Computer Networks
Assignment 1

Students Jaskirat Singh Maskeen (23110146) & Karan Sagar Gandhi (23110157)
Professor Sameer G. Kulkarni

1 DNS Resolver

The objective of this task was to develop a custom DNS resolution system. The system consists of a client that parses DNS queries from a PCAP file and a server that resolves these queries based on a set of custom rules.

1.1 DNS Packet Structure

We read the RFC 1035 [2], and the packet structure is summarized in the figure below (following that we have given an example).

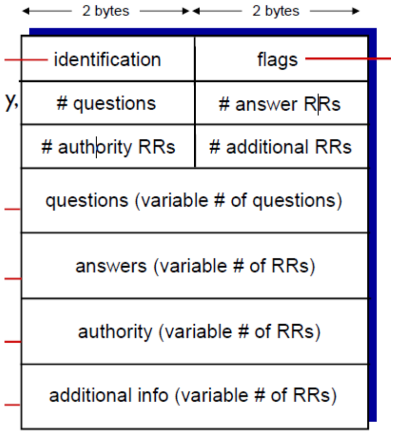

2 bytes
2 bytes
identification
flags
# questions
# answer RRs
# authority RRs
# additional RRs
questions (variable # of questions)
answers (variable # of RRs)
authority (variable # of RRs)
additional info (variable # of RRs)

Figure 1: Packet structure from slides [1].

Then we write our custom parser, which specifically parses the domain name from the DNS packet, as that is the only thing required from the DNS packet for this task. The meaning of fields can be read from the RFC, however the main part we had to implement was decompression of the labels. Strings which appear more than once are stored as a pointer and a offset, after their first appearance. For example:

Suppose our quesiton is `A` (IPV4) record for `iitgn.ac.in`, and the answer will be `CNAME` pointing to `iitgn.ac.in`. So to save space, the answer will reuse the bytes of `iitgn.ac.in` which would be present in the question itself.

```
Header (12 bytes):
  TxnID  Flags
  1A 2B  81 80
   QD     AN
  00 01  00 01
   NS     AR
  00 00  00 00
             ^
            0x0C
Question section (starts at offset 0x0C = 12 which is the size of header):
This section is stored as length labelled format.
          0x10
            v
  03 77 77 77 05 69 69 74 67 6E 02 61 63 02 69 6E 00
     w  w  w     i  i  t  g  n     a  c     i  n  (terminated by 00)
  00 01     (QTYPE = A)
  00 01     (QCLASS = IN)

Answer section:
  C0 0C     (NAME = pointer to offset 0x0C -> "www.iitgn.ac.in")
            (C0 means 1100 0000, which means that this is a pointer)
  00 05     (TYPE = CNAME)
  00 01     (CLASS = IN)
  00 00 00 3C (TTL = 60)
  00 02     (RDLENGTH = 2 bytes)
  C0 10     (RDATA = pointer to offset 0x10 -> "iitgn.ac.in")
            (C0 means 1100 0000, which means that this is a pointer)
```

1.2 System Architecture and Flow

The operational flow of the system is as follows:

1. **Packet Filtering:** The client begins by reading a given `.pcap` file and filtering it to isolate the DNS query packets (these are the queries which are sent to UDP port 53).

2. **Custom Header Addition:** For each DNS query, the client generates an 8-byte custom header with the format `"HHMMSSID"`. This header contains the current time and a two-digit sequence ID for the query (The sequence ID is incremented after each query).

3. **Communication:** The client sends the original DNS query, prefixed with this custom header, to the server.

4. **Server-Side Processing:** The server receives the message, parses the custom header to determine the appropriate IP pool based on the timestamp, and extracts the domain name from the DNS query payload (following [2]).

5. **Response and Logging:** The server sends the resolved IP address, the domain name, and the original custom header back to the client. The client then logs this information. To better simulate real-world conditions, we allow the client introduces an artificial delay (random, configurable) between sending DNS queries.

1.3 Transport Protocol: TCP to UDP

1.3.1 Initial TCP Implementation

Our initial implementation for the client-server communication was built using TCP (`SOCK_STREAM`). However, TCP being a stream protocol, without message boundaries, we had to implement our own mechanism to identify the start of the message. This was done by prefixing each payload with a 4-byte unsigned integer which represents the payload's length. The receiver (client or server) would first read these 4 bytes to determine the message size and then read that exact number of bytes to get the complete message.

1.3.2 Current UDP Implementation

After a discussion with the professor, we decided to use UDP (`SOCK_DGRAM`) as our communication protocol. This change was motivated by the fact that real-world DNS queries majorly use UDP due to its low overhead (No handshakes, unlike TCP). However it is to be noted that incase size of the message is more than 512 bytes, TCP will be used [2], but we stick to UDP in our implementation.

UDP is a message-oriented protocol, meaning it preserves message boundaries automatically. Hence this allowed us to remove the manual 4-byte length prefixing, simplifying our message handling logic.

1.4 Implementation Details

- `3.pcap`: The pcap file from where we process the DNS queries. $(23110146 + 23110157 \equiv 3 \pmod{10})$

- `client.py`: Manages reading the PCAP file, sending queries, and displaying results.

