CS 331: Computer Networks
Assignment 1

Acharya Vedant (23110010), Chavda Haarit (23110077)

Github Repository: (link)

TABLE OF CONTENTS

Task 1: DNS Resolver

Introduction

We implemented a custom DNS resolver by parsing DNS packets and transmitting them between a client and server. The client extracts DNS packets from a pcap file, appends a custom header, and sends them to the server. The server processes these packets using predefined rules to allocate IPs based on timestamp information in the header.

Client Implementation

The client application is responsible for processing DNS packets and forwarding them.
Key steps:

- Parsing DNS packets with dpkt:
    - We used dpkt.pcap.Reader to read and parse packets from a .pcap file.
    - The Ethernet layer (data link layer) is parsed first.
    - We checked if the packet contains an IP payload.
    - If it is an IP packet, the DNS payload is extracted from it.
- Appending a custom header:
    - A custom header was designed to include additional metadata (such as a timestamp).
    - This header is appended to the DNS packet before transmission.
- Sending to server:
    - We used Python’s socket library for communication.
    - The client created a socket with socket.socket (socket.AF_INET, socket.SOCK_DGRAM), specifying IPv4 and UDP as the transport protocol.
    - The processed packets were then sent to the server via this socket.

Server Implementation

The server is responsible for receiving and processing the DNS packets sent by the client.

- Receiving packets:
    - The server listens for incoming UDP packets through a socket.
    - Upon receiving a packet, it reads the custom header and extracts its information.
- Processing with rules.json:
    - The packet header includes a timestamp that is used to allocate IPs based on rules specified in a rules.json file.
    - The server checks these rules and allocates IPs accordingly.
- Sending edited packets
    - We are crafting new dns packets with ip addr as answers and sending it back to the client.
    - The client then parses these packets to obtain ip.

Findings and Learnings

This project helped us gain practical knowledge of network packet parsing, sockets, and DNS resolution flow.

- Sockets in Python:
  - socket.socket() creates a socket object.
  - socket.AF_INET specifies IPv4 addressing.
  - socket.SOCK_DGRAM specifies UDP as the transport protocol.
- Packet parsing with dpkt:
  - dpkt.pcap.Reader is useful for processing .pcap files.
  - Ethernet frames must be parsed first before accessing higher-layer protocols.
  - Conditional checks ensure we only process IP and DNS packets.
- Integration of application logic:
  - Added custom headers for metadata transmission.
  - Implemented server-side logic for IP allocation using JSON rule files.

Results

```
Timestamp,Client_IP,Client_Port,Domain,Selected_IP
14530800,127.0.0.1,53196,wikipedia.org,192.168.1.6
14530801,127.0.0.1,53196,reddit.com,192.168.1.7
14530802,127.0.0.1,53196,apple.com,192.168.1.8
14530803,127.0.0.1,53196,twitter.com,192.168.1.9
14530804,127.0.0.1,53196,yahoo.com,192.168.1.10
14530805,127.0.0.1,53196,linkedin.com,192.168.1.6
```

Task 2: Traceroute Protocol Behavior

Introduction

- The purpose of this task was to understand how traceroute works on two different operating systems, among Windows, Linux and MacOS, we chose Linux and Windows for our experiments
- The network IP address was identified in Windows using the ipconfig command, as shown below:

```
PS C:\Users\Vedant> ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : iitgn.ac.in

Ethernet adapter Ethernet 3:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::d431:ed3a:9e30:10ca%15
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2409:40c1:5c:2da6:ddec:6990:a50:1301
   Temporary IPv6 Address. . . . . . : 2409:40c1:5c:2da6:86:a2a5:cb27:21a4
   Link-local IPv6 Address . . . . . : fe80::2d9c:d515:fff5:c241%17
   IPv4 Address. . . . . . . . . . . : 192.168.155.177
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::f0db:c2ff:fe6c:1ef4%17
                                       192.168.155.124
```

- Network's IPv4 address: **192.168.155.177**
- Network's IPv6 address: **2409:40c1:5c:2da6:ddec:6990:a50:1301**

Basic working of traceroute

- Traceroute is a network diagnostic tool used to trace the path that packets take from a source computer to a destination host, such as a website. It works by sending packets with gradually increasing TTL (Time To Live) values, starting from 1. Each router along the path decrements the TTL by 1, and when it reaches 0, the router discards the packet and sends an ICMP "Time Exceeded" message back to the source. This allows traceroute to record the IP address of each router and the round-trip time for the response. By repeating this process with increasing TTL values until the destination is reached, traceroute reveals the sequence of routers along the path and the round-trip time at each hop.

