Computer Networks Assignment 1

-Chaitanya Attanti(23110052)
-Praneeth Pabbathi(2310226)

TASK-1

Objective

- Implement a client-server system to parse DNS queries from a PCAP file.
- Add custom headers to queries in the format **HHMMSSID.**
- Resolve DNS requests using time-based rules and IP pool on the server.
- Generate a report with Custom Header, Domain, and Resolved IP.

Implementation:

Server:

- Maintains a pool of 15 IP addresses: 192.168.1.1 - 192.168.1.15
- Resolved DNS queries using time-based routing using DNS Resolution Rules

Time Slot	time_range	IP Pool Index
Morning	04:00-11:59	**0**–4
Afternoon	12:00-19:59	**5**–9
Night	20:00-03:59	**10**–14

- Extracts custom heade**r HHMMSSID** from queries to determine time slot and sequence-based IP.
- Extracts the domain from the DNS packet and sends response: **CustomHeader|Domain|ResolvedIP.**

Server Code:

server.py M server.py/...

```python
import socket
import json

# IP Pool
IP_POOL = [
    "192.168.1.1", "192.168.1.2", "192.168.1.3", "192.168.1.4", "192.168.1.5",
    "192.168.1.6", "192.168.1.7", "192.168.1.8", "192.168.1.9", "192.168.1.10",
    "192.168.1.11", "192.168.1.12", "192.168.1.13", "192.168.1.14", "192.168.1.15"
]

SERVER_HOST = "127.0.0.1"   # Local host
SERVER_PORT = 9999


def resolve_ip(custom_header: str) -> str:
    """Applying timestamp("HHMMSSID") rules to choose an IP from the pool"""
    hour = int(custom_header[:2])   # HH first two chars
    seq_id = int(custom_header[-2:])  # ID last two chars

    # Time based Selection
    if 4 <= hour <= 11:        # Morning
        start = 0
    elif 12 <= hour <= 19:     # Afternoon
        start = 5
    else:                      # Night
        start = 10

    ip_id = start + (seq_id % 5)
    return IP_POOL[ip_id]

```

```python
def server():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((SERVER_HOST, SERVER_PORT))
    print(f"[SERVER] Listening on {SERVER_HOST}:{SERVER_PORT}...")

    while True:
        data, addr = sock.recvfrom(4096)
        try:
            # Decode the plain text message from client
            msg = data.decode()
            if "|" not in msg:
                print(f"[SERVER] Invalid message: {msg}")
                continue

            header, domain = msg.split("|", 1)
            ip = resolve_ip(header)

            # Response format
            response = f"{header}|{domain}|{ip}"
            sock.sendto(response.encode(), addr)

            print(f"[SERVER] {domain} -> {ip} (Header={header})")
        except Exception as e:
            print("[SERVER] Error:", e)


if __name__ == "__main__":
    server()
```

```
chaitanyaattanti@Chaitanyas-MacBook-Air CN_Assignment % /usr/local/bin/python3 /Users/chaitanyaattanti/Downl
oads/CN_Assignment/server.py
[SERVER] Listening on 127.0.0.1:9999...
```

Client:

- Reads DNS queries from a PCAP file(8.pcap was used in our assignment), filters UDP port 53.
- Generates custom header **HHMMSSID** using packet timestamp and query sequence number.
- Sends queries to the server and receives resolved IPs.
- Writes report.txt containing: **CustomHeader, Domain, ResolvedIP**

Client Code:

client.py client.py/ get_queries

```python
import socket
from scapy.all import PcapReader, DNS, DNSQR, UDP

SERVER_HOST = "127.0.0.1"
SERVER_PORT = 9999

def get_queries(pcap_file, limit=None):
    """Extract DNS queries (UDP dst port 53) with custom HHMMSSID header"""
    queries = []
    seq = 0
    with PcapReader(pcap_file) as pcap:
        for pkt in pcap:
            if pkt.haslayer(DNS) and pkt[DNS].qr == 0:
                if pkt.haslayer(UDP) and pkt[UDP].dport == 53:
                    dom = pkt[DNSQR].qname.decode().strip(".")
                    # HHMMSS from packet time
                    seconds=24*60*60
                    ts = int(pkt.time) % seconds
                    hh = ts // 3600
                    mm = (ts % 3600) // 60
                    ss = ts % 60
                    hdr = f"{hh:02d}{mm:02d}{ss:02d}{seq:02d}"
                    seq += 1
                    queries.append((hdr, dom))
                    if limit and seq >= limit:
                        break
    return queries
```

```python
def run_client(pcap_file):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    queries = get_queries(pcap_file)

    results = []
    for hdr, dom in queries:
        msg = f"{hdr}|{dom}"
        sock.sendto(msg.encode(), (SERVER_HOST, SERVER_PORT))
        data, _ = sock.recvfrom(4096)
        rhdr, rdom, ip = data.decode().split("|")
        print(f"[CLIENT] {rdom} -> {ip} (Header={rhdr})")
        results.append((rhdr, rdom, ip))

    # Write report
    with open("report.txt", "w") as f:
        f.write("CustomHeader\tDomain\tResolvedIP\n")
        for h, d, ip in results:
            f.write(f"{h}\t{d}\t{ip}\n")

if __name__ == "__main__":
    pcap_file = r"8.pcap"  # Set PCAP path here
    run_client(pcap_file)

