# Computer Networks
# CSE351

**DNS-SERVER-SYSTEM**

**Final Report**

## Team 31

**Team Members:**

| | |
|---|---|
| Mohamed Khaled Elhelaly | 21P0185 |
| Mahmoud Saber | 21P0231 |
| Ali Sherif Hassan | 21P0212 |

## Table of Contents

# Introduction

The Domain Name System (DNS) is a cornerstone of modern internet communication, translating human-readable domain names into IP addresses required for network routing. This system operates as a hierarchical and distributed database, defined by RFCs 1034 and 1035, which standardize DNS behavior and message formatting. Additionally, RFC 2181 clarifies DNS operational concepts, extending the reliability and robustness of DNS implementations.

This report provides an in-depth explanation and analysis of a DNS system implemented for academic and experimental purposes. The implemented system is composed of a DNS resolver, a root server, a top-level domain (TLD) server, and an authoritative server, designed to follow the iterative query resolution process. Each component adheres to the specifications outlined in the aforementioned RFCs, ensuring compliance with established standards.

## Purpose & Scope

The primary objective of this project is to implement a fully functional DNS resolver and server hierarchy, including a root server, TLD server, and authoritative server. This implementation emphasizes the following core aspects:

1. **Iterative Query Resolution**: The resolver performs iterative DNS queries, starting from the root server and traversing the hierarchy until the authoritative server provides the final answer.
2. **Error Handling**: The system adheres to DNS standards by identifying and responding to errors such as non-existent domains (NXDOMAIN), format errors (FORMERR), unsupported query types, and server failures (SERVFAIL).
3. **UDP Protocol Usage**: All DNS communication is conducted over the User Datagram Protocol (UDP), reflecting its widespread use in DNS due to its efficiency and low overhead.

## Compliance with RFC standards

This project aligns with the following key principles and requirements outlined in the RFCs:

- **RFC 1034**: Defines the hierarchical and distributed nature of DNS, detailing the query process and the interaction between resolvers and name servers.
- **RFC 1035**: Specifies the DNS message format, including the question, answer, authority, and additional sections, as well as the rules for name resolution and record retrieval.
- **RFC 2181**: Enhances the original specifications with clarifications on DNS consistency, error handling, and the interpretation of resource records.

## Learning Objectives

This implementation aims to:

- Provide a hands-on understanding of DNS operations, message formatting, and protocol standards.

- Illustrate the design and functionality of a DNS hierarchy, including its resolver and server components.
- Explore error handling and compliance with the DNS protocol to ensure robust and predictable behavior.

## Overview

The report documents the architecture and codebase of the DNS system, detailing the functionality of each component (resolver, root server, TLD server, and authoritative server) and their interactions. Additionally, it highlights the implementation of error handling mechanisms and test cases to verify system correctness under various scenarios. Finally, instructions for installation, configuration, and usage are provided to facilitate replication and testing.

This project offers a comprehensive insight into the principles of DNS and its real-world implementation, serving as both a learning resource and a demonstration of the DNS architecture in practice.

# DNS Message

DNS allows you to interact with devices on the Internet without having to remember long strings of numbers. Changing of information between client and server is carried out by two types of DNS messages:

1) Query message
2) Response message.

The format is similar for both types of messages. The information is held up in up to five different sections of DNS message format.

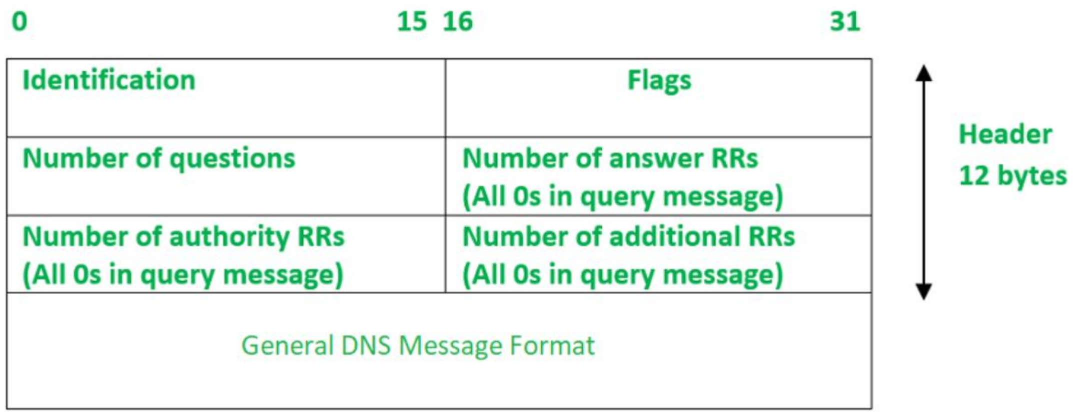
The query message consist of two sections:

