Assignment 1

Harshith Venigalla 22110283, Nakka Naga Bhuvith 22110163

Task 1:

Report: Analysis of report.csv and Code Functionality

1. CSV File Analysis (report.csv)

	A	B	C	D
	Custom Header	Domain	Resolved IP	
	3590000	_apple-mobdev._tcp.local	192.168.1.11	
	3590001	_apple-mobdev._tcp.local	192.168.1.12	
	3590002	linkedin.com	192.168.1.13	
	3590003	reddit.com	192.168.1.14	
	3590004	facebook.com	192.168.1.15	
	3590005	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.11	
	3590006	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.12	
	3590007	bing.com	192.168.1.13	
	3590008	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.14	
	3590009	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.15	
	3590010	example.com	192.168.1.11	
	3590011	_apple-mobdev._tcp.local	192.168.1.12	
	3590112	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.13	
	3590113	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.14	
	3590114	wikipedia.org	192.168.1.15	
	3590115	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.11	
	3590116	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.12	
	3590117	_apple-mobdev._tcp.local	192.168.1.13	
	3590118	_apple-mobdev._tcp.local	192.168.1.14	
	3590119	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.15	
	3590120	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.11	
	3590121	github.com	192.168.1.12	
	3590122	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.13	
	3590123	Brother MFC-7860DW._pdl-datastream._tcp.local	192.168.1.14	

The report.csv file logs the results of DNS-like queries sent from the client to the server. Each row contains:

Custom Header: A string encoding the time (HHMMSS) and a sequence ID (last two digits).

Domain: The domain name queried.

Resolved IP: The IP address assigned by the server based on the time-based rules.

Observations:

Time Slot:

All headers start with 0359, indicating the queries were generated at 03:59 (likely during the "night" slot, as per the rules).

Domains:

The domains include a mix of local service discovery names (e.g., _apple-mobdev._tcp.local, Brother MFC-7860DW._pdl-datastream._tcp.local) and public domains (e.g., linkedin.com, reddit.com, facebook.com, wikipedia.org, github.com).

IP Assignment:

The resolved IPs cycle through 192.168.1.11 to 192.168.1.15, which are the last 5 IPs in the pool. This matches the "night" rule (20:00–03:59), where the server assigns from the last 5 IPs.

Pattern:

The sequence ID (last two digits of the header) increments with each query.

The resolved IP is determined by:

index = pool_start + (seq_id % 5)

where pool_start is 10 for the night slot (so IPs 11–15).

Example:

Header 03590000 (seq_id 00) → IP: 192.168.1.11

Header 03590001 (seq_id 01) → IP: 192.168.1.12

...and so on, cycling every 5 queries.

