Computer Networks: Assignment 1

Nimitt (22110169)

Tapananshu (22110270)

Task 1: DNS Resolver

PCAP file Selection

Taking sum over three digits together:

PCAP File Calculation → `((169)+(270)) % 10 = 9` Selected `9.pcap` file.

Taking sum over individual digits:

PCAP File Calculation → `((1 + 6 + 9) + (2 + 7 + 0)) % 10 = 5` Selected `5.pcap` file.

Resolution Results:

`9.pcap`

A1 > Task1 > client_ans.txt

```
Custom Header Value | Domain Name   | Resolved IP Address
-----------------------------------------------------
13000600            | twitter.com.  | 192.168.1.6
13002601            | example.com.  | 192.168.1.7
13003302            | netflix.com.  | 192.168.1.8
13004803            | linkedin.com. | 192.168.1.9
13010104            | reddit.com.   | 192.168.1.10
13011805            | openai.com.   | 192.168.1.6
-----------------------------------------------------
```

The table in the screenshot shows the **Domain Names** present in the `9.pcap` file, the **Custom Header** and the corresponding **IP Address** resolved according to given rules.

`5.pcap`

A1 > Task1 > client_ans.txt

```
Custom Header Value | Domain Name       | Resolved IP Address
--------------------------------------------------------
17012200            | apple.com.        | 192.168.1.6
17013401            | facebook.com.     | 192.168.1.7
17015302            | amazon.com.       | 192.168.1.8
17021403            | twitter.com.      | 192.168.1.9
17022704            | wikipedia.org.    | 192.168.1.10
17024005            | stackoverflow.com.| 192.168.1.6
--------------------------------------------------------
```

The table in the screenshot shows the **Domain Names** present in the `5.pcap` file, the **Custom Header** and the corresponding **IP Address** resolved according to given rules.

Client.py Implementation

The **client.py** reads DNS query packets from a specified PCAP file. For each valid query, it generates an 8-byte custom header containing the current time and a sequence ID (`HHMMSSID`). It then prepends this header to the raw DNS data and sends to the server using a UDP socket. Finally, it waits to receive a DNS response, parses the resolved IP address from the answer section, and logs the request and response details to an output file.

Server.py Implementation

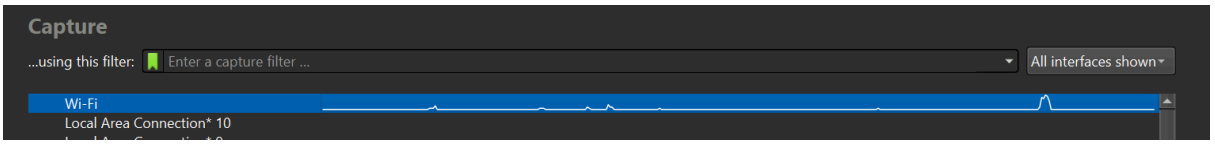
The **server.py** listens for incoming UDP traffic on a designated port. When it receives a packet from the client, it separates the 8-byte custom header from the DNS query data. It then uses a set of rules to select an IP address from a predefined pool based on the time and sequence ID found in the header. Using this selected IP, it crafts a valid DNS response packet, making sure to include the original query's transaction ID. This crafted response is then sent back to the client.

Task 2: **Traceroute Protocol Behavior**

Execution (Windows)

Step 1:

Started network traffic capture using Wireshark using WiFi interface. Closed all other application to avoid unwanted data capture.