- Header
- Question

The response message consists of five sections:

- Header
- Question
- Answer records
- Authoritative records
- Additional records

## DNS Header Section


0
15 16
31
Identification
Flags
Number of questions
Number of answer RRs (All 0s in query message)
Number of authority RRs (All 0s in query message)
Number of additional RRs (All 0s in query message)
General DNS Message Format
Header 12 bytes

**Identification**: The identification field is made up of 16 bits which are used to match the response with the request sent from the client-side. The matching is carried out by this field as the server copies the 16-bit value of identification in the response message so the client device can match the queries with the corresponding response received from the server-side.

**Flags:** It is 16 bits and is divided into the following Fields:

| QR | Opcode | AA | TC | RD | RA | zero | rCode |
|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 1 | 1 | 1 | 3 | 4 |

1) **QR (query/response):** It is a 1-bit subfield. If its value is 0, the message is of request type and if its value is 1, the message is of response type.
2) **opcode:** It is a 4-bit subfield that defines the type of query carried by a message. This field value is repeated in the response. Following is the list of opcode values with a brief description:
    - If the value of the opcode subfield is 0 then it is a standard query.
    - The value 1 corresponds to an inverse of query that implies finding the domain name from the IP Address.
    - The value 2 refers to the server status request. The value 3 specifies the status reserved and therefore not used.
3) **AA:** It is an Authoritative Answer. It is a 1-bit subfield that specifies the server is authoritative if the value is 1 otherwise it is non-authoritative for a 0 value.
4) **TC:** It is Truncation. This is a 1-bit subfield that specifies if the length of the message exceeds the allowed length of 512 bytes, the message is truncated when using UDP services.

5) **RD:** It is Recursion Desired. It is a 1-bit subfield that specifies if the value is set to 1 in the query message then the server needs to answer the query recursively. Its value is copied to the response message.
6) **RA:** It is Recursion Available. It is a 1-bit subfield that specifies the availability of recursive response if the value is set to 1 in the response message.
7) **Zero:** It is a 3-bit reserved subfield set to 0.
8) **rCode:** It stands for Response Code. It is a 4-bit subfield used to denote whether the query was answered successfully or not. If not answered successfully then the status of error is provided in the response. Following is the list of values with their error status –
    - The value 0 of rcode indicates no error.
    - A value of 1 indicates that there is a problem with the format specification.
    - Value 2 indicates server failure.
    - Value 3 refers to the Name Error that implies the name given by the query does not exist in the domain.
    - Value of 4 indicates that the request type is not supported by the server.
    - The value 5 refers to the nonexecution of queries by the server due to policy reasons.

**Number of Questions:** It is a 16-bit field to specify the count of questions in the Question Section of the message. It is present in both query and response messages.

**A number of answer RRs:** It is a 16-bit field that specifies the count of answer records in the Answer section of the message. This section has a value of 0 in query messages. The server answers the query received from the client. It is available only in response messages.

**A number of authority RRs:** It is a 16-bit field that gives the count of the resource records in the Authoritative section of the message. This section has a value of 0 in query messages. It is available only in response messages. It gives information that comprises domain names about one or more authoritative servers.

**A number of additional RRs:** It is a 16-bit field that holds additional records to keep additional information to help the resolver. This section has a value of 0 in query messages. It is available only in response messages.

## DNS Question Section

The Question Section contains information about the query being made. It is typically the part of the DNS message where the client specifies the domain name and the type of record it is requesting (such as an A, AAAA, MX, or NS record). The Question Section is included in both DNS queries and responses.

The Question Section consists of the following fields:

### 1) QNAME (Domain Name):

This is the domain name being queried, represented in a sequence of labels. Each label starts with a byte indicating its length, followed by the label itself (the name), and labels are separated by a

zero byte. The domain name ends with a zero byte to indicate the end of the name.
For example, "www.example.com" would be represented as:

- 3 bytes for "www"
- 7 bytes for "example"
- 3 bytes for "com"
- 1 byte to indicate the end (0x00)

### 2) QTYPE (Query Type):

This field specifies the type of DNS record being requested. The QTYPE field is a 2-byte field that can take several values:

- 1: A (Address record - IPv4 address)
- 2: NS (Nameserver record)
- 5: CNAME (Canonical Name record)
- 6: SOA (Start of Authority record)
- 12: PTR (Pointer record - Reverse DNS)
- 15: MX (Mail Exchange record)
- 16: TXT (Text record)
- 28: AAAA (IPv6 Address record)
- 33: SRV (Service record)
- 53: CAA (Certification Authority Authorization record)

### 3) QCLASS (Query Class):

This field specifies the class of the DNS query. Typically, this is set to 1, which represents the IN (Internet) class.