2. Code Functionality Analysis

server.py

```python
import socket
import json
import datetime

HOST = "0.0.0.0"
PORT = 53535

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"
]

RULES = {
  "morning": {"range": (4, 11), "ip_pool_start": 0},
  "afternoon": {"range": (12, 19), "ip_pool_start": 5},
  "night": {"range": (20, 23), "ip_pool_start": 10},
}

def resolve_ip(header: str):
    """Resolve IP from header based on rules."""
    hour = int(header[:2])
    seq_id = int(header[-2:])

    # Determine time slot
    if 4 <= hour <= 11:
        pool_start = RULES["morning"]["ip_pool_start"]
    elif 12 <= hour <= 19:
        pool_start = RULES["afternoon"]["ip_pool_start"]
    else:  # 20-23 or 0-3
        pool_start = RULES["night"]["ip_pool_start"]

    index = pool_start + (seq_id % 5)
    return IP_POOL[index]

def server_main():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind((HOST, PORT))
    sock.listen(5)
    print(f"[Server] Listening on {HOST}:{PORT}")

    while True:
        conn, addr = sock.accept()
        print(f"[Server] Connection from {addr}")

        while True:
            data = conn.recv(1024)
            if not data:
                break
            header, domain = data.decode().split("|")
            ip = resolve_ip(header)
            print(f"[Server] {domain} ({header}) -> {ip}")
            conn.send(ip.encode())

        conn.close()

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

Purpose:

Acts as a custom DNS server, assigning IPs based on the time of the query.

Logic:

Listens for TCP connections on port 53535.

For each query, extracts the hour and sequence ID from the header.

Determines the time slot (morning, afternoon, night) and selects the corresponding IP pool.

Assigns an IP using the formula:

index = pool_start + (seq_id % 5)

Responds to the client with the resolved IP.

client.py

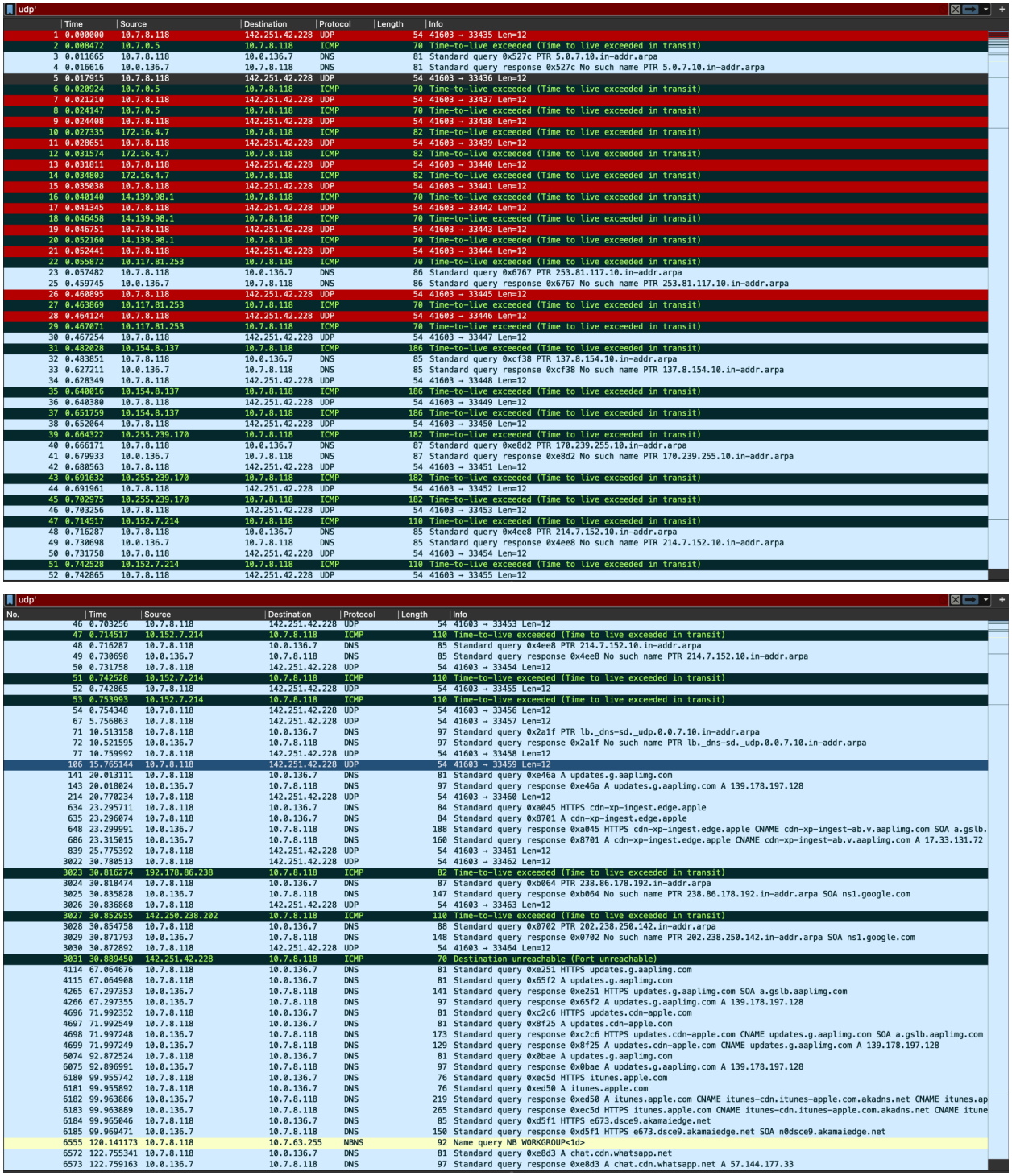
```python
# client.py
import socket
import sys
import datetime
import csv
from scapy.all import rdpcap, DNS

SERVER_IP = "10.240.17.208"
SERVER_PORT = 53535

def generate_custom_header(seq_id: int) -> str:
    """Generate HHMMSSID format timestamp + sequence ID."""
    now = datetime.datetime.now()
    hh = now.strftime("%H")
    mm = now.strftime("%M")
    ss = now.strftime("%S")
    return f"{hh}{mm}{ss}{seq_id:02d}"

def extract_dns_queries(pcap_file: str):
    """Return list of (header, domain) from PCAP DNS queries."""
    packets = rdpcap(pcap_file)
    queries = []
    seq_id = 0
    for pkt in packets:
        if pkt.haslayer(DNS) and pkt[DNS].qr == 0:  # query only
            dns_layer = pkt[DNS]
            domain = dns_layer.qd.qname.decode().strip(".")
            header = generate_custom_header(seq_id)
            queries.append((header, domain))
            seq_id += 1
    return queries

def client_main(pcap_file):
    queries = extract_dns_queries(pcap_file)

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((SERVER_IP, SERVER_PORT))

    report = []
    for header, domain in queries:
        msg = f"{header}|{domain}"
        sock.send(msg.encode())
        response = sock.recv(1024).decode()
        report.append((header, domain, response))
        print(f"[Client] {domain} ({header}) -> {response}")

    sock.close()