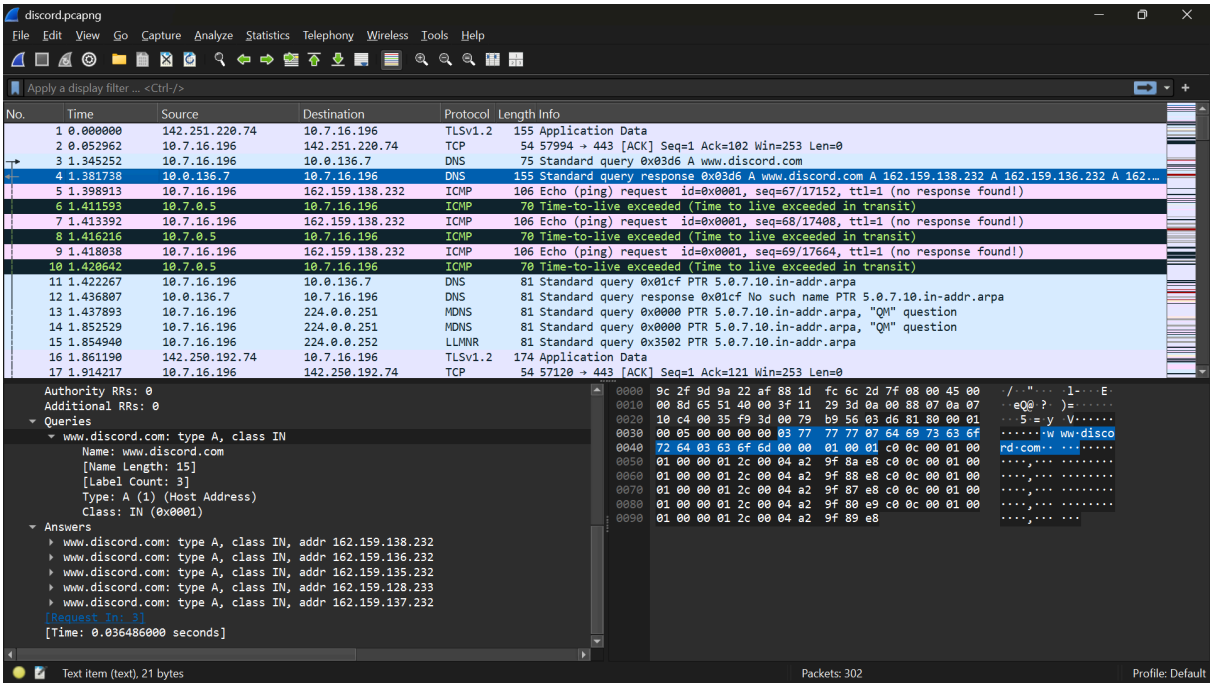
Step 2

Ran the command `tracert www.discord.com` and ended Wireshark after trace completed.

```
PS C:\Users\Tapananshu Gandhi> tracert www.discord.com

Tracing route to www.discord.com [162.159.138.232]
over a maximum of 30 hops:

  1    13 ms     3 ms     2 ms  10.7.0.5
  2    12 ms     3 ms     2 ms  172.16.4.7
  3    14 ms     5 ms     4 ms  14.139.98.1
  4    11 ms     3 ms    49 ms  10.117.81.253
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7     *        *        *     Request timed out.
  8    29 ms    40 ms    64 ms  10.119.234.162
  9    59 ms    90 ms   115 ms  103.218.244.94
 10    98 ms   101 ms   104 ms  104.23.231.11
 11    98 ms   102 ms   104 ms  162.159.138.232

Trace complete.
```

