	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1245/56580, ttl=11 (no response found!)
16779	385.754107	103.218.244.94	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
16934	391.296116	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1246/56836, ttl=12 (no response found!)
16935	391.337464	104.23.231.30	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
16936	391.338640	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1247/57092, ttl=12 (no response found!)
16941	391.374725	104.23.231.30	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
16942	391.376063	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1248/57348, ttl=12 (no response found!)
16944	391.382950	103.218.244.94	10.240.8.6	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
17069	396.534271	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1249/57604, ttl=13 (reply in 17070)
17070	396.576633	162.159.135.232	10.240.8.6	ICMP	106 Echo (ping) reply	id=0x0001, seq=1249/57604, ttl=51 (request in 17069)
17071	396.577754	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1250/57860, ttl=13 (reply in 17072)
17072	397.014378	162.159.135.232	10.240.8.6	ICMP	106 Echo (ping) reply	id=0x0001, seq=1250/57860, ttl=51 (request in 17071)
17073	397.015745	10.240.8.6	162.159.135.232	ICMP	106 Echo (ping) request	id=0x0001, seq=1251/58116, ttl=13 (reply in 17074)
17074	397.051440	162.159.135.232	10.240.8.6	ICMP	106 Echo (ping) reply	id=0x0001, seq=1251/58116, ttl=51 (request in 17073)

Figure 3.5: Wireshark Captures with packets (TTL 11 to TTL 13)

### 3.2.2 Linux

On Linux, we run the **traceroute** tool with **discord** as the destination. In the output, each line corresponds to a hop along the network path, listing the router's IP address and the round-trip times (RTT) for three probes. Some hops return **\* \* \***, meaning they did not respond, likely due to ICMP being blocked. The trace successfully reaches the Discord server at hop 13 (162.159.135.232), showing the complete path from the local machine to the destination.

```
student@student-VirtualBox:~$ traceroute www.discord.com
traceroute to www.discord.com (162.159.135.232), 30 hops max, 60 byte packets
 1 10.0.177.2 (10.0.177.2) 1.006 ms 0.838 ms 0.775 ms
 2 10.3.0.37 (10.3.0.37) 0.692 ms 0.613 ms 0.754 ms
 3 10.3.0.5 (10.3.0.5) 0.675 ms 0.944 ms 0.866 ms
 4 172.16.4.7 (172.16.4.7) 0.420 ms 0.323 ms 0.548 ms
 5 14.139.98.1 (14.139.98.1) 3.216 ms 3.145 ms 3.076 ms
 6 10.117.81.253 (10.117.81.253) 1.766 ms 0.789 ms 0.730 ms
 7 * * *
 8 * * *
 9 * * *
10 10.119.234.162 (10.119.234.162) 18.766 ms 19.254 ms 18.981 ms
11 103.218.244.94 (103.218.244.94) 35.177 ms 35.144 ms 35.062 ms
12 104.23.231.11 (104.23.231.11) 27.842 ms 104.23.231.7 (104.23.231.7) 25.732 ms 104.23.231.30 (104.23.231.30) 32.183 ms
13 162.159.135.232 (162.159.135.232) 32.360 ms 36.081 ms 36.714 ms
student@student-VirtualBox:~$
```

Figure 3.6: Output of traceroute

In figure 3.7, we can see the **Wireshark** capture of packets, generated by the Linux traceroute. The destination ports (33459, 33460, 33461, etc.) are deliberately varied for each probe. This variation allows traceroute to uniquely identify the corresponding ICMP replies for each packet.