## DNS Answer Section

The DNS Answer Section is a part of the DNS response message that contains the resource records (RRs) answering the DNS query. It provides the actual data that the client requested, typically for a specific domain name.

**Key Components:**

- **Resource Records (RRs**): Each record in the answer section provides the requested data, such as IP addresses, mail exchange information, or name server details.
- **Format:** Each record in the answer section follows a specific format that includes:
- **Name:** The domain name for which the record is being returned (usually encoded).
- **Type:** The type of DNS record (A, AAAA, CNAME, NS, MX, etc.).
- **Class:** The class of the record, generally IN (Internet).

- **TTL (Time to Live):** The time in seconds that the record is considered valid, after which it should be refreshed.
- **RDATA:** The data specific to the record type. This could be an IP address (for A/AAAA records), a CNAME (for CNAME records), a list of mail exchange servers (for MX records), etc.

### DNS Authority Section

The DNS Authority Section is a part of the DNS response message that provides information about the authoritative name servers for the domain being queried. It typically contains NS (Nameserver) records that point to the servers responsible for the domain, which can be used by the client to query further if necessary.

#### Key Components:

- **NS (Nameserver) Records**: These records point to the authoritative DNS servers for the queried domain. Each NS record specifies the domain name of a nameserver responsible for the queried zone.
- **Format of a Nameserver (NS) Record:**
    - Name: The domain for which this NS record is authoritative.
    - Type: 2 (for NS records).
    - Class: IN (for Internet).
- **TTL:** The Time To Live for this NS record, which specifies how long the information can be cached.
- **RDATA:** The domain name of the authoritative nameserver for the zone. This is often a fully qualified domain name (FQDN), such as ns1.example.com.

### DNS Additional Section

The **DNS Additional Section** is a part of the DNS response message that provides additional resource records (RRs) which might be useful for the client to complete its query. This section typically contains extra data related to the response, such as **A (Address)** or **AAAA (IPv6 Address)** records for the nameservers listed in the **Authority Section**.