Step 3

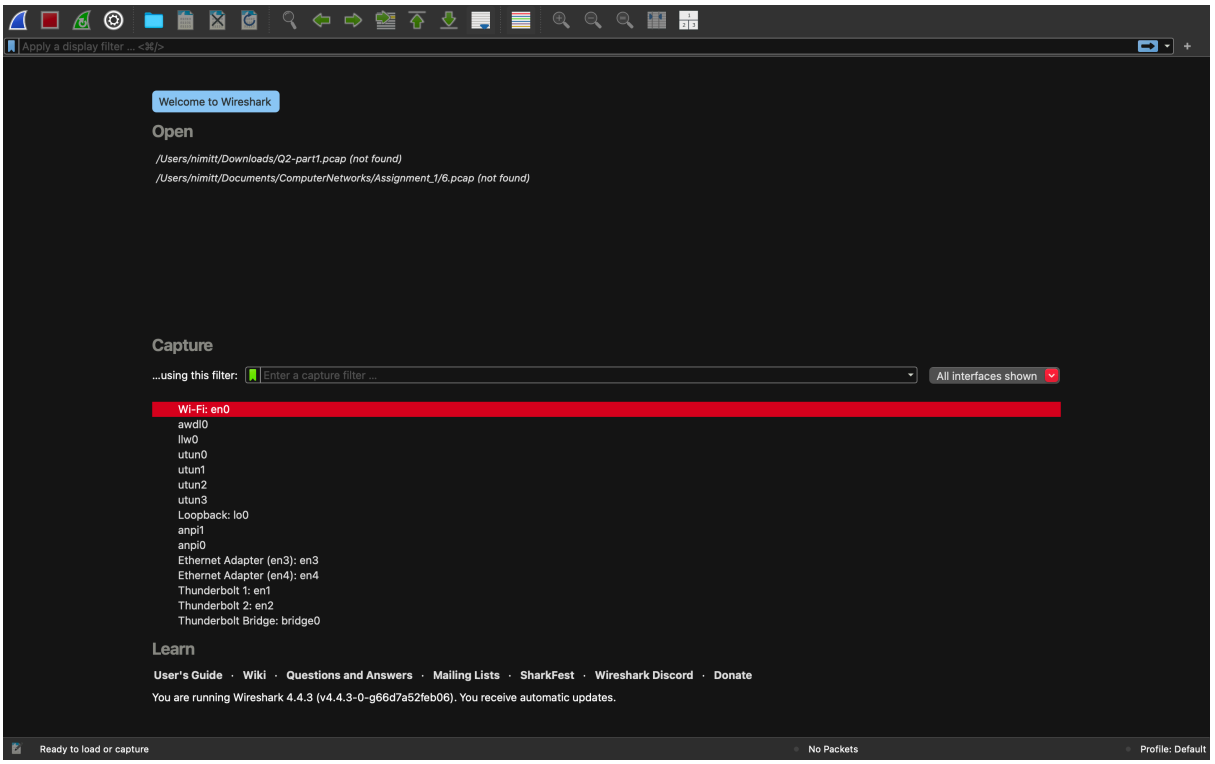
Created a '`.pcapng`' file for the captured data



Execution (Mac OS)

Step 1

Starting the Wireshark on WiFi: en0 interface, with all the other network applications closed.



Step 2

Run the traceroute on *www.discord.com*.