    # Save report into CSV
    with open("report.csv", "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Custom Header", "Domain", "Resolved IP"])
        writer.writerows(report)

    print("\n=== Report Saved to report.csv ===")
    print("{:<10} {:<25} {:<15}".format("Header", "Domain", "Resolved IP"))
    for h, d, ip in report:
        print("{:<10} {:<25} {:<15}".format(h, d, ip))

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python client.py <pcap_file>")
        sys.exit(1)
    client_main(sys.argv[1])
```

Purpose:
Reads DNS queries from a PCAP file and sends them to the server.
Logic:
Uses Scapy to extract DNS queries from the PCAP file.
For each query, generates a custom header (current time + sequence ID).
Sends the query to the server and receives the resolved IP.
Logs the results to report.csv.
3. Conclusion
The system correctly implements time-based routing of DNS queries, as evidenced by the IP assignment pattern in report.csv.
The code is modular and follows the logic described in the README and rules.json.
The CSV report demonstrates that the server's rules are being applied as intended, with all queries during the "night" slot being assigned IPs from the last 5 in the pool.

Task 2:

The screenshot below shows a PCAP file in a Linux system:

The screenshot below shows a PCAP file in a Mac system:



udp'

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.7.8.118	142.251.42.228	UDP	54	41603 → 33435 Len=12
2	0.008472	10.7.0.5	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
3	0.011665	10.7.8.118	10.0.136.7	DNS	81	Standard query 0x527c PTR 5.0.7.10.in-addr.arpa
4	0.016616	10.0.136.7	10.7.8.118	DNS	81	Standard query response 0x527c No such name PTR 5.0.7.10.in-addr.arpa
5	0.017915	10.7.8.118	142.251.42.228	UDP	54	41603 → 33436 Len=12
6	0.020924	10.7.0.5	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	0.021210	10.7.8.118	142.251.42.228	UDP	54	41603 → 33437 Len=12
8	0.024147	10.7.0.5	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	0.024408	10.7.8.118	142.251.42.228	UDP	54	41603 → 33438 Len=12
10	0.027335	172.16.4.7	10.7.8.118	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
11	0.028651	10.7.8.118	142.251.42.228	UDP	54	41603 → 33439 Len=12
12	0.031574	172.16.4.7	10.7.8.118	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
13	0.031811	10.7.8.118	142.251.42.228	UDP	54	41603 → 33440 Len=12
14	0.034803	172.16.4.7	10.7.8.118	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
15	0.035038	10.7.8.118	142.251.42.228	UDP	54	41603 → 33441 Len=12
16	0.040140	14.139.98.1	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	0.041345	10.7.8.118	142.251.42.228	UDP	54	41603 → 33442 Len=12
18	0.046458	14.139.98.1	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	0.046751	10.7.8.118	142.251.42.228	UDP	54	41603 → 33443 Len=12