- Following is the tracert used for Windows:

```
PS C:\Users\Vedant> tracert instagram.com

Tracing route to instagram.com [2a03:2880:f26e:1e9:face:b00c:0:4420]
over a maximum of 30 hops:

  1     4 ms     3 ms     4 ms  2409:40c1:5c:2da6::1f
  2    81 ms    97 ms    29 ms  2405:200:5210:0:3924:0:3:89
  3    64 ms   101 ms   101 ms  2405:200:5210:0:3925::1
  4     *        *        *     Request timed out.
  5     *        *        *     Request timed out.
  6    98 ms   101 ms   101 ms  2405:200:801:b00::e0c
  7     *        *        *     Request timed out.
  8    37 ms    29 ms   133 ms  ae1.pr04.pnq1.tfbnw.net [2620:0:1cff:dead:beee::5da]
  9    49 ms    82 ms    72 ms  ae1.pr04.pnq1.tfbnw.net [2620:0:1cff:dead:beee::5da]
 10    36 ms    23 ms    18 ms  po104.psw02.pnq1.tfbnw.net [2620:0:1cff:dead:bef0::8a7]
 11    47 ms    86 ms    58 ms  po2.msw1aa.02.pnq1.tfbnw.net [2a03:2880:f076:ffff::35]
 12    83 ms    35 ms    29 ms  instagram-p426-shv-02-pnq1.fbcdn.net [2a03:2880:f26e:1e9:face:b00c:0:4420]

Trace complete.
```

- The first column represents the router number, the second, third, and fourth columns show the round-trip times (RTTs) of the three packets, and the fifth column displays the IPv6 address of the router.
- Last IPv6 is of instagram.com: **2a03:2880:f26e:1e9:face:b00c:0:4420**

Questions

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

- By default, Windows tracert uses the **ICMP protocol's** Echo packets to send probes from the source to the destination. This can be observed in the Wireshark output's Info column when running the command tracert -4 instagram.com, where the -4 option ensures that IPv4 is used.

icmp

No.	Time	Source	Destination	Protocol	Length	Info
9	2.696068	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=40/10240, ttl=1 (no response found!)
10	2.700039	192.168.155.124	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
11	2.701115	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=41/10496, ttl=1 (no response found!)
12	2.705589	192.168.155.124	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
13	2.706663	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=42/10752, ttl=1 (no response found!)
14	2.709726	192.168.155.124	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
28	3.137641	192.168.155.124	192.168.155.177	ICMP	120	Destination unreachable (Port unreachable)
38	4.649820	192.168.155.124	192.168.155.177	ICMP	120	Destination unreachable (Port unreachable)
42	6.152133	192.168.155.124	192.168.155.177	ICMP	120	Destination unreachable (Port unreachable)
79	8.666777	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=43/11008, ttl=2 (no response found!)
82	8.722713	255.0.0.0	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
208	12.485897	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=44/11264, ttl=2 (no response found!)
209	12.541694	255.0.0.0	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
220	16.482052	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=45/11520, ttl=2 (no response found!)
221	16.518978	255.0.0.0	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
230	20.489479	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=46/11776, ttl=3 (no response found!)
231	20.522286	255.0.0.2	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
243	24.486131	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=47/12032, ttl=3 (no response found!)
244	24.503527	255.0.0.2	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
249	28.487303	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=48/12288, ttl=3 (no response found!)
250	28.542375	255.0.0.2	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
260	32.482743	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=49/12544, ttl=4 (no response found!)
263	32.520606	255.0.0.3	192.168.155.177	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
329	36.480108	192.168.155.177	157.240.237.174	ICMP	106	Echo (ping) request id=0x0001, seq=50/12800, ttl=4 (no response found!)

