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 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 [1]).
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.

1.2 Transport Protocol: TCP to UDP

1.2.1 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.

1.2.2 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 [1], 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.

1.3 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 [1].
- `rules.json`: An external configuration file that defines the time-based routing rules, allowing for easy modification without changing the server code.

1.4 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`

1.5 Important sections refered in [1]

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

References

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

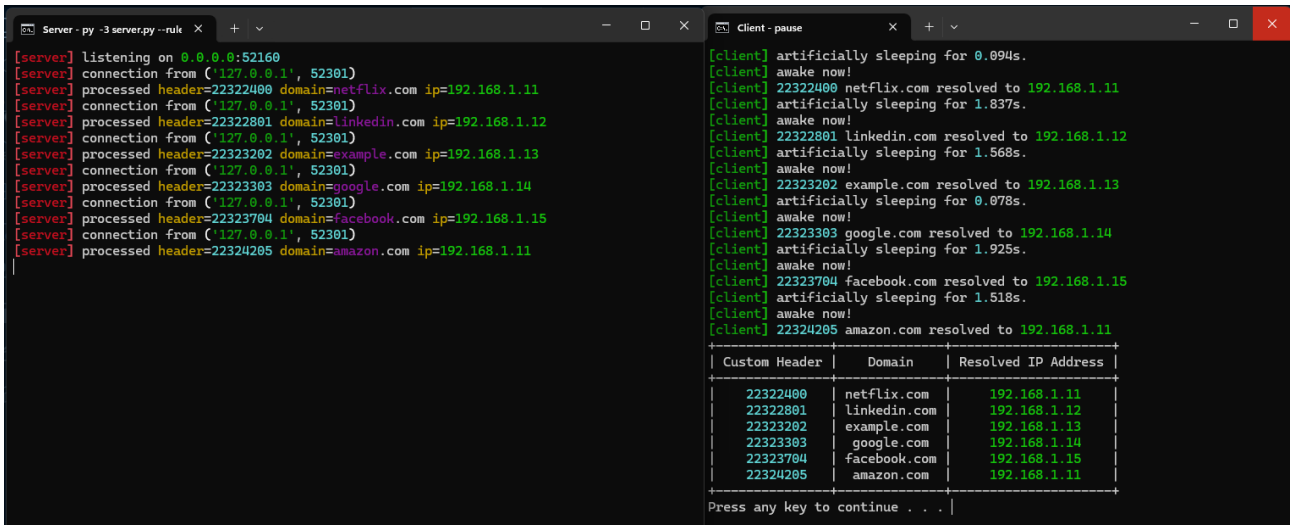

Figure 1: Screenshot of the client and server running.