```

Results and Observations:

- Successfully filtered DNS packets (port 53) from PCAP file.
- Custom header ensured traceability of each query (time + sequence).
  Server resolved domains based on time-slot rules.
- Client received and logged correct mappings.
- **Console output**: The Console Output section shows basically how the server and client interact with each other. The client sent DNS queries for the domain. For each query, the server resolved the domain to an IP address from the pool and attached a unique header ID.

```
chaitanyaattanti@Chaitanyas-MacBook-Air CN_Assignment % /usr/local/bin/python3 /Users/chaitanyaa
ttanti/Downl
oads/CN_Assignment/server.py
[SERVER] Listening on 127.0.0.1:9999...
[SERVER] github.com -> 192.168.1.6 (Header=12341600)
[SERVER] bing.com -> 192.168.1.7 (Header=12341601)
[SERVER] facebook.com -> 192.168.1.8 (Header=12341602)
[SERVER] amazon.com -> 192.168.1.9 (Header=12341603)
[SERVER] linkedin.com -> 192.168.1.10 (Header=12341604)
[SERVER] stackoverflow.com -> 192.168.1.6 (Header=12341605)
```

Report text file:

report.txt M report.txt

CustomHeader	Domain	ResolvedIP
12341600	github.com	192.168.1.6
12341601	bing.com	192.168.1.7
12341602	facebook.com	192.168.1.8
12341603	amazon.com	192.168.1.9
12341604	linkedin.com	192.168.1.10
12341605	stackoverflow.com	192.168.1.6

TASK-2

Objective:

The objective of this task is to study how the traceroute utility works in different operating systems (We used Mac and Windows) to understand the protocols used, and analyze the behavior of packets during the route discovery process. The experiment also aims to observe how intermediate routers and the final destination respond differently, and how network security configurations (e.g., firewalls) can affect traceroute.

Implementation:

SETUP:

For the operating system we used Windows and MACOS, and Wireshark as the Packet capture too.