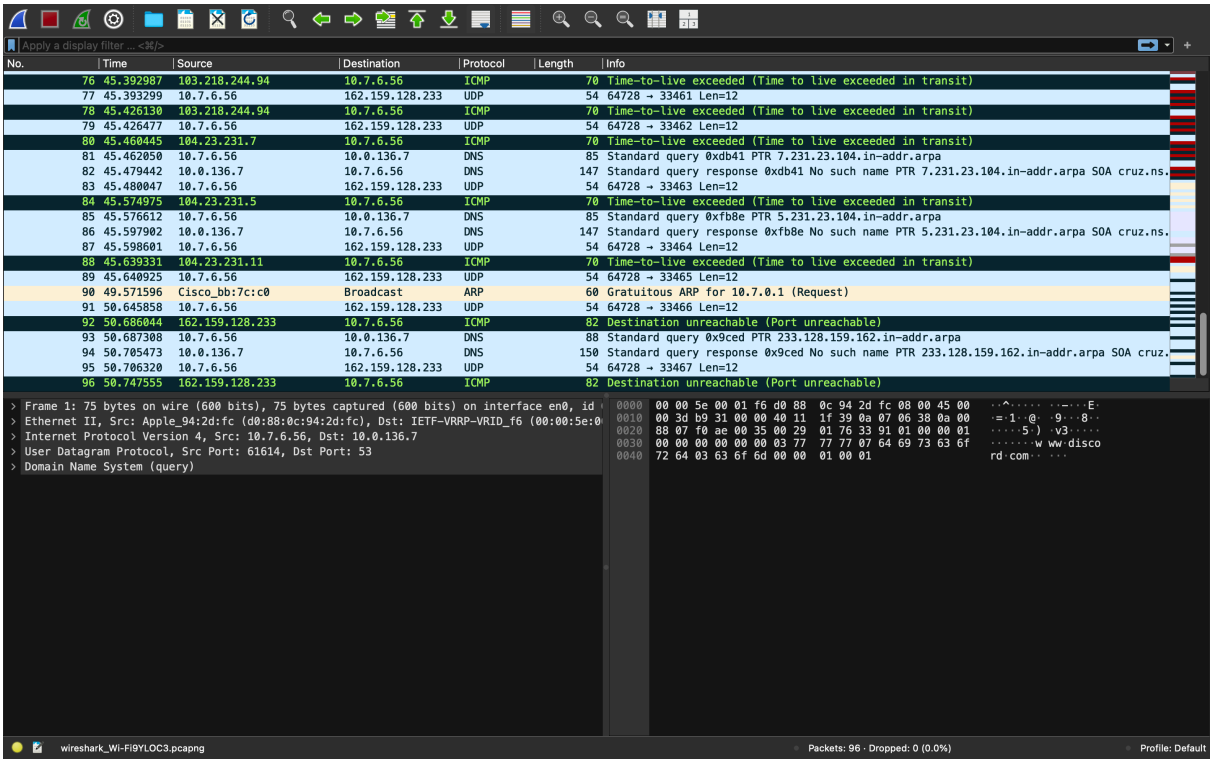
```
traceroute www.discord.com
```

Output:

```
Last login: Sat Sep 13 14:44:59 on ttys009
nimitt@Nimitts-MacBook-Air-10 ~ % traceroute www.discord.com
traceroute: Warning: www.discord.com has multiple addresses; using 162.159.128.233
traceroute to www.discord.com (162.159.128.233), 64 hops max, 40 byte packets
 1  10.7.0.5 (10.7.0.5)  5.113 ms  3.981 ms  4.293 ms
 2  172.16.4.7 (172.16.4.7)  4.236 ms  3.826 ms  5.641 ms
 3  14.139.98.1 (14.139.98.1)  5.785 ms  5.664 ms  5.133 ms
 4  10.117.81.253 (10.117.81.253)  4.431 ms  4.722 ms  4.379 ms
 5  * * *
 6  * * *
 7  * * *
 8  10.119.234.162 (10.119.234.162)  32.002 ms  32.525 ms  31.655 ms
 9  103.218.244.94 (103.218.244.94)  32.949 ms  43.339 ms  33.122 ms
10  104.23.231.7 (104.23.231.7)  34.278 ms
    104.23.231.5 (104.23.231.5)  95.187 ms
    104.23.231.11 (104.23.231.11)  41.007 ms
11  * 162.159.128.233 (162.159.128.233)  40.490 ms  41.424 ms
```

Step 3

Analyze the network traffic in Wireshark.



Step 3

Save the captured network traffic as a `.pcapng` file. Link to the file.

Results

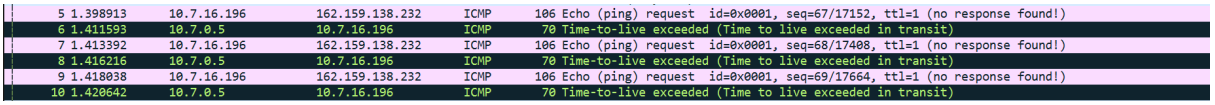
1. `traceroute` and `tracert` were able to trace the route to `www.discord.com` on MacOS and Windows respectively for intermediate 8-9 routers while failing to provide RTT for 2-3 intermediate routers.

Questions

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

tracert

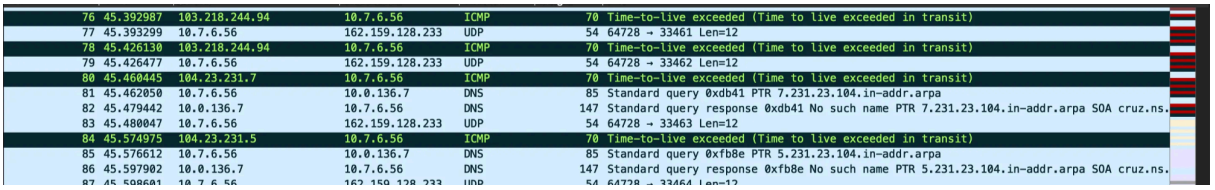
Windows `tracert` uses *Internet Control Message Protocol (**ICMP**)* which is a network protocol used by network devices to diagnose network communication issues. ICMP is mainly used to determine whether or not data is reaching its intended destination in a timely manner. The primary purpose of ICMP is for error reporting. A secondary use of ICMP protocol is to perform network diagnostics. The ICMP echo-request and echo-reply messages are commonly used for this purpose.



