Assignment 1

CS331: Computer Networks

Siddhesh Umarjee (23110347) Shounak Ranade (23110304)

# task 1: Custom header and dns resolver

## Introduction

Task 1 implements a lightweight client–server DNS resolver. It helps us understand packet parsing, header design, and policy-driven IP routing. The client filters DNS queries from a given PCAP, prepends an 8-byte custom header “HHMMSSID” carrying a UTC timestamp and sequential ID, and sends the message unchanged otherwise; the server parses the header, extracts the domain from the original DNS payload, and applies time-based routing with a 15-IP pool by selecting a 5-address segment based on hour and choosing a specific address via ID modulo, finally returning and logging the resolved address alongside the original query.

We use the language Go for this task.

## Prerequisite

The GitHub repository can be found here: <https://github.com/IdkRandomTry/DNS-Resolver>. The README.md file has the necessary instructions to run the code. Go 1.21+ recommended

Required Packages:

* Standard library: net (UDP sockets), time (UTC header), fmt/log/os/bufio/flag/strconv.
* github.com/google/gopacket and github.com/google/gopacket/pcap for PCAP reading and packet iteration.
* github.com/google/gopacket/layers for DNS layer extraction.
* golang.org/x/net/dns/dnsmessage for safe DNS question parsing on the server.

## Methodology

We maintain a folder structure as follows:

dns\_resolver/

- client/

  - client.go       # Client code

  - 1.pcap           # PCAP file

- server/

  - extract\_domain.go # Extract domain name

- server.go       # UDP server

  - ip\_select.go     # IP selection logic

- .gitignore        # Recommended ignore rules

- go.mod            # Go module file

### Client

The client takes a PCAP (Packet Caapture), iterates packets, and filters DNS queries, ensuring responses (QR=1) are skipped to avoid duplication. For each query, it constructs an 8-byte custom header “HHMMSSID” using UTC time and a (zero-padded) sequential ID. It then prefixes this header to the untouched DNS payload and transmits the resulting datagram via UDP to the server endpoint. The client maintains a simple, sequential ID counter modulo 100 to bound header size. It also logs the timestamp, ID, and send status for traceability during test runs.

### Server

The server listens on a UDP socket on port 1053 (stand-in for port 53). It extracts the DNS payload and slices the first 8 bytes as the custom header. The remaining bytes are parsed as a DNS message to extract the queried domain name using the function defined in `extract\_domain.go`. The server the applies time-based routing by mapping the hour to one of three 5-IP pool segments (morning/afternoon/night) and selects an IP using ID % 5 to index within that segment, thus implementing deterministic DNS-style load distribution. The function used is defined in `ip\_select.go`

## Results

A computer screen with white text

AI-generated content may be incorrect.A computer screen with white text

AI-generated content may be incorrect.Here are results from running the code at different times. (Adjusted time for testing)

## Conclusion