- The packets highlighted in light pink represent ping requests sent by tracert from my network's IPv4 address **(192.168.155.177)** to the host IPv4 address **(157.240.237.174)**, which belongs to Instagram. As shown, traceroute sends three packets with increasing TTL values (1, 2, …) until the destination host is reached.
- The output of the command on Windows is shown below:

```
PS C:\Users\Vedant> tracert -4 instagram.com

Tracing route to instagram.com [157.240.237.174]
over a maximum of 30 hops:

  1     4 ms     4 ms     3 ms  192.168.155.124
  2     *        *        *     Request timed out.
  3     *        *        *     Request timed out.
  4     *        *        *     Request timed out.
  5    15 ms    20 ms    47 ms  192.168.227.194
  6    56 ms    37 ms    38 ms  192.168.188.54
  7     *        *        *     Request timed out.
  8     *        *        *     Request timed out.
  9    35 ms    57 ms    70 ms  ae1.pr04.pnq1.tfbnw.net [157.240.76.56]
 10    28 ms    31 ms    38 ms  po104.psw02.pnq1.tfbnw.net [129.134.108.201]
 11    57 ms    32 ms    29 ms  msw1ad.02.pnq1.tfbnw.net [129.134.86.83]
 12    41 ms    40 ms    46 ms  instagram-p42-shv-02-pnq1.fbcdn.net [157.240.237.174]

Trace complete.
```

- By default, Linux traceroute sends **UDP protocol** packets to send probes from the source to the destination. This can be observed in the Wireshark output's Info column when running the command traceroute -4 instagram.com, where the -4 option ensures that IPv4 is used.
- The packets highlighted in red represent UDP probes sent by tracert from my network's IPv4 address **(192.168.155.177)** to the host IPv4 address **(157.240.16.174)**, which belongs to Instagram.

No.	Time	Source	Destination	Protocol	Lengtl	Info
91174	2805.6840840…	192.168.155.177	157.240.16.174	UDP	74	52524 → 33436 Len=32
91175	2805.6840933…	192.168.155.177	157.240.16.174	UDP	74	34457 → 33437 Len=32
91176	2805.6841030…	192.168.155.177	157.240.16.174	UDP	74	49887 → 33438 Len=32
91177	2805.6841133…	192.168.155.177	157.240.16.174	UDP	74	44689 → 33439 Len=32
91178	2805.6841224…	192.168.155.177	157.240.16.174	UDP	74	36125 → 33440 Len=32
91179	2805.6841310…	192.168.155.177	157.240.16.174	UDP	74	59847 → 33441 Len=32
91180	2805.6841396…	192.168.155.177	157.240.16.174	UDP	74	43991 → 33442 Len=32
91181	2805.6841487…	192.168.155.177	157.240.16.174	UDP	74	34660 → 33443 Len=32
91182	2805.6841573…	192.168.155.177	157.240.16.174	UDP	74	40793 → 33444 Len=32
91183	2805.6841655…	192.168.155.177	157.240.16.174	UDP	74	49248 → 33445 Len=32
91184	2805.6841755…	192.168.155.177	157.240.16.174	UDP	74	41307 → 33446 Len=32
91185	2805.6841844…	192.168.155.177	157.240.16.174	UDP	74	53855 → 33447 Len=32
91186	2805.6841931…	192.168.155.177	157.240.16.174	UDP	74	55872 → 33448 Len=32
91187	2805.6842015…	192.168.155.177	157.240.16.174	UDP	74	47921 → 33449 Len=32
91188	2805.6875151…	192.168.155.124	192.168.155.177	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
91189	2805.6875153…	192.168.155.124	192.168.155.177	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
91190	2805.6875736…	192.168.155.124	192.168.155.177	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)