5	1.398913	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=67/17152, ttl=1 (no response found!)
6	1.411593	10.7.0.5	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
7	1.413392	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=68/17408, ttl=1 (no response found!)
8	1.416216	10.7.0.5	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
9	1.418038	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=69/17664, ttl=1 (no response found!)
10	1.420642	10.7.0.5	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)

traceroute

Mac/Linux `traceroute` uses *User Datagram Protocol (**UDP**)* which is a transport layer protocol.
Both ICMP and UDP are connection-less protocols. Although, the usage of ICMP packets can be enabled using the flag `-I` with `traceroute`. Following screenshot, shows the usage of **UDP (No. 77)** and router responds using **ICMP (No.78)**.



76	45.392987	103.218.244.94	10.7.6.56	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
77	45.393299	10.7.6.56	162.159.128.233	UDP	54 64728 → 33461 Len=12
78	45.426130	103.218.244.94	10.7.6.56	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
79	45.426477	10.7.6.56	162.159.128.233	UDP	54 64728 → 33462 Len=12
80	45.460445	104.23.231.7	10.7.6.56	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
81	45.462050	10.7.6.56	10.0.136.7	DNS	85 Standard query 0xdb41 PTR 7.231.23.104.in-addr.arpa
82	45.479442	10.0.136.7	10.7.6.56	DNS	147 Standard query response 0xdb41 No such name PTR 7.231.23.104.in-addr.arpa SOA cruz.ns.
83	45.480047	10.7.6.56	162.159.128.233	UDP	54 64728 → 33463 Len=12
84	45.574975	104.23.231.5	10.7.6.56	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
85	45.576612	10.7.6.56	10.0.136.7	DNS	85 Standard query 0xfb8e PTR 5.231.23.104.in-addr.arpa
86	45.597902	10.0.136.7	10.7.6.56	DNS	147 Standard query response 0xfb8e No such name PTR 5.231.23.104.in-addr.arpa SOA cruz.ns.

Both Windows/Mac receive response from the router using ICMP protocol.