20	0.052160	14.139.98.1	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.052441	10.7.8.118	142.251.42.228	UDP	54	41603 → 33444 Len=12
22	0.055872	10.117.81.253	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	0.057482	10.7.8.118	10.0.136.7	DNS	86	Standard query 0x6767 PTR 253.81.117.10.in-addr.arpa
25	0.459745	10.0.136.7	10.7.8.118	DNS	86	Standard query response 0x6767 No such name PTR 253.81.117.10.in-addr.arpa
26	0.460895	10.7.8.118	142.251.42.228	UDP	54	41603 → 33445 Len=12
27	0.463869	10.117.81.253	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
28	0.464124	10.7.8.118	142.251.42.228	UDP	54	41603 → 33446 Len=12
29	0.467071	10.117.81.253	10.7.8.118	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
30	0.467254	10.7.8.118	142.251.42.228	UDP	54	41603 → 33447 Len=12
31	0.482028	10.154.8.137	10.7.8.118	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
32	0.483851	10.7.8.118	10.0.136.7	DNS	85	Standard query 0xcf38 PTR 137.8.154.10.in-addr.arpa
33	0.627211	10.0.136.7	10.7.8.118	DNS	85	Standard query response 0xcf38 No such name PTR 137.8.154.10.in-addr.arpa
34	0.628349	10.7.8.118	142.251.42.228	UDP	54	41603 → 33448 Len=12
35	0.640016	10.154.8.137	10.7.8.118	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
36	0.640380	10.7.8.118	142.251.42.228	UDP	54	41603 → 33449 Len=12
37	0.651759	10.154.8.137	10.7.8.118	ICMP	186	Time-to-live exceeded (Time to live exceeded in transit)
38	0.652064	10.7.8.118	142.251.42.228	UDP	54	41603 → 33450 Len=12
39	0.664322	10.255.239.170	10.7.8.118	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
40	0.666171	10.7.8.118	10.0.136.7	DNS	87	Standard query 0xe8d2 PTR 170.239.255.10.in-addr.arpa
41	0.679933	10.0.136.7	10.7.8.118	DNS	87	Standard query response 0xe8d2 No such name PTR 170.239.255.10.in-addr.arpa
42	0.680563	10.7.8.118	142.251.42.228	UDP	54	41603 → 33451 Len=12
43	0.691632	10.255.239.170	10.7.8.118	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
44	0.691961	10.7.8.118	142.251.42.228	UDP	54	41603 → 33452 Len=12
45	0.702975	10.255.239.170	10.7.8.118	ICMP	182	Time-to-live exceeded (Time to live exceeded in transit)
46	0.703256	10.7.8.118	142.251.42.228	UDP	54	41603 → 33453 Len=12
47	0.714517	10.152.7.214	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
48	0.716287	10.7.8.118	10.0.136.7	DNS	85	Standard query 0x4ee8 PTR 214.7.152.10.in-addr.arpa
49	0.730698	10.0.136.7	10.7.8.118	DNS	85	Standard query response 0x4ee8 No such name PTR 214.7.152.10.in-addr.arpa
50	0.731758	10.7.8.118	142.251.42.228	UDP	54	41603 → 33454 Len=12
51	0.742528	10.152.7.214	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
52	0.742865	10.7.8.118	142.251.42.228	UDP	54	41603 → 33455 Len=12

udp'

No.	Time	Source	Destination	Protocol	Length	Info