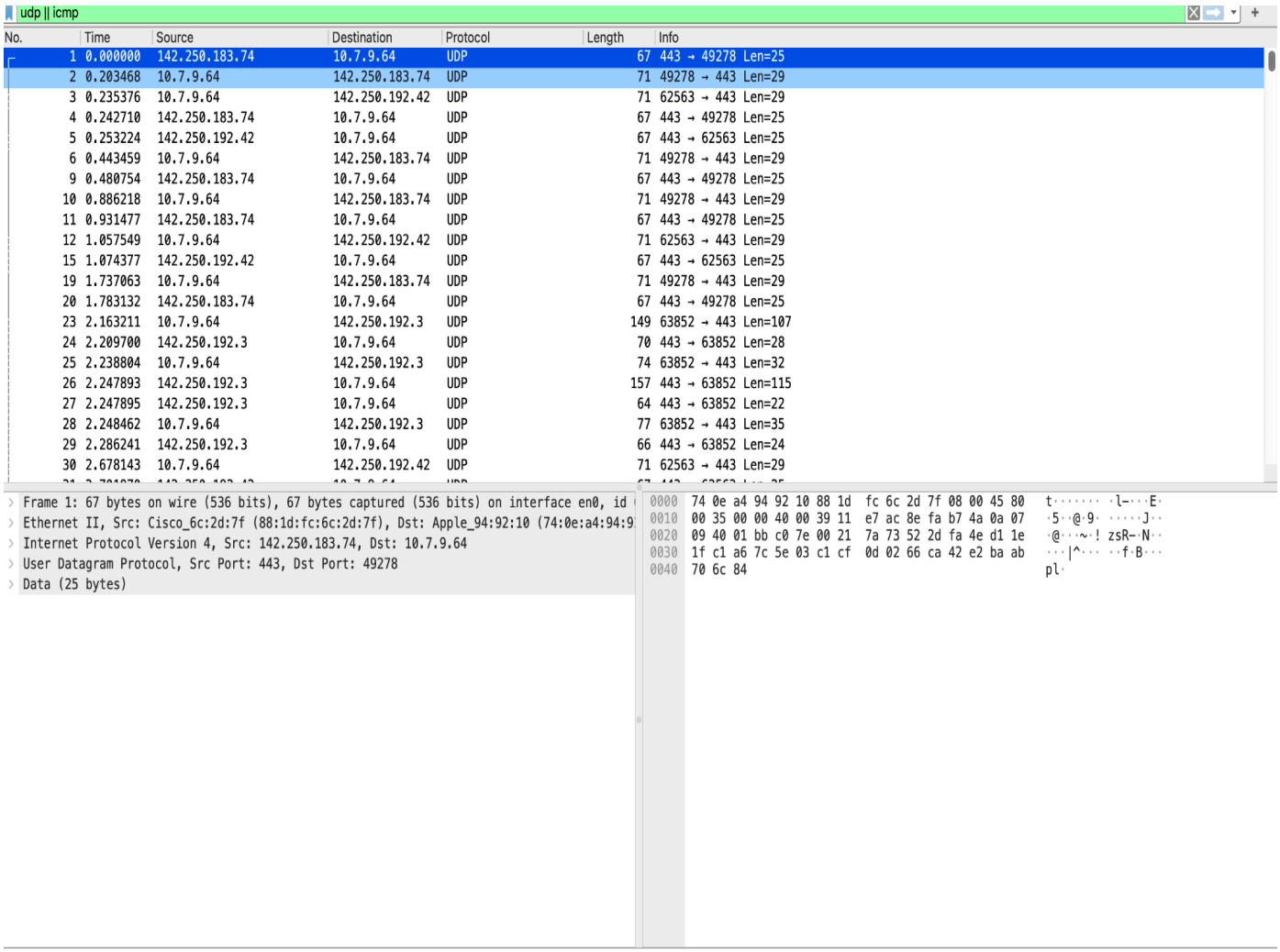
PROCEDURE:

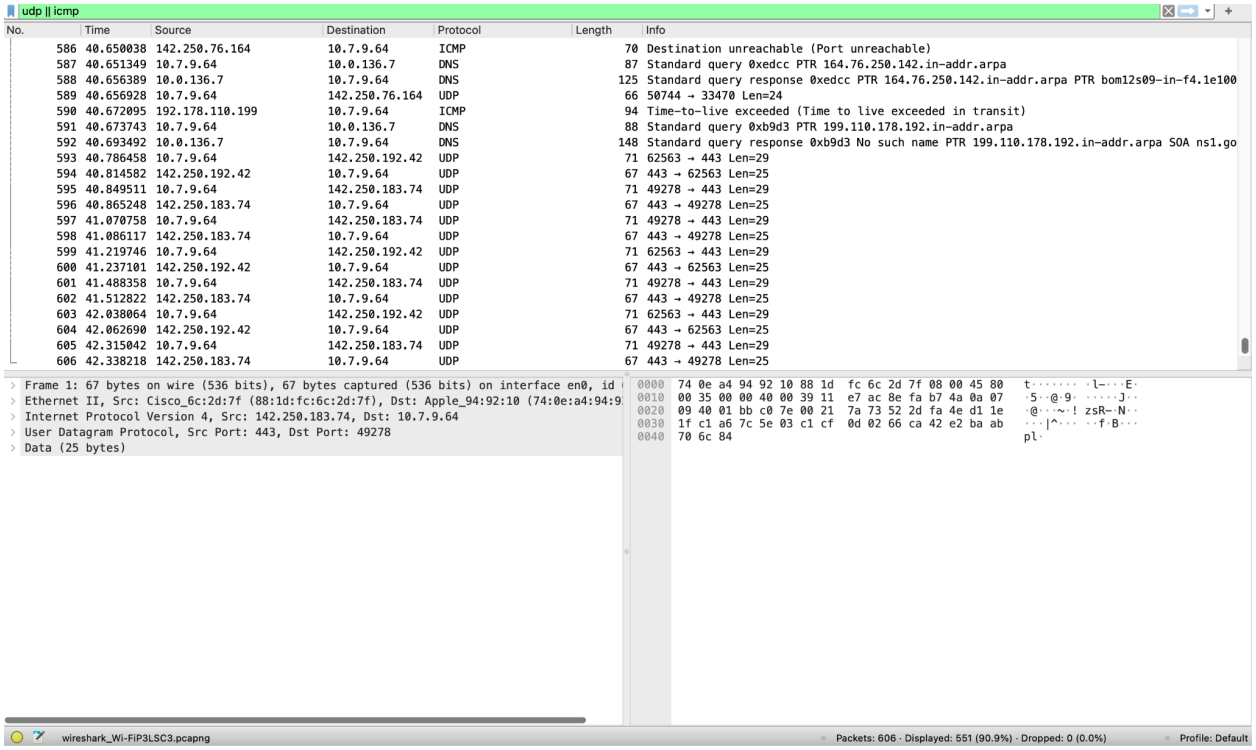
1. Ran tracert www.google.com on MAC and saved output.

```
praneethpabbathi@Praneeths-MacBook-Air ~ % traceroute www.google.com
traceroute to www.google.com (142.251.42.228), 64 hops max, 52 byte packets
 1  10.7.0.5 (10.7.0.5)  4.321 ms  4.739 ms  3.703 ms
 2  172.16.4.7 (172.16.4.7)  4.479 ms  3.530 ms  4.218 ms
 3  14.139.98.1 (14.139.98.1)  7.478 ms  5.395 ms  6.076 ms
 4  10.117.81.253 (10.117.81.253)  4.447 ms  3.799 ms  4.125 ms
 5  10.154.8.137 (10.154.8.137)  11.791 ms  12.582 ms  11.774 ms
 6  10.255.239.170 (10.255.239.170)  12.728 ms  12.045 ms  11.674 ms
 7  10.152.7.214 (10.152.7.214)  11.718 ms  11.750 ms  11.699 ms
 8  72.14.204.62 (72.14.204.62)  12.708 ms * *
 9  * * *
10  142.250.238.196 (142.250.238.196)  15.556 ms
    192.178.86.202 (192.178.86.202)  14.453 ms
    142.251.64.12 (142.251.64.12)  17.985 ms
11  142.250.214.107 (142.250.214.107)  13.992 ms
    142.250.214.109 (142.250.214.109)  13.535 ms  13.431 ms
12  192.178.110.249 (192.178.110.249)  28.646 ms
    142.250.208.227 (142.250.208.227)  14.465 ms  14.768 ms
13  142.250.214.109 (142.250.214.109)  14.734 ms
    tsa01s11-in-f4.1e100.net (142.251.42.228)  17.770 ms
    142.250.214.107 (142.250.214.107)  13.448 ms
praneethpabbathi@Praneeths-MacBook-Air ~ %
```