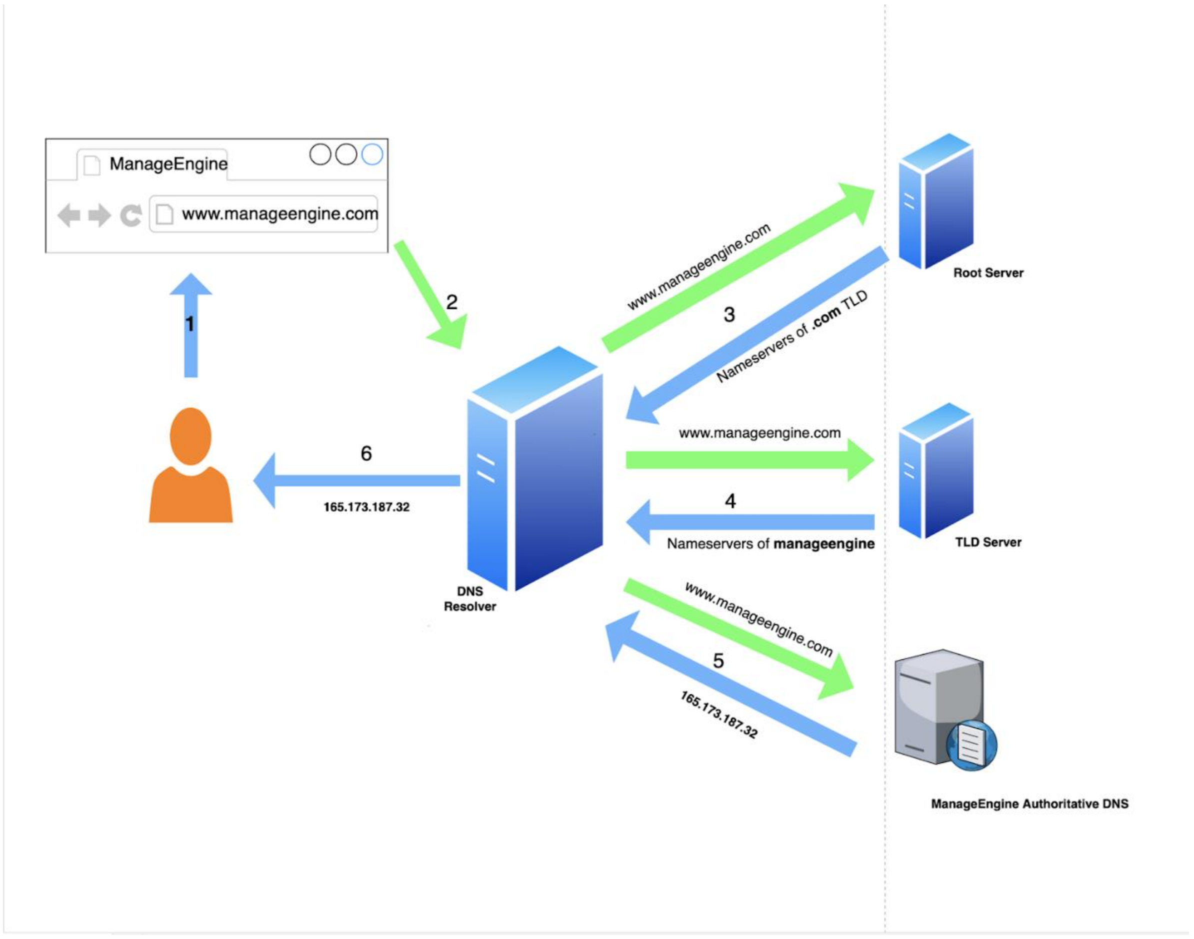
## System Architecture

The implemented DNS system follows the hierarchical and distributed model defined in **RFC 1034** and **RFC 1035**, with a clear delineation of roles and responsibilities among its components. This architecture replicates the real-world structure of the Domain Name System by simulating the interactions between a DNS resolver and the three primary levels of the DNS hierarchy: the root server, TLD server, and authoritative server.

## Overview of Components

1. **DNS Resolver**:
    - The resolver acts as the intermediary between the client and the DNS hierarchy.
    - It receives queries from clients, processes them iteratively by querying the DNS hierarchy, and returns the final answer or error to the client.
    - The resolver is implemented using the **UDP protocol** and adheres to the iterative resolution process defined in the DNS specifications.
2. **Root Server**:
    - The root server is the entry point to the DNS hierarchy.
    - It maintains a database of NS (Name Server) records for top-level domains (TLDs) such as .com, .org, .net, etc.
    - Upon receiving a query, it responds with the NS record pointing to the relevant TLD server.
3. **TLD Server**:
    - The TLD server handles queries for domains within its namespace (e.g., example.com, example.org).
    - It maintains NS records for authoritative servers responsible for specific domains under its TLD.
    - The TLD server forwards the resolver to the authoritative server for the requested domain.
4. **Authoritative Server**:
    - The authoritative server provides the final resolution for a domain query.
    - It stores detailed resource records (e.g., A, MX, CNAME) for the domains it manages.
    - When queried, it returns the requested resource record to the resolver.

## System Flow Diagram



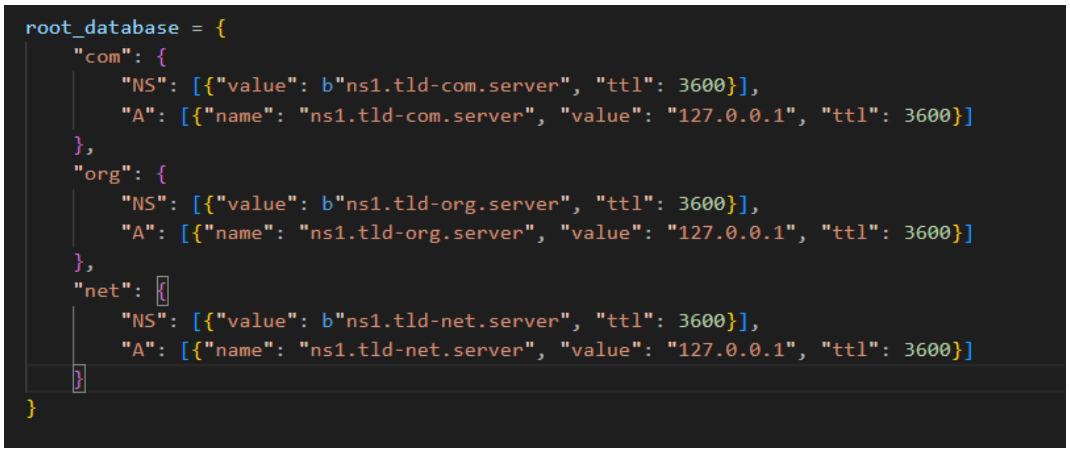
1. **Client Query**:
    - Client → Resolver (Query: example.com, A record).
2. **Resolver to Root Server**:
    - Resolver → Root Server: Query for example.com.
    - Root Server → Resolver: NS record for .com TLD server.
3. **Resolver to TLD Server**:
    - Resolver → TLD Server: Query for example.com.
    - TLD Server → Resolver: NS record for example.com authoritative server.
4. **Resolver to Authoritative Server**:
    - Resolver → Authoritative Server: Query for example.com, A record.