46	0.703256	10.7.8.118	142.251.42.228	UDP	54	41603 → 33453 Len=12
47	0.714517	10.152.7.214	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
48	0.716287	10.7.8.118	10.0.136.7	DNS	85	Standard query 0x4ee8 PTR 214.7.152.10.in-addr.arpa
49	0.730698	10.0.136.7	10.7.8.118	DNS	85	Standard query response 0x4ee8 No such name PTR 214.7.152.10.in-addr.arpa
50	0.731758	10.7.8.118	142.251.42.228	UDP	54	41603 → 33454 Len=12
51	0.742528	10.152.7.214	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
52	0.742865	10.7.8.118	142.251.42.228	UDP	54	41603 → 33455 Len=12
53	0.753993	10.152.7.214	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
54	0.754348	10.7.8.118	142.251.42.228	UDP	54	41603 → 33456 Len=12
67	5.756863	10.7.8.118	142.251.42.228	UDP	54	41603 → 33457 Len=12
71	10.513158	10.7.8.118	10.0.136.7	DNS	97	Standard query 0x2a1f PTR lb._dns-sd._udp.0.0.7.10.in-addr.arpa
72	10.521595	10.0.136.7	10.7.8.118	DNS	97	Standard query response 0x2a1f No such name PTR lb._dns-sd._udp.0.0.7.10.in-addr.arpa
77	10.759992	10.7.8.118	142.251.42.228	UDP	54	41603 → 33458 Len=12
106	15.765144	10.7.8.118	142.251.42.228	UDP	54	41603 → 33459 Len=12
141	20.013111	10.7.8.118	10.0.136.7	DNS	81	Standard query 0xe46a A updates.g.aaplimg.com
143	20.018024	10.0.136.7	10.7.8.118	DNS	97	Standard query response 0xe46a A updates.g.aaplimg.com A 139.178.197.128
214	20.770234	10.7.8.118	142.251.42.228	UDP	54	41603 → 33460 Len=12
634	23.295711	10.7.8.118	10.0.136.7	DNS	84	Standard query 0xa045 HTTPS cdn-xp-ingest.edge.apple
635	23.296074	10.7.8.118	10.0.136.7	DNS	84	Standard query 0x8701 A cdn-xp-ingest.edge.apple
648	23.299991	10.0.136.7	10.7.8.118	DNS	188	Standard query response 0xa045 HTTPS cdn-xp-ingest.edge.apple CNAME cdn-xp-ingest-ab.v.aaplimg.com SOA a.gslb.
686	23.315015	10.0.136.7	10.7.8.118	DNS	160	Standard query response 0x8701 A cdn-xp-ingest.edge.apple CNAME cdn-xp-ingest-ab.v.aaplimg.com A 17.33.131.72
839	25.775392	10.7.8.118	142.251.42.228	UDP	54	41603 → 33461 Len=12
3022	30.780513	10.7.8.118	142.251.42.228	UDP	54	41603 → 33462 Len=12
3023	30.816274	192.178.86.238	10.7.8.118	ICMP	82	Time-to-live exceeded (Time to live exceeded in transit)
3024	30.818474	10.7.8.118	10.0.136.7	DNS	87	Standard query 0xb064 PTR 238.86.178.192.in-addr.arpa
3025	30.835828	10.0.136.7	10.7.8.118	DNS	147	Standard query response 0xb064 No such name PTR 238.86.178.192.in-addr.arpa SOA ns1.google.com
3026	30.836868	10.7.8.118	142.251.42.228	UDP	54	41603 → 33463 Len=12
3027	30.852955	142.250.238.202	10.7.8.118	ICMP	110	Time-to-live exceeded (Time to live exceeded in transit)
3028	30.854758	10.7.8.118	10.0.136.7	DNS	88	Standard query 0x0702 PTR 202.238.250.142.in-addr.arpa
3029	30.871793	10.0.136.7	10.7.8.118	DNS	148	Standard query response 0x0702 No such name PTR 202.238.250.142.in-addr.arpa SOA ns1.google.com