- The output of the command on Linux is shown below:

```
traceroute -4 instagram.com
traceroute to instagram.com (157.240.237.174), 30 hops max, 60 byte packets
 1  _gateway (192.168.155.124)  1.675 ms  1.642 ms  1.812 ms
 2  255.0.0.0 (255.0.0.0)  45.711 ms  45.926 ms  47.591 ms
 3  255.0.0.2 (255.0.0.2)  48.385 ms  48.543 ms  49.280 ms
 4  255.0.0.3 (255.0.0.3)  47.708 ms  47.955 ms  48.132 ms
 5  192.168.227.195 (192.168.227.195)  50.680 ms 192.168.227.194 (192.168.227.194)  53.308 ms  53.759 ms
 6  192.168.188.54 (192.168.188.54)  53.237 ms  51.696 ms 192.168.188.52 (192.168.188.52)  52.903 ms
 7  * * *
 8  * * *
 9  ae1.pr04.pnq1.tfbnw.net (157.240.76.56)  63.544 ms  59.659 ms  95.023 ms
10  po104.psw04.pnq1.tfbnw.net (129.134.108.217)  63.193 ms po104.psw01.pnq1.tfbnw.net (129.134.108.193)  63.183 ms po104.psw03.pnq1.tfbnw.net (129.134.108.209)  60.344 ms
11  msw1ac.02.pnq1.tfbnw.net (129.134.86.84)  62.819 ms msw1ah.02.pnq1.tfbnw.net (129.134.86.79)  62.758 ms msw1aa.02.pnq1.tfbnw.net (129.134.86.86)  63.815 ms
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

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

- Traceroute sends 3 probes to each hop. If a probe does not receive a response within the timeout period, traceroute displays an asterisk for that probe. If all three probes for a hop time out, we see * * *. This indicates that the router is not replying to traceroute. The possible reasons include:
    - **Firewall:** Firewalls at the local network, ISP, or intermediate routers may block the packets that traceroute uses (such as ICMP, UDP, or certain ports). As a result, traceroute cannot receive replies from that router.
    - **Configuration:** Many routers are intentionally configured not to send back TTL-expired (or similar) ICMP/UDP responses to traceroute probes. For security or policy reasons, they may drop or ignore these packets.
    - **Packet loss:** A router may be overloaded, busy, or experiencing traffic loss. So it might simply fail to respond in time or drop replies.

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

- traceroute -4 google.com command output on Linux is shown below:

```
 ~  traceroute -4  google.com                                          ✓  base
traceroute to google.com (172.217.24.78), 30 hops max, 60 byte packets
 1  _gateway (192.168.155.124)  3.169 ms  3.148 ms  3.371 ms
 2  255.0.0.0 (255.0.0.0)  18.060 ms  22.527 ms  22.732 ms
 3  255.0.0.2 (255.0.0.2)  25.075 ms  25.274 ms  25.832 ms
 4  255.0.0.3 (255.0.0.3)  25.587 ms  28.892 ms  28.665 ms
 5  192.168.227.195 (192.168.227.195)  29.118 ms 192.168.227.194 (192.168.227.194)  29.843 ms  29.879 ms
 6  192.168.188.48 (192.168.188.48)  29.609 ms 192.168.188.50 (192.168.188.50)  27.038 ms  26.842 ms
 7  * * *
 8  * * *
 9  * 173.194.121.8 (173.194.121.8)  30.372 ms *
10  * * *
11  142.251.77.94 (142.251.77.94)  30.689 ms 142.251.69.102 (142.251.69.102)  32.458 ms 142.250.212.170 (142.250.212.170)  31.885 ms
12  192.178.110.104 (192.178.110.104)  32.129 ms  31.249 ms  27.346 ms
13  142.251.197.253 (142.251.197.253)  47.593 ms 142.251.198.3 (142.251.198.3)  47.338 ms 142.251.198.1 (142.251.198.1)  46.705 ms
14  142.251.255.57 (142.251.255.57)  46.421 ms 192.178.252.110 (192.178.252.110)  42.093 ms 192.178.252.114 (192.178.252.114)  41.255 ms
15  142.251.76.199 (142.251.76.199)  47.562 ms 192.178.82.233 (192.178.82.233)  43.561 ms 192.178.82.235 (192.178.82.235)  43.530 ms
16  sin10s06-in-f14.1e100.net (172.217.24.78)  48.213 ms  45.941 ms  41.604 ms
```

- As shown in the Wireshark output, the Info field displays the source and destination UDP ports for each probe. The first value corresponds to the source UDP port, and the second value corresponds to the destination UDP port.

No.	Time	Source	Destination	Protocol	Lengtl	Info
353	76.200784934	192.168.155.177	172.217.24.78	UDP	74	60508 → 33434 Len=32
354	76.200797603	192.168.155.177	172.217.24.78	UDP	74	41096 → 33435 Len=32
355	76.200803919	192.168.155.177	172.217.24.78	UDP	74	35873 → 33436 Len=32
356	76.200810269	192.168.155.177	172.217.24.78	UDP	74	54600 → 33437 Len=32
357	76.200816011	192.168.155.177	172.217.24.78	UDP	74	58473 → 33438 Len=32
358	76.200821530	192.168.155.177	172.217.24.78	UDP	74	45148 → 33439 Len=32
359	76.200827512	192.168.155.177	172.217.24.78	UDP	74	42173 → 33440 Len=32
360	76.200832971	192.168.155.177	172.217.24.78	UDP	74	60478 → 33441 Len=32
361	76.200839571	192.168.155.177	172.217.24.78	UDP	74	41348 → 33442 Len=32
362	76.200844962	192.168.155.177	172.217.24.78	UDP	74	43754 → 33443 Len=32
363	76.200850713	192.168.155.177	172.217.24.78	UDP	74	46729 → 33444 Len=32
364	76.200856445	192.168.155.177	172.217.24.78	UDP	74	37525 → 33445 Len=32

- Across successive probes, both values change. For example, in the green-highlighted entry, the first probe uses src port = 60508, dest port = 33434, while the following probe uses src port = 41096, dest port = 33435. Additionally, after every three probes, the TTL (Time to Live) value increases. This allows traceroute to explore the next hop along the path.

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

- As shown in the Wireshark output from running traceroute -4 google.com on Linux, the ICMP packets highlighted in black represent the responses to the probe requests. For intermediate hops, the response type is **Time-to-live exceeded**, since the probe's TTL expires before reaching the destination. At the final hop (highlighted in green), because the destination host has been reached, the response changes to **Destination unreachable (Port unreachable)**.

No.	Time	Source	Destination	Protocol	Lengtl	Info
443	76.383904803	192.178.252.114	192.168.155.177	ICMP	70	Time-to-live exceeded (Time to live exceeded i
444	76.384102714	192.168.155.177	172.217.24.78	UDP	74	56619 → 33480 Len=32
445	76.386277427	192.178.82.233	192.168.155.177	ICMP	110	Time-to-live exceeded (Time to live exceeded i
446	76.386277720	192.178.82.235	192.168.155.177	ICMP	110	Time-to-live exceeded (Time to live exceeded i
447	76.386449226	192.168.155.177	172.217.24.78	UDP	74	38831 → 33481 Len=32
448	76.386471914	192.168.155.177	172.217.24.78	UDP	74	46218 → 33482 Len=32
449	76.390262582	142.251.76.199	192.168.155.177	ICMP	102	Time-to-live exceeded (Time to live exceeded i
450	76.390444933	192.168.155.177	172.217.24.78	UDP	74	47849 → 33483 Len=32
451	76.412416402	172.217.24.78	192.168.155.177	ICMP	70	Destination unreachable (Port unreachable)
452	76.428040433	172.217.24.78	192.168.155.177	ICMP	70	Destination unreachable (Port unreachable)
453	76.428040842	172.217.24.78	192.168.155.177	ICMP	70	Destination unreachable (Port unreachable)
454	76.430029874	172.217.24.78	192.168.155.177	ICMP	70	Destination unreachable (Port unreachable)
455	76.431166465	172.217.24.78	192.168.155.177	ICMP	70	Destination unreachable (Port unreachable)

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

- By default, Linux traceroute sends UDP probes to high-numbered destination ports. Since the firewall blocks UDP, these probes won't reach the destination and won't trigger responses, causing traceroute to display * * * for many hops, effectively failing unless run with **ICMP (-I)** or **TCP (-T)** options. In contrast, Windows tracert uses ICMP Echo Request packets by default. Because ICMP is allowed, probes will reach the destination, and intermediate routers will reply with ICMP Time Exceeded messages, and the final destination will respond with an ICMP Echo Reply. As a result, Windows tracert continues to work normally even when UDP is blocked.