92	2.750275524	10.0.177.49	162.159.135.232	UDP	74 38840 → 33459 Len=32
93	2.750330558	10.0.177.49	162.159.135.232	UDP	74 58396 → 33460 Len=32
94	2.750372463	10.0.177.49	162.159.135.232	UDP	74 36191 → 33461 Len=32
95	2.750414368	10.0.177.49	162.159.135.232	UDP	74 39614 → 33462 Len=32
96	2.750454038	10.0.177.49	162.159.135.232	UDP	74 34652 → 33463 Len=32
97	2.750493149	10.0.177.49	162.159.135.232	UDP	74 39083 → 33464 Len=32
98	2.750533378	10.0.177.49	162.159.135.232	UDP	74 45948 → 33465 Len=32
99	2.750622495	10.0.177.49	162.159.135.232	UDP	74 58846 → 33466 Len=32
100	2.750661886	10.0.177.49	162.159.135.232	UDP	74 45047 → 33467 Len=32

Figure 3.7: Port increments for each probe

The figure 3.8 shows that traceroute sends UDP packets from the source (10.0.177.49) to the destination (162.159.135.232). Intermediate routers respond with ICMP "Time-to-live exceeded" messages when the packet's TTL reaches zero, which traceroute uses to discover each hop. Finally, the destination host replies with ICMP "Port unreachable", confirming that the probes reached the target since no application is listening on the specified ports.



No.	Time	Source	Destination	Protocol	Length	Info
97	2.750493149	10.0.177.49	162.159.135.232	UDP	74	39083 → 33464 Len=32
98	2.750533378	10.0.177.49	162.159.135.232	UDP	74	45948 → 33465 Len=32
99	2.750622495	10.0.177.49	162.159.135.232	UDP	74	58846 → 33466 Len=32
100	2.750661886	10.0.177.49	162.159.135.232	UDP	74	45047 → 33467 Len=32
101	2.769130155	10.119.234.162	10.0.177.49	ICMP	110	Time-to-live exceeded (Time to live exceeded in trans
102	2.769327386	10.0.177.49	162.159.135.232	UDP	74	35861 → 33468 Len=32
103	2.769426002	10.119.234.162	10.0.177.49	ICMP	110	Time-to-live exceeded (Time to live exceeded in trans
104	2.769515678	10.0.177.49	162.159.135.232	UDP	74	40815 → 33469 Len=32
105	2.769658713	10.119.234.162	10.0.177.49	ICMP	110	Time-to-live exceeded (Time to live exceeded in trans
106	2.769790853	10.0.177.49	162.159.135.232	UDP	74	48671 → 33470 Len=32
107	2.778494753	104.23.231.11	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
108	2.778825241	10.0.177.49	162.159.135.232	UDP	74	57550 → 33471 Len=32
109	2.785661585	103.218.244.94	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
110	2.785668849	103.218.244.94	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
111	2.785675833	103.218.244.94	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
112	2.785804620	10.0.177.49	162.159.135.232	UDP	74	56002 → 33472 Len=32
113	2.785858538	10.0.177.49	162.159.135.232	UDP	74	47435 → 33473 Len=32
114	2.785917484	10.0.177.49	162.159.135.232	UDP	74	58969 → 33474 Len=32
115	2.795039872	104.23.231.7	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
116	2.795169498	10.0.177.49	162.159.135.232	UDP	74	35927 → 33475 Len=32
117	2.801684851	104.23.231.30	10.0.177.49	ICMP	70	Time-to-live exceeded (Time to live exceeded in trans
118	2.801880127	10.0.177.49	162.159.135.232	UDP	74	53612 → 33476 Len=32
119	2.802137981	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)
120	2.814888764	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)
121	2.818551520	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)
122	2.822508168	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)
123	2.822531914	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)
124	2.831141388	162.159.135.232	10.0.177.49	ICMP	102	Destination unreachable (Port unreachable)

```

0100 .... = Version: 4
... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0xfff5f (65375)
  Flags: 0x0000
  Fragment offset: 0
  Time to live: 13
  Protocol: UDP (17)
  Header checksum: 0xc898 [validation disabled]
  [Header checksum status: Unverified]
  Source: 10.0.177.49
  Destination: 162.159.135.232
  User Datagram Protocol, Src Port: 57550, Dst Port: 33471
  Data (32 bytes)

```

Figure 3.8: Wireshark captures UDP requests &amp; ICMP replies

### 3.3 Results

Based on the observations from the network traffic captures, the following answers were formulated:

1. What protocol does Windows `tracert` use by default, and what protocol does Linux `traceroute` use by default?

- **Windows `tracert`** uses the **Internet Control Message Protocol (ICMP)** by default. It sends a series of ICMP Echo Request packets to trace the route.
- **Linux `traceroute`** uses the **User Datagram Protocol (UDP)** by default. It sends UDP probe packets to a high, unlikely destination port number.