2. Ran traceroute again Simultaneously started Wireshark capture with protocol filters (icmp for Windows, udp || icmp for MAC).

Stopped capture after traceroute completed.Analyzed captured packets:Protocol type (ICMP/UDP).TTL values.Destination ports.ICMP reply types (Time Exceeded, Port Unreachable).





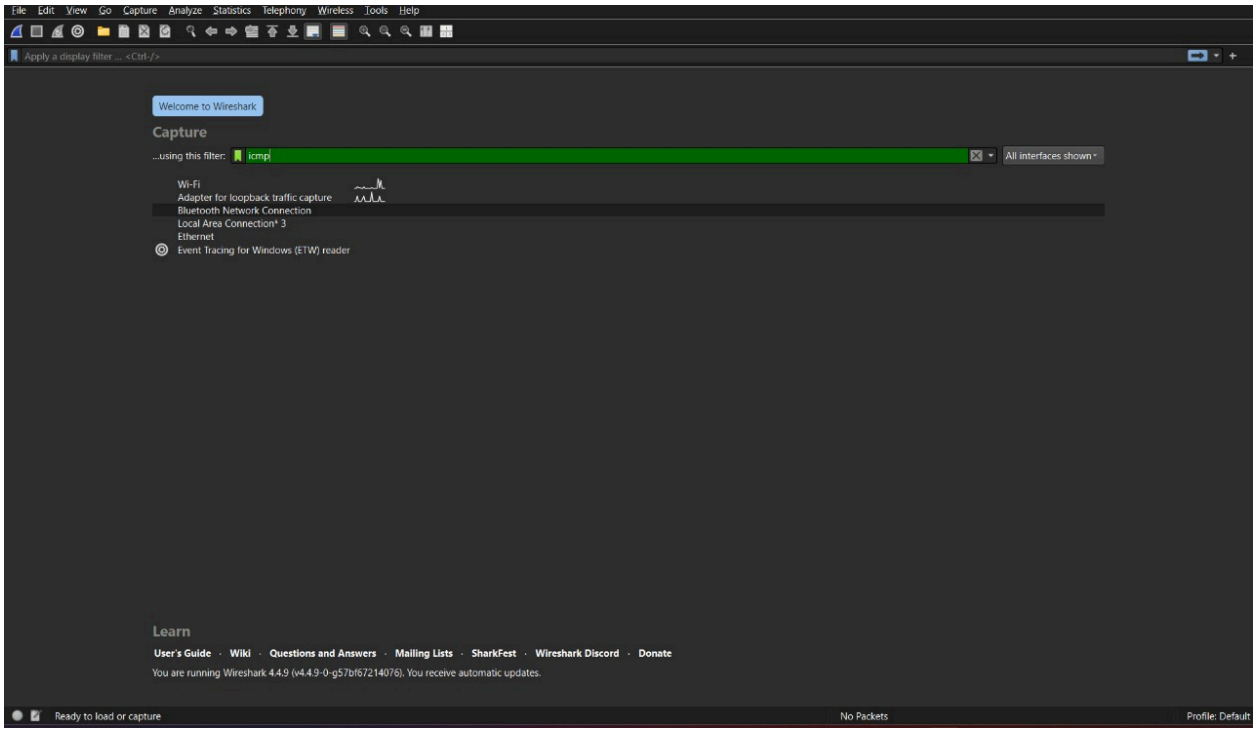

```
praneethpabbathi@Praneeths-MacBook-Air ~ % traceroute www.google.com
traceroute to www.google.com (142.250.76.164), 64 hops max, 52 byte packets
 1  10.7.0.5 (10.7.0.5)  5.022 ms  3.775 ms  4.213 ms
 2  172.16.4.7 (172.16.4.7)  4.062 ms  3.894 ms  4.190 ms
 3  14.139.98.1 (14.139.98.1)  6.287 ms  5.451 ms  5.687 ms
 4  10.117.81.253 (10.117.81.253)  4.156 ms  4.656 ms  4.094 ms
 5  10.154.8.137 (10.154.8.137)  12.611 ms  11.730 ms  12.362 ms
 6  10.255.239.170 (10.255.239.170)  12.425 ms  12.081 ms  12.999 ms
 7  10.152.7.214 (10.152.7.214)  12.752 ms  14.090 ms  13.565 ms
 8  * * *
 9  * * *
10  142.250.227.74 (142.250.227.74)  30.248 ms
    142.250.235.10 (142.250.235.10)  16.012 ms
    142.250.227.74 (142.250.227.74)  29.649 ms
11  74.125.253.165 (74.125.253.165)  14.841 ms  14.468 ms  13.715 ms
12  192.178.110.207 (192.178.110.207)  21.231 ms
    bom12s09-in-f4.1e100.net (142.250.76.164)  23.258 ms
    192.178.110.199 (192.178.110.199)  15.426 ms
praneethpabbathi@Praneeths-MacBook-Air ~ %
```

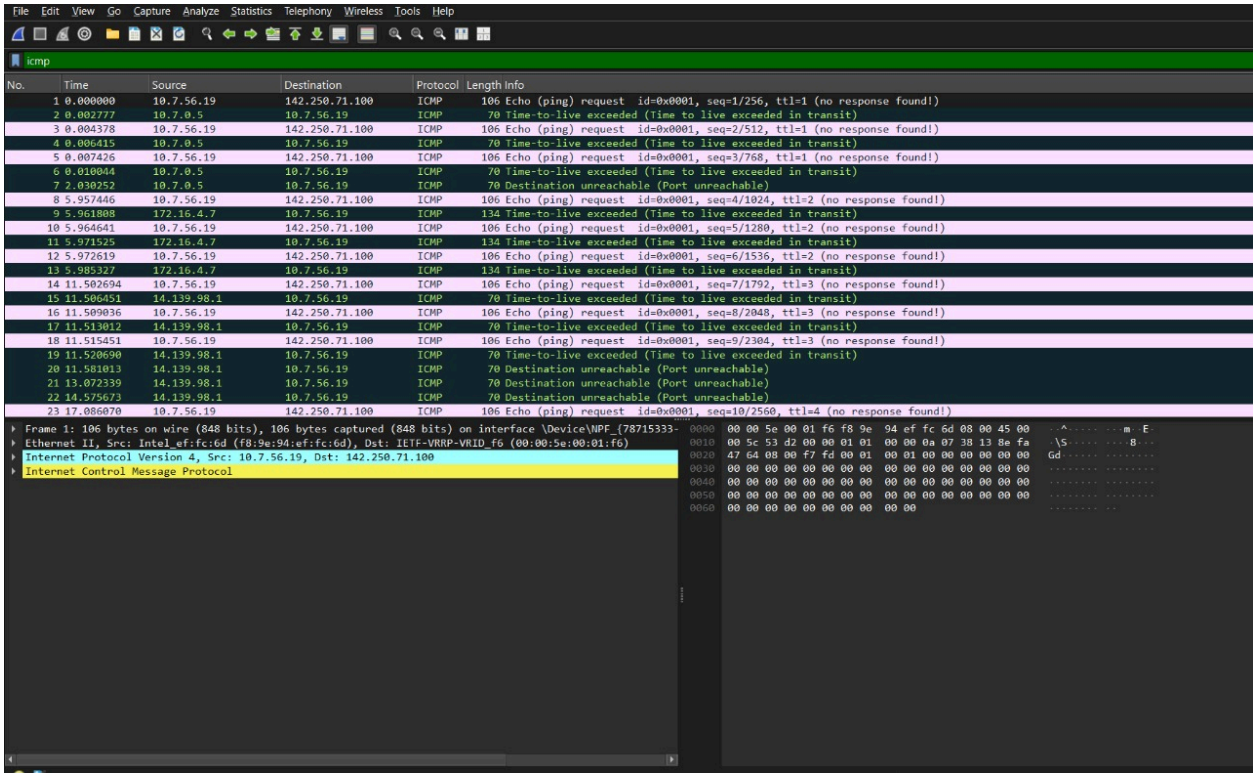
Similarly I repeated the process with **Windows**