3030	30.872892	10.7.8.118	142.251.42.228	UDP	54	41603 → 33464 Len=12
3031	30.889450	142.251.42.228	10.7.8.118	ICMP	70	Destination unreachable (Port unreachable)
4114	67.064676	10.7.8.118	10.0.136.7	DNS	81	Standard query 0xe251 HTTPS updates.g.aaplimg.com
4115	67.064908	10.7.8.118	10.0.136.7	DNS	81	Standard query 0x65f2 A updates.g.aaplimg.com
4265	67.297353	10.0.136.7	10.7.8.118	DNS	141	Standard query response 0xe251 HTTPS updates.g.aaplimg.com SOA a.gslb.aaplimg.com
4266	67.297355	10.0.136.7	10.7.8.118	DNS	97	Standard query response 0x65f2 A updates.g.aaplimg.com A 139.178.197.128
4696	71.992352	10.7.8.118	10.0.136.7	DNS	81	Standard query 0xc2c6 HTTPS updates.cdn-apple.com
4697	71.992549	10.7.8.118	10.0.136.7	DNS	81	Standard query 0x8f25 A updates.cdn-apple.com
4698	71.997248	10.0.136.7	10.7.8.118	DNS	173	Standard query response 0xc2c6 HTTPS updates.cdn-apple.com CNAME updates.g.aaplimg.com SOA a.gslb.aaplimg.com
4699	71.997249	10.0.136.7	10.7.8.118	DNS	129	Standard query response 0x8f25 A updates.cdn-apple.com CNAME updates.g.aaplimg.com A 139.178.197.128
6074	92.872524	10.7.8.118	10.0.136.7	DNS	81	Standard query 0x0bae A updates.g.aaplimg.com
6075	92.896991	10.0.136.7	10.7.8.118	DNS	97	Standard query response 0x0bae A updates.g.aaplimg.com A 139.178.197.128
6180	99.955742	10.7.8.118	10.0.136.7	DNS	76	Standard query 0xec5d HTTPS itunes.apple.com
6181	99.955892	10.7.8.118	10.0.136.7	DNS	76	Standard query 0xed50 A itunes.apple.com
6182	99.963886	10.0.136.7	10.7.8.118	DNS	219	Standard query response 0xed50 A itunes.apple.com CNAME itunes-cdn.itunes-apple.com.akadns.net CNAME itunes.ap
6183	99.963889	10.0.136.7	10.7.8.118	DNS	265	Standard query response 0xec5d HTTPS itunes.apple.com CNAME itunes-cdn.itunes-apple.com.akadns.net CNAME itune
6184	99.965046	10.7.8.118	10.0.136.7	DNS	85	Standard query 0xd5f1 HTTPS e673.dsce9.akamaiedge.net
6185	99.969471	10.0.136.7	10.7.8.118	DNS	150	Standard query response 0xd5f1 HTTPS e673.dsce9.akamaiedge.net SOA n0dsce9.akamaiedge.net
6555	120.141173	10.7.8.118	10.7.63.255	NBNS	92	Name query NB WORKGROUP<1d>
6572	122.755341	10.7.8.118	10.0.136.7	DNS	81	Standard query 0xe8d3 A chat.cdn.whatsapp.net
6573	122.759163	10.0.136.7	10.7.8.118	DNS	97	Standard query response 0xe8d3 A chat.cdn.whatsapp.net A 57.144.177.33

Q1. What protocol does Windows tracert use by default, and what protocol does Linux traceroute use by default?

- Windows tracert uses ICMP Echo Requests by default.

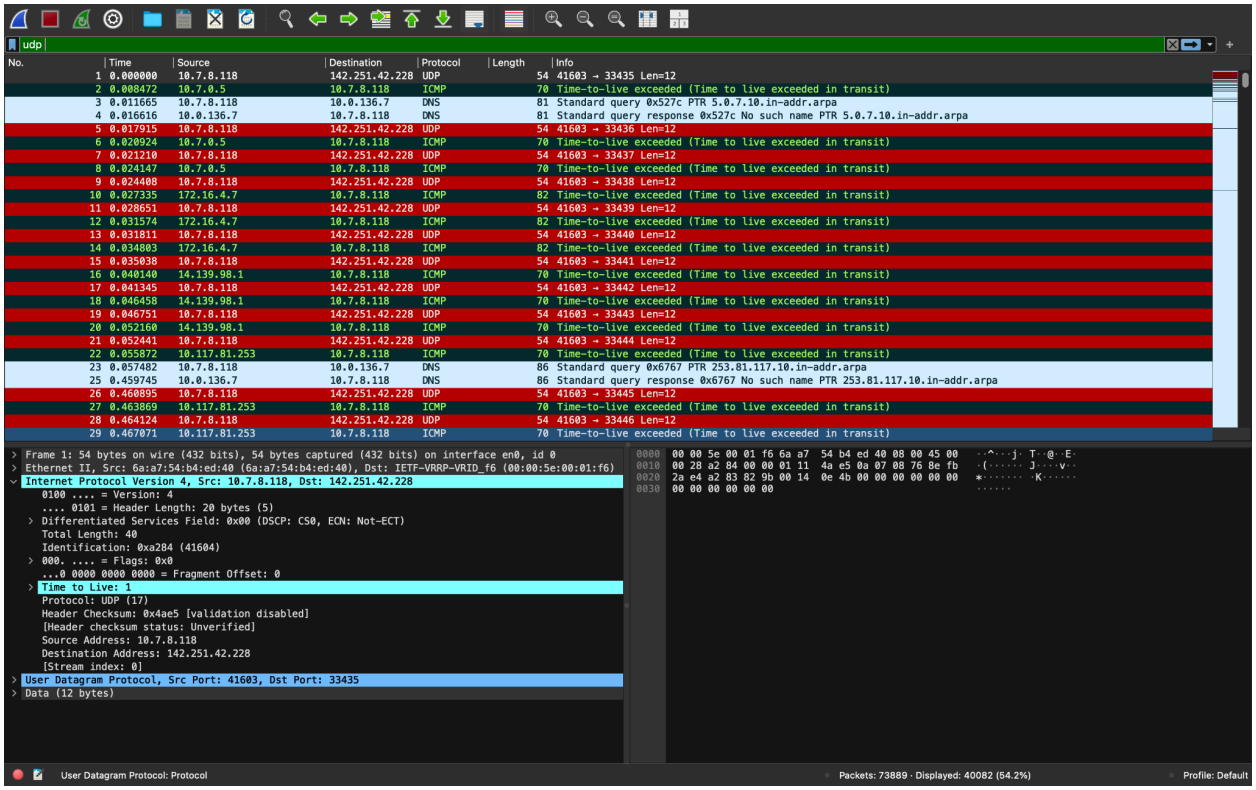
- Linux traceroute uses UDP probes to high-numbered ports (starting from 33434) by default and ICMP as a response from the route.

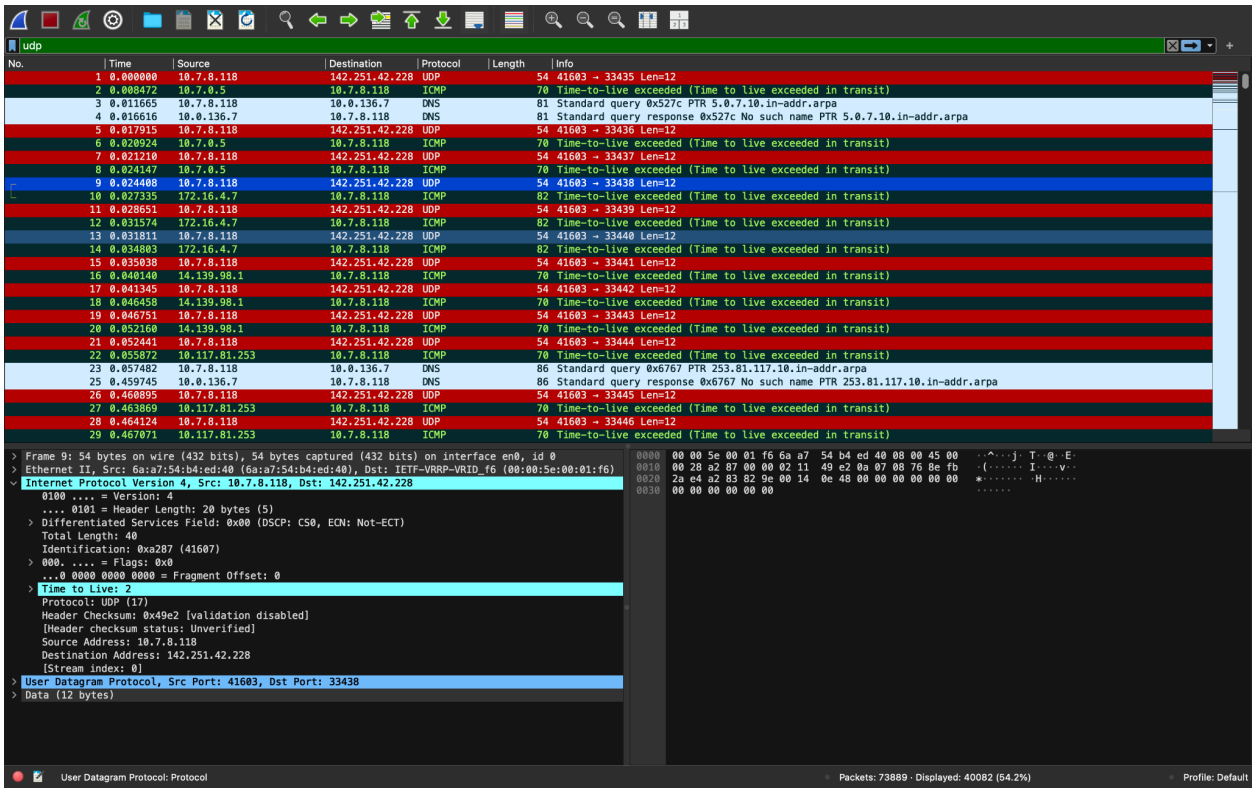
Q2. Some hops in your traceroute output may show ***. Provide at least two reasons why a router might not reply.