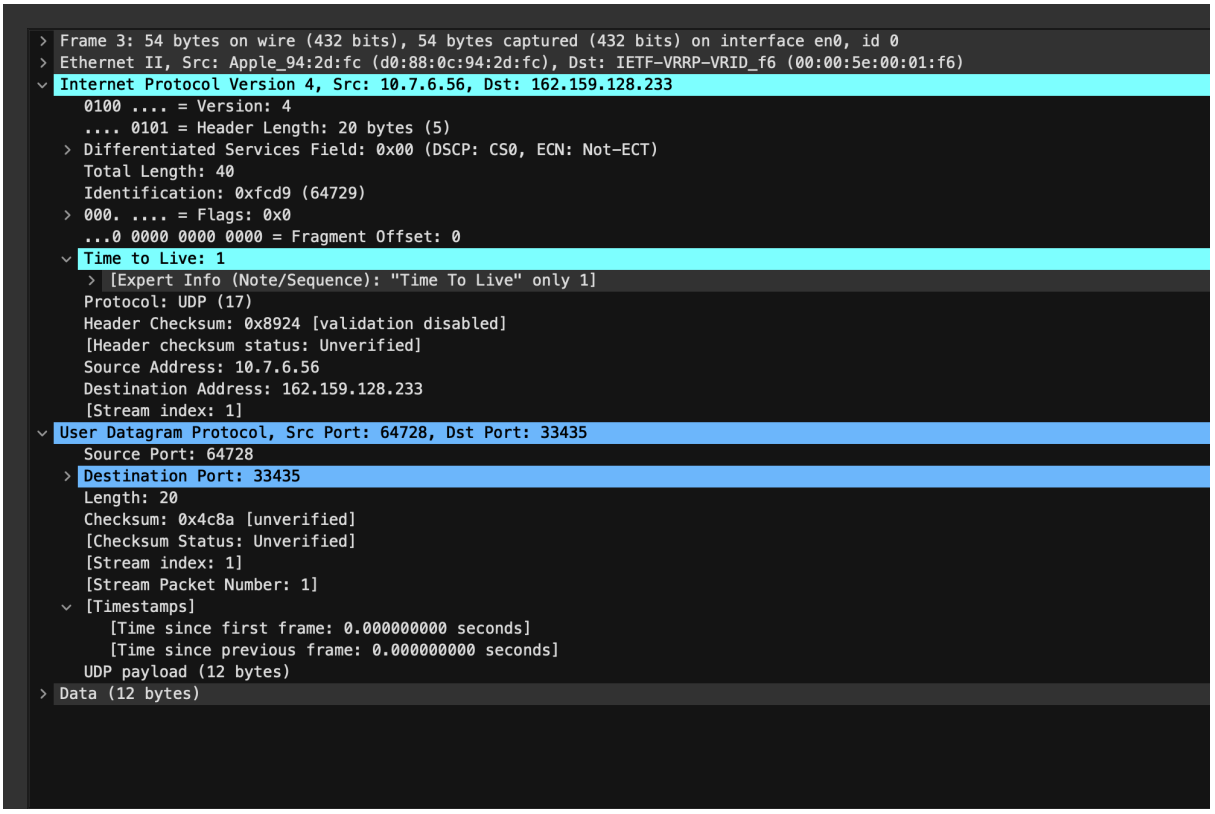
2. Some hops in you `traceroute` output may show `***`. Provide at least **two reasons.** why a router might not reply.

Reasons for why a router might not reply:

1. **Security:** Some routers may not respond to tracing queries in order to prevent outsiders from easily mapping the network's internal structure.
2. **Congestion/Overload:** In case of congestion and overload, routers may prioritize other packets/queries over tracing packets/queries.
3. **ICMP Rate Limiting:** Routers may have a limit on ICMP queries to avoid Denial of Service (DoS) attacks that use flood of ICMP packets. Breaching this limit may result in routers not replying.

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

Linux `traceroute` uses UDP instead of ICMP as in Windows. It uses the **Time to Live (TTL)** field in the IP layer which changes in successive probes by one. The following packet configuration from Wireshark shows this:



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

Intermediate Hop Response

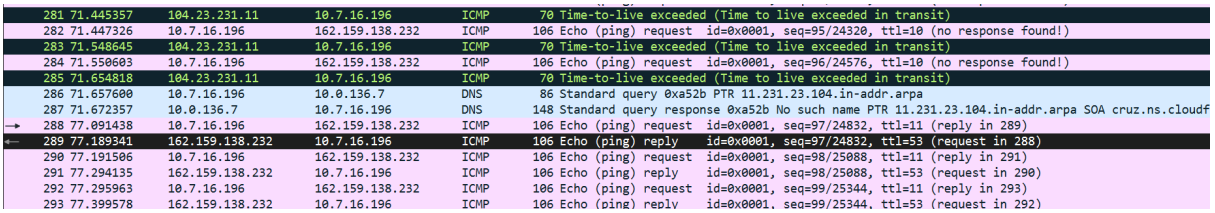
An intermediate hop, which is a router along the path, replies with an **ICMP "Time to live exceeded"** message. This occurs when a probe packet's Time-To-Live (TTL) value expires at that router. The router discards the packet and sends this message to inform the source that the packet could not continue its journey.

Final Hop Response

When a probe packet reaches the final destination, its TTL is greater than zero. The host processes the packet and sends a reply based on the protocol of the probe it received:

- **ICMP "Echo Reply":** This is sent in response to an **ICMP Echo Request** (used by Windows `tracert`). This standard "ping" reply confirms that the host is reachable.