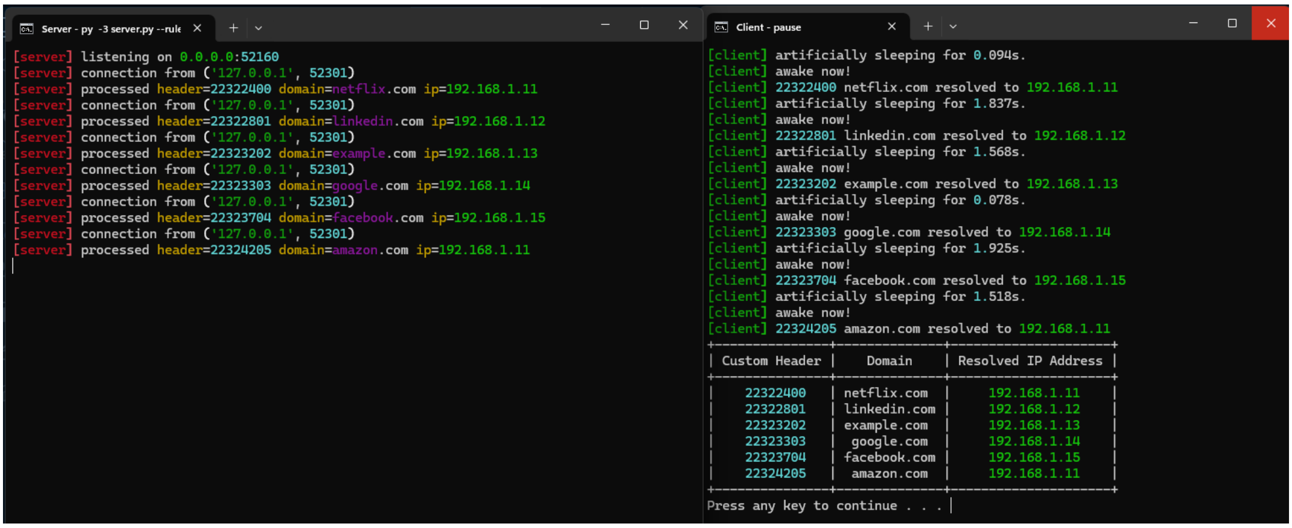
- `server.py`: Listens for incoming queries, applies the routing logic, and sends back responses.

- `helpers.py`: Contains utility functions for DNS packet parsing, including logic to handle domain name decompression as specified in [2].
- `rules.json`: An external configuration file that defines the time-based routing rules, allowing for easy modification without changing the server code.

1.5 Results

The client successfully processed the DNS queries from `3.pcap` and received the resolved IP addresses from the server. The final output is shown in the table below. Note that we ran this at `Tuesday 09/09/2025 22:32:20`.

Custom Header	Domain	Resolved IP Address
22322400	`netflix.com`	`192.168.1.11`
22322801	`linkedin.com`	`192.168.1.12`
22323202	`example.com`	`192.168.1.13`
22323303	`google.com`	`192.168.1.14`
22323704	`facebook.com`	`192.168.1.15`
22324205	`amazon.com`	`192.168.1.11`



```
[server] listening on 0.0.0.0:52160
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22322400 domain=netflix.com ip=192.168.1.11
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22322801 domain=linkedin.com ip=192.168.1.12
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22323202 domain=example.com ip=192.168.1.13
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22323303 domain=google.com ip=192.168.1.14
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22323704 domain=facebook.com ip=192.168.1.15
[server] connection from ('127.0.0.1', 52301)
[server] processed header=22324205 domain=amazon.com ip=192.168.1.11
```

```
[client] artificially sleeping for 0.094s.
[client] awake now!
[client] 22322400 netflix.com resolved to 192.168.1.11
[client] artificially sleeping for 1.837s.
[client] awake now!
[client] 22322801 linkedin.com resolved to 192.168.1.12
[client] artificially sleeping for 1.568s.
[client] awake now!
[client] 22323202 example.com resolved to 192.168.1.13
[client] artificially sleeping for 0.078s.
[client] awake now!
[client] 22323303 google.com resolved to 192.168.1.14
[client] artificially sleeping for 1.925s.
[client] awake now!
[client] 22323704 facebook.com resolved to 192.168.1.15
[client] artificially sleeping for 1.518s.
[client] awake now!
[client] 22324205 amazon.com resolved to 192.168.1.11
+---------------+--------------+---------------------+
| Custom Header |    Domain    | Resolved IP Address |
+---------------+--------------+---------------------+
|    22322400   | netflix.com  |     192.168.1.11    |
|    22322801   | linkedin.com |     192.168.1.12    |
|    22323202   | example.com  |     192.168.1.13    |
|    22323303   |  google.com  |     192.168.1.14    |
|    22323704   | facebook.com |     192.168.1.15    |
|    22324205   |  amazon.com  |     192.168.1.11    |
+---------------+--------------+---------------------+
Press any key to continue . . .
```

Figure 2: Screenshot of the client and server running.

1.6 Important sections refered in [2]

1. 4.1.1. Header section format
2. 4.1.2. Question section format
3. 4.1.4. Message compression

References

[1] S. G. Kulkarni. DNS PROTOCOL, MESSAGES. Microsoft Teams, September 2025.

[2] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, November 1987.