```
(base) harshithvenigalla@harshiths-MacBook-Pro-8 ~ % traceroute www.google.com
traceroute to www.google.com (142.251.42.228), 64 hops max, 40 byte packets
 1  10.7.0.5 (10.7.0.5)  9.334 ms  3.290 ms  3.184 ms
 2  172.16.4.7 (172.16.4.7)  3.162 ms  3.138 ms  3.199 ms
 3  14.139.98.1 (14.139.98.1)  5.353 ms  5.374 ms  5.662 ms
 4  10.117.81.253 (10.117.81.253)  3.653 ms  3.256 ms  3.124 ms
 5  10.154.8.137 (10.154.8.137)  14.970 ms  12.024 ms  11.670 ms
 6  10.255.239.170 (10.255.239.170)  12.522 ms  11.329 ms  11.278 ms
 7  10.152.7.214 (10.152.7.214)  11.533 ms  11.090 ms  11.429 ms
 8  * * *
 9  * * *
10  192.178.86.238 (192.178.86.238)  36.250 ms
    142.250.238.202 (142.250.238.202)  16.429 ms
    pnbomb-aw-in-f4.1e100.net (142.251.42.228)  16.940 ms
```

- The router may be configured to suppress ICMP responses (common for security or load reasons).
- A firewall or ACL may be blocking ICMP or UDP responses.
- (Additionally, some routers only forward packets and are not set to respond to TTL expiry messages.)