2. Some hops in your `traceroute` output may show `***`. Provide at least two reasons why a router might not reply

The `***`, indicating a "request timed out," occurs when a router along the path does not send a reply packet within the expected time frame. Based on the captures, two primary reasons for this are:

- **Firewall or ACL (Access Control List):** The router's firewall or ACL might be configured to drop ICMP or UDP packets to prevent denial-of-service attacks or to hide network topology. This would prevent the router from sending the necessary ICMP Time-to-live exceeded message back to the source.
- **Router Prioritization:** The router may be configured to prioritize routing functions

over responding to ICMP or UDP probe packets. During periods of high traffic or heavy load, the router might drop these probe packets to focus on critical data forwarding tasks.

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

In Linux `traceroute`, the **destination port number** of the UDP probe packets changes between successive probes. The initial probes for a specific hop start at a high port number (e.g., 33434) and increment with each subsequent probe. This mechanism, combined with the incrementing TTL, allows the utility to identify intermediate routers.

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

The response received from the final hop is fundamentally different from the responses from intermediate hops.

- **Intermediate Hops:** Intermediate routers respond with an **ICMP Time-to-live exceeded in transit** message. This is because the probe packet's TTL value reaches zero at that hop, and the router discards the packet and sends this error message back to the source.
- **Final Hop:** The final destination host responds with a different message. For Linux `traceroute` (using UDP), since the destination port is an unlikely, unlistened-to port, the host responds with an **ICMP Destination Unreachable (Port Unreachable)** message. For Windows `tracert` (using ICMP Echo), the final hop responds with an **ICMP Echo Reply**. This indicates that the probe packet successfully reached its intended destination.

5. Suppose a firewall blocks UDP traffic but allows ICMP — how would this affect the results of Linux `traceroute` vs. Windows `tracert`?

Assuming a firewall is present on the destination router only, if a firewall blocks UDP traffic but allows ICMP:

- **Windows `tracert`** would be **unaffected**. Since it relies on ICMP Echo Request packets and ICMP Time-to-live exceeded messages, its functionality would remain intact.
- **Linux `traceroute`** would be **affected**. The intermediate hops would still respond with ICMP Time-to-live exceeded messages, as this protocol is not blocked. However, the final destination host, which sends an ICMP Destination Unreachable message in response to a UDP packet, would not receive the probe packet, and thus no response would be returned. The `traceroute` would show **\*\*\***, timing out at the final hop.

## 4 Conclusion

This assignment provided valuable insights into both the design of network applications and the analysis of diagnostic tools.

The first task provided hands-on experience in building a functional client-server system for custom DNS resolution. We successfully implemented packet parsing logic to filter DNS queries from a PCAP file and designed a custom header for a timestamp-rule-based IP address resolution. The results demonstrate the working and correctness of the custom DNS resolver.

The second task, examining the behavior of `tracert` on Windows and `traceroute` on Linux, deepened our understanding of how different operating systems adopt distinct probing strategies. We understand the importance of protocol choice in this task as we examine the difference in routine behavior of `tracert` (using ICMP protocol) and `traceroute` (using the UDP protocol). Capturing and analyzing the traffic clarified why responses differ between intermediate hops and the destination, and how network policies such as UDP blocking influence results and how sometimes we might not get a response back from the relevant router.

Together, these tasks bridged the gap between theoretical networking concepts and their practical applications. They underscored the role of protocol details in shaping system behavior, highlighted the diagnostic power of tools like `traceroute`, and demonstrated the challenges posed by real-world network conditions. Overall, this work strengthened both conceptual understanding and applied proficiency in computer networking.