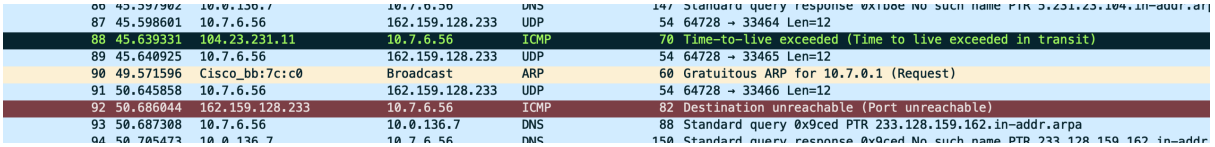
Below screenshot shows reply from one of the intermediate hops and the destination hop:



281	71.445357	104.23.231.11	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
282	71.447326	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=95/24320, ttl=10 (no response found!)
283	71.548645	104.23.231.11	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
284	71.550603	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=96/24576, ttl=10 (no response found!)
285	71.654818	104.23.231.11	10.7.16.196	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
286	71.657600	10.7.16.196	10.0.136.7	DNS	86 Standard query 0xa52b PTR 11.231.23.104.in-addr.arpa
287	71.672357	10.0.136.7	10.7.16.196	DNS	148 Standard query response 0xa52b No such name PTR 11.231.23.104.in-addr.arpa SOA cruz.ns.cloudf
288	77.091438	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=97/24832, ttl=11 (reply in 289)
289	77.189341	162.159.138.232	10.7.16.196	ICMP	106 Echo (ping) reply id=0x0001, seq=97/24832, ttl=53 (request in 288)
290	77.191506	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=98/25088, ttl=11 (reply in 291)
291	77.294135	162.159.138.232	10.7.16.196	ICMP	106 Echo (ping) reply id=0x0001, seq=98/25088, ttl=53 (request in 290)
292	77.295963	10.7.16.196	162.159.138.232	ICMP	106 Echo (ping) request id=0x0001, seq=99/25344, ttl=11 (reply in 293)
293	77.399578	162.159.138.232	10.7.16.196	ICMP	106 Echo (ping) reply id=0x0001, seq=99/25344, ttl=53 (request in 292)

- **ICMP "Destination Unreachable (Port Unreachable)":** This is sent in response to a **UDP packet** sent to a high, unused port (used by tools like Linux `traceroute`). This error message paradoxically confirms that the destination was reached, as the host's operating system is reporting that while it received the packet, no application was listening on that specific port.

The following screenshot shows thay in case of intermediate hops, response is **"Time to live exceeded"** and **"Destination Unreachable (Port Unreachable)"** for the final hop.



87	45.598601	10.7.6.56	162.159.128.233	UDP	54 64728 → 33464 Len=12
88	45.639331	104.23.231.11	10.7.6.56	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
89	45.640925	10.7.6.56	162.159.128.233	UDP	54 64728 → 33465 Len=12
90	49.571596	Cisco_bb:7c:c0	Broadcast	ARP	60 Gratuitous ARP for 10.7.0.1 (Request)
91	50.645858	10.7.6.56	162.159.128.233	UDP	54 64728 → 33466 Len=12
92	50.686044	162.159.128.233	10.7.6.56	ICMP	82 Destination unreachable (Port unreachable)
93	50.687308	10.7.6.56	10.0.136.7	DNS	88 Standard query 0x9ced PTR 233.128.159.162.in-addr.arpa

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

Windows `tracert` (Unaffected): This utility would **succeed**. Because `tracert` uses **ICMP** packets for its probes and the firewall allows ICMP, its traffic would pass through unaffected, allowing the trace to complete normally.

Linux `traceroute` (Blocked): This utility would **fail** at the firewall. By default, `traceroute` uses **UDP** packets for its probes, which the firewall is configured to block. The output would show the hops leading up to the firewall, followed by timeouts for all subsequent hops.

A Linux user could work around this by using the `-I` flag (`traceroute -I`), which forces the tool to use ICMP packets, allowing it to bypass the default UDP usage.