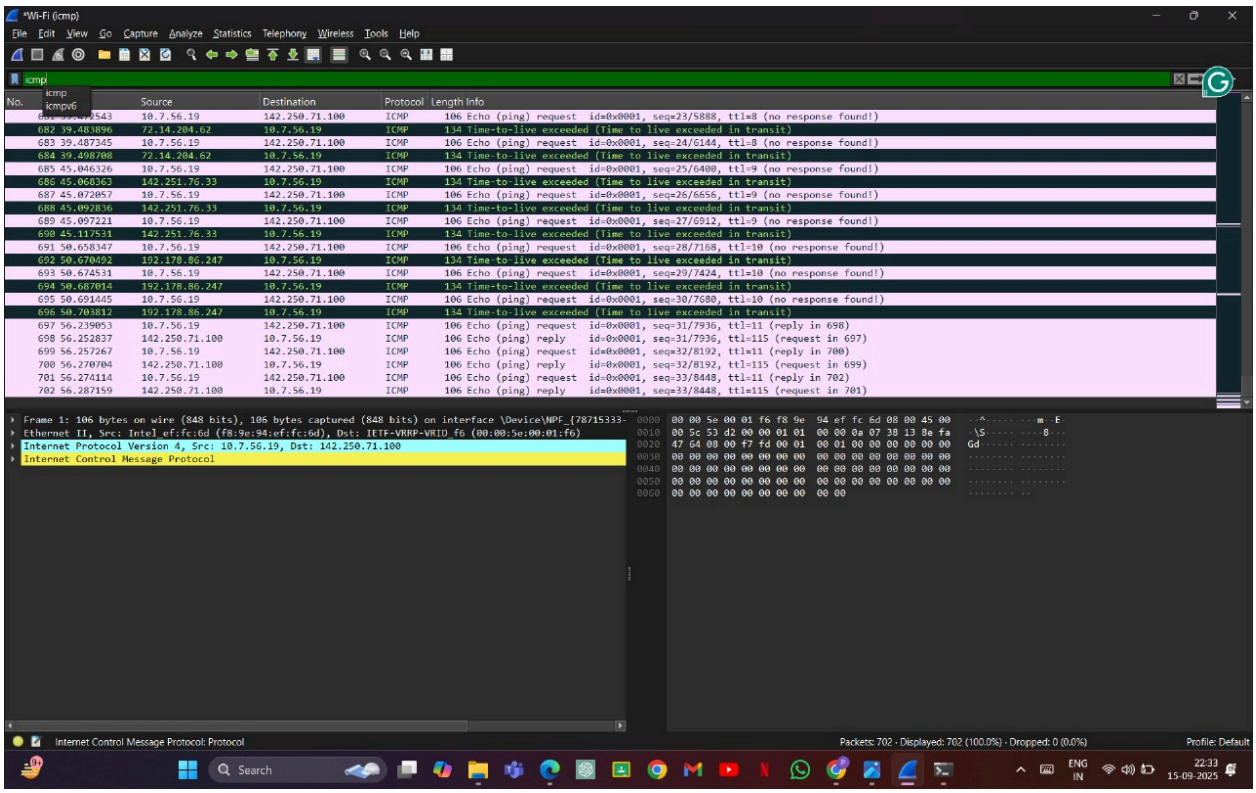
```
PS C:\Users\pilla> tracert www.google.com

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

  1     2 ms     2 ms     3 ms  10.7.0.5
  2     5 ms     3 ms     2 ms  172.16.4.7
  3     4 ms     6 ms     5 ms  14.139.98.1
  4     6 ms     3 ms     4 ms  10.117.81.253
  5    10 ms    11 ms    80 ms  10.154.8.137
  6    14 ms    11 ms    11 ms  10.255.239.170
  7    11 ms    11 ms    11 ms  10.152.7.214
  8    16 ms    13 ms    18 ms  72.14.204.62
  9    24 ms    18 ms    15 ms  72.14.239.103
 10    16 ms    14 ms    11 ms  142.251.69.105
 11    16 ms    16 ms    14 ms  bom12s21-in-f4.1e100.net [142.251.42.68]
```