- Authoritative Server → Resolver: A record for example.com.

## Database Structure

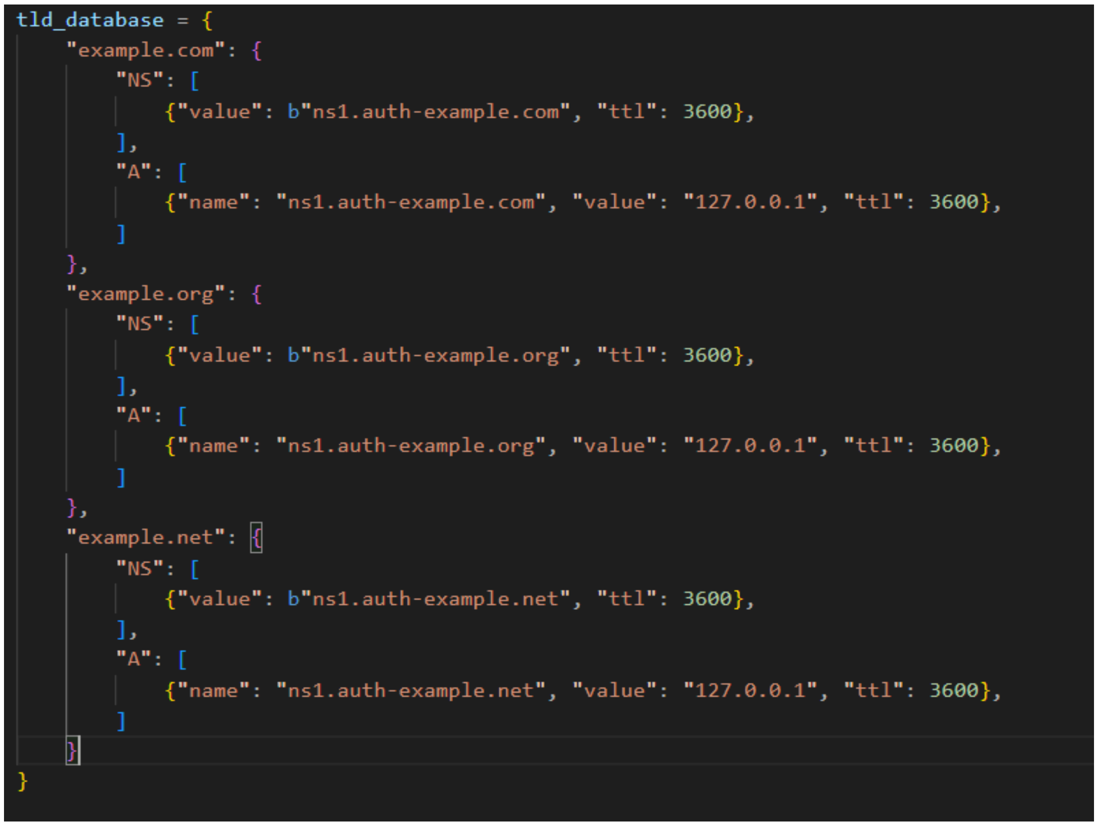
Each server maintains its own database:

1. **Root Server**:
   - **NS Records**: Lists the authoritative nameservers for top-level domains (TLDs), such as .com, .org, and .net.
   - **A Records (Glue Records)**: Provides the IP addresses of the nameservers listed in the NS records to aid in resolving the next query step efficiently.



```
root_database = {
    "com": {
        "NS": [{"value": b"ns1.tld-com.server", "ttl": 3600}],
        "A": [{"name": "ns1.tld-com.server", "value": "127.0.0.1", "ttl": 3600}]
    },
    "org": {
        "NS": [{"value": b"ns1.tld-org.server", "ttl": 3600}],
        "A": [{"name": "ns1.tld-org.server", "value": "127.0.0.1", "ttl": 3600}]
    },
    "net": {
        "NS": [{"value": b"ns1.tld-net.server", "ttl": 3600}],
        "A": [{"name": "ns1.tld-net.server", "value": "127.0.0.1", "ttl": 3600}]
    }
}
```

2. **TLD Server**:
   - **NS Records**: Lists the authoritative nameservers for domains within the TLD (e.g., example.com under .com).
   - **A Records (Glue Records)**: Provides the IP addresses of the nameservers listed in the NS records.