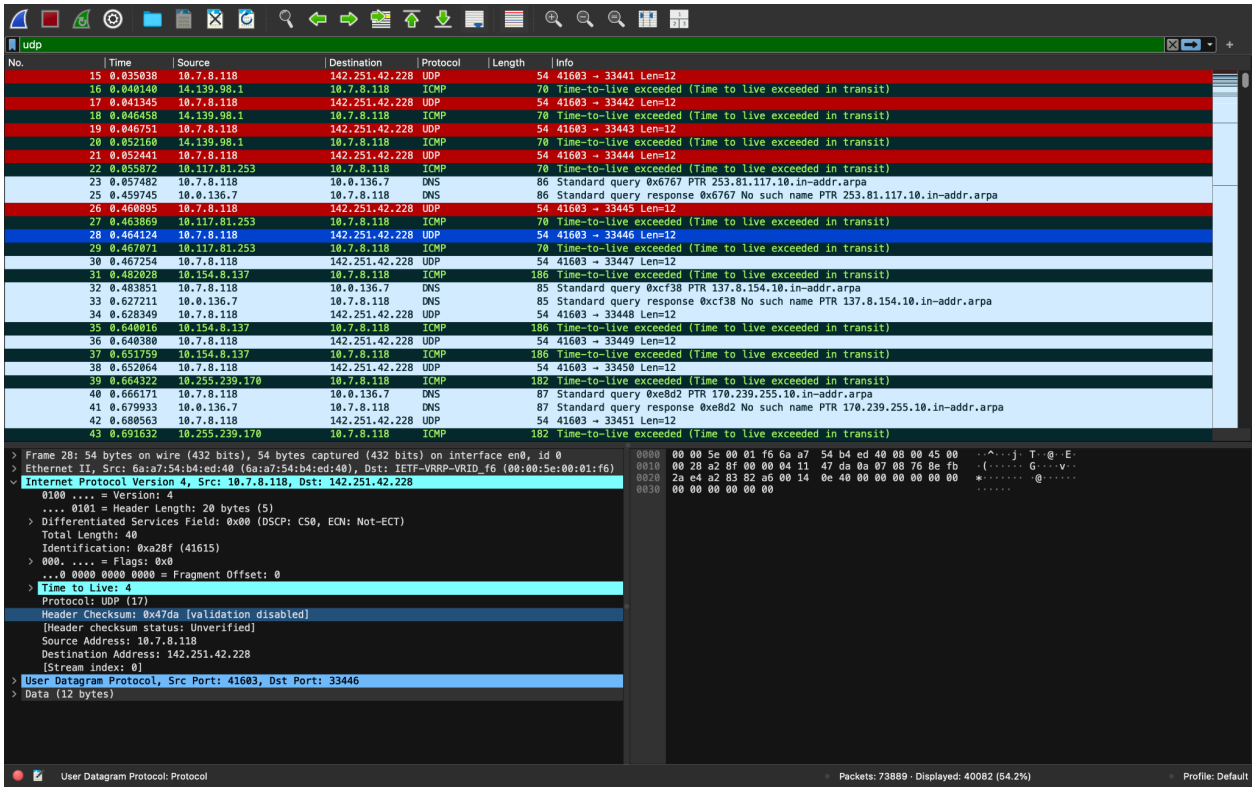
Q3. In Linux traceroute, which field in the probe packets changes between successive probes sent to the destination?

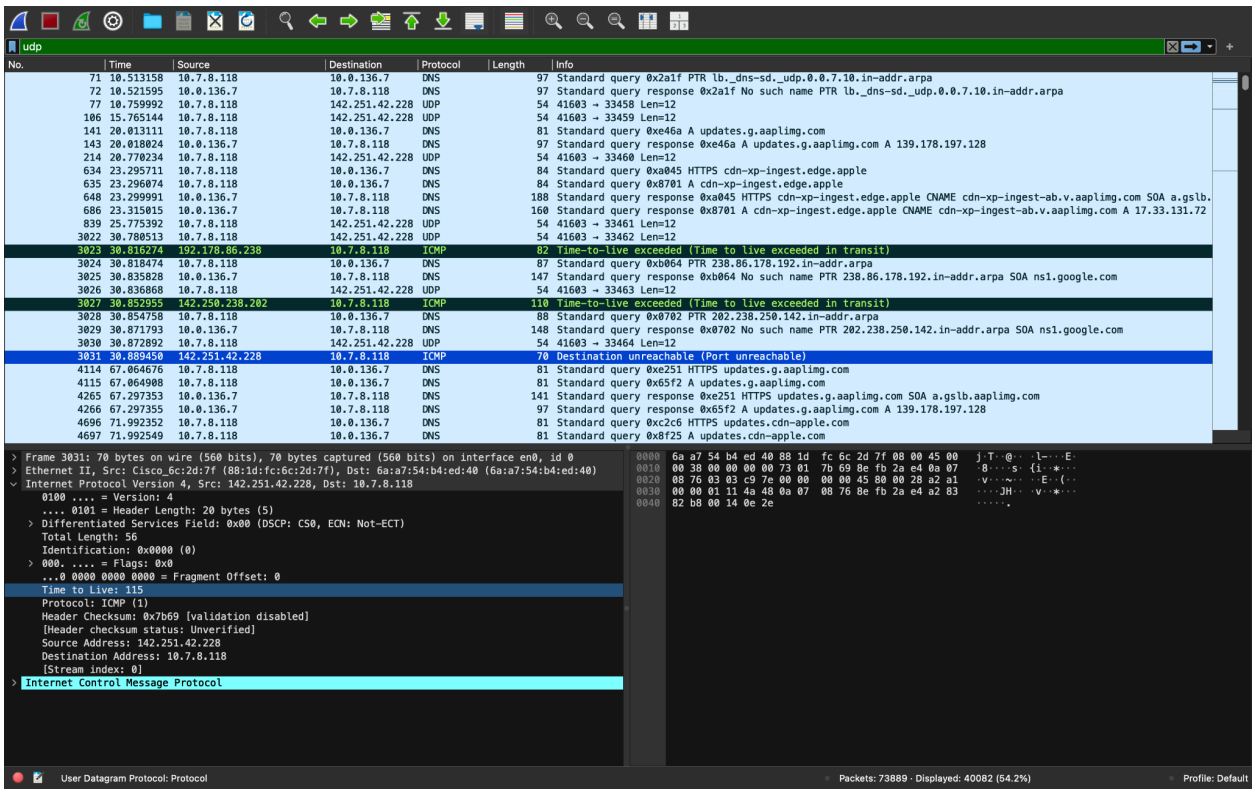




- The UDP destination port number changes between successive probes.
- TTL also changes.

Q4. At the final hop, how is the response different compared to the intermediate hop?





- Intermediate hops send back ICMP “Time-to-live exceeded” messages.
- Final hop responds differently:

- On Linux (UDP traceroute) → Destination replies with ICMP Port Unreachable (since no service listens on high UDP port).
- On Windows (ICMP tracert) → Destination replies with an ICMP Echo Reply.

Q5. Suppose a firewall blocks UDP traffic but allows ICMP — how would this affect the results of Linux traceroute vs. Windows tracert?

- Linux traceroute (UDP-based) would fail because UDP probes cannot reach the destination.
- Windows tracert (ICMP-based) would still work normally since ICMP is allowed.