File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
No. Time Source Destination Protocol Length Info
1 0.000000 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no response found!)
2 0.002777 10.7.0.5 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
3 0.004378 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=2/512, ttl=1 (no response found!)
4 0.006415 10.7.0.5 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
5 0.007426 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=3/768, ttl=1 (no response found!)
6 0.010044 10.7.0.5 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
7 2.030252 10.7.0.5 10.7.56.19 ICMP 70 Destination unreachable (Port unreachable)
8 5.957446 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=4/1024, ttl=2 (no response found!)
9 5.961808 172.16.4.7 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
10 5.964641 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=5/1280, ttl=2 (no response found!)
11 5.971525 172.16.4.7 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
12 5.972619 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=6/1536, ttl=2 (no response found!)
13 5.985327 172.16.4.7 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
14 11.502694 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=7/1792, ttl=3 (no response found!)
15 11.506451 14.139.98.1 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
16 11.509036 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=8/2048, ttl=3 (no response found!)
17 11.513012 14.139.98.1 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
18 11.515451 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=9/2304, ttl=3 (no response found!)
19 11.520690 14.139.98.1 10.7.56.19 ICMP 70 Time-to-live exceeded (Time to live exceeded in transit)
20 11.581013 14.139.98.1 10.7.56.19 ICMP 70 Destination unreachable (Port unreachable)
21 13.072339 14.139.98.1 10.7.56.19 ICMP 70 Destination unreachable (Port unreachable)
22 14.575673 14.139.98.1 10.7.56.19 ICMP 70 Destination unreachable (Port unreachable)
23 17.086070 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=10/2560, ttl=4 (no response found!)
Frame 1: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{78715333-
Ethernet II, Src: Intel_ef:fc:6d (f8:9e:94:ef:fc:6d), Dst: IETF-VRRP-VRID_f6 (00:00:5e:00:01:f6)
Internet Protocol Version 4, Src: 10.7.56.19, Dst: 142.250.71.100
Internet Control Message Protocol
0000 00 00 5e 00 01 f6 f8 9e 94 ef fc 6d 08 00 45 00
0010 00 5c 53 d2 00 00 01 01 00 00 0a 07 38 13 8e fa
0020 47 64 08 00 f7 fd 00 01 00 01 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00


*Wi-Fi (icmp)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
icmpv6
No. Source Destination Protocol Length Info
682 39.483896 72.14.204.62 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
683 39.487345 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=24/6144, ttl=8 (no response found!)
684 39.498708 72.14.204.62 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
685 45.046326 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=25/6400, ttl=9 (no response found!)
686 45.068363 142.251.76.33 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
687 45.072057 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=26/6656, ttl=9 (no response found!)
688 45.092836 142.251.76.33 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
689 45.097221 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=27/6912, ttl=9 (no response found!)
690 45.117531 142.251.76.33 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
691 50.658347 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=28/7168, ttl=10 (no response found!)
692 50.670492 192.178.86.247 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
693 50.674531 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=29/7424, ttl=10 (no response found!)
694 50.687014 192.178.86.247 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
695 50.691445 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=30/7680, ttl=10 (no response found!)
696 50.703812 192.178.86.247 10.7.56.19 ICMP 134 Time-to-live exceeded (Time to live exceeded in transit)
697 56.239053 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=31/7936, ttl=11 (reply in 698)
698 56.252837 142.250.71.100 10.7.56.19 ICMP 106 Echo (ping) reply id=0x0001, seq=31/7936, ttl=115 (request in 697)
699 56.257267 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=32/8192, ttl=11 (reply in 700)
700 56.270704 142.250.71.100 10.7.56.19 ICMP 106 Echo (ping) reply id=0x0001, seq=32/8192, ttl=115 (request in 699)
701 56.274114 10.7.56.19 142.250.71.100 ICMP 106 Echo (ping) request id=0x0001, seq=33/8448, ttl=11 (reply in 702)
702 56.287159 142.250.71.100 10.7.56.19 ICMP 106 Echo (ping) reply id=0x0001, seq=33/8448, ttl=115 (request in 701)
Frame 1: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{78715333-
Ethernet II, Src: Intel_ef:fc:6d (f8:9e:94:ef:fc:6d), Dst: IETF-VRRP-VRID_f6 (00:00:5e:00:01:f6)
Internet Protocol Version 4, Src: 10.7.56.19, Dst: 142.250.71.100
Internet Control Message Protocol
0000 00 00 5e 00 01 f6 f8 9e 94 ef fc 6d 08 00 45 00
0010 00 5c 53 d2 00 00 01 01 00 00 0a 07 38 13 8e fa
0020 47 64 08 00 f7 fd 00 01 00 01 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00
Internet Control Message Protocol: Protocol
Packets: 702 · Displayed: 702 (100.0%) · Dropped: 0 (0.0%)
Profile: Default
Search
ENG
IN
22:33
15-09-2025

```
PS C:\Users\pilla> tracert www.google.com

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

  1     2 ms     2 ms     3 ms  10.7.0.5
  2     5 ms     3 ms     2 ms  172.16.4.7
  3     4 ms     6 ms     5 ms  14.139.98.1
  4     6 ms     3 ms     4 ms  10.117.81.253
  5    10 ms    11 ms    80 ms  10.154.8.137
  6    14 ms    11 ms    11 ms  10.255.239.170
  7    11 ms    11 ms    11 ms  10.152.7.214
  8    16 ms    13 ms    18 ms  72.14.204.62
  9    24 ms    18 ms    15 ms  72.14.239.103
 10    16 ms    14 ms    11 ms  142.251.69.105
 11    16 ms    16 ms    14 ms  bom12s21-in-f4.1e100.net [142.251.42.68]

Trace complete.
PS C:\Users\pilla>
```

Answers of the Questions asked..

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

Answer:
While using **Windows**, the tracert command sends ICMP Echo Request packets by default. These are the same type of packets used by the ping command, and each router along the path returns an ICMP Time Exceeded message when the packet's (Time-To-Live) **TTL** expires.This allows Windows to map the route hop by hop.

While using **Mac**, the traceroute command typically sends UDP probe packets to high, unused destination port numbers. As the TTL is incremented, routers along the way return ICMP Time Exceeded messages, while the final destination returns a Port Unreachable error because no application is listening on that high UDP port. This is how the route is determined.

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

Answer:
Sometimes, instead of seeing a router's IP address in the output, traceroute shows * * *, which means no response was received for that probe. There are several possible reasons:

- **Firewall or ACL blocking ICMP:** The router may be configured to drop ICMP responses for security reasons, so it doesn't reply to traceroute queries.

- **Rate-limiting or de-prioritization:** Routers generally treat ICMP responses as low priority compared to forwarding actual traffic. If a router is busy, it may ignore or delay ICMP responses, resulting in missing hops.

- **Policy-based configuration:** Some ISPs and enterprise networks deliberately disable traceroute responses to prevent network reconnaissance or probing.

So, missing hops don't necessarily mean a broken connection; they may just reflect network policy.

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

Answer:
IIn Mac traceroute, the UDP destination port number changes for each successive probe, starting with a high-numbered port (e.g., 33434). Incrementing the port ensures each response is matched to the correct probe and avoids confusion when multiple probes are in-flight. When the packet reaches the destination, the host replies with an ICMP Port Unreachable message (since no service listens on that port). Additionally, the TTL also increases with each probe, which allows traceroute to reveal every intermediate hop

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

Answer:
When the probe reaches an intermediate router, the TTL (Time-To-Live) field in the packet expires before reaching the destination. That router sends back an ICMP Time Exceeded message, which traceroute uses to record the hop's IP address.

At the final hop (the destination host), the response is different because the packet is actually delivered:

- In Windows tracert (ICMP-based): the destination responds with an ICMP Echo Reply, just like in a normal ping.

- In Mac traceroute (UDP-based): the destination host sees the packet arrive on a high-numbered port where no service is listening, so it responds with an ICMP Port Unreachable message.

This difference in response is how traceroute knows it has reached the final destination.

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

Answer:

If a firewall blocks UDP traffic but still allows ICMP traffic:

- **Mac traceroute (UDP-based)** would fail. The UDP probe packets would be dropped before reaching their destination, so no ICMP responses would come back. This would make the traceroute output incomplete or entirely unresponsive.

- **Windows tracert (ICMP-based)** would still work, because it uses ICMP Echo Requests directly. Since ICMP is allowed, each router and the destination host would reply normally, and the full path could still be mapped.

This shows that the success of traceroute depends not only on the operating system but also on how the network is configured to handle ICMP and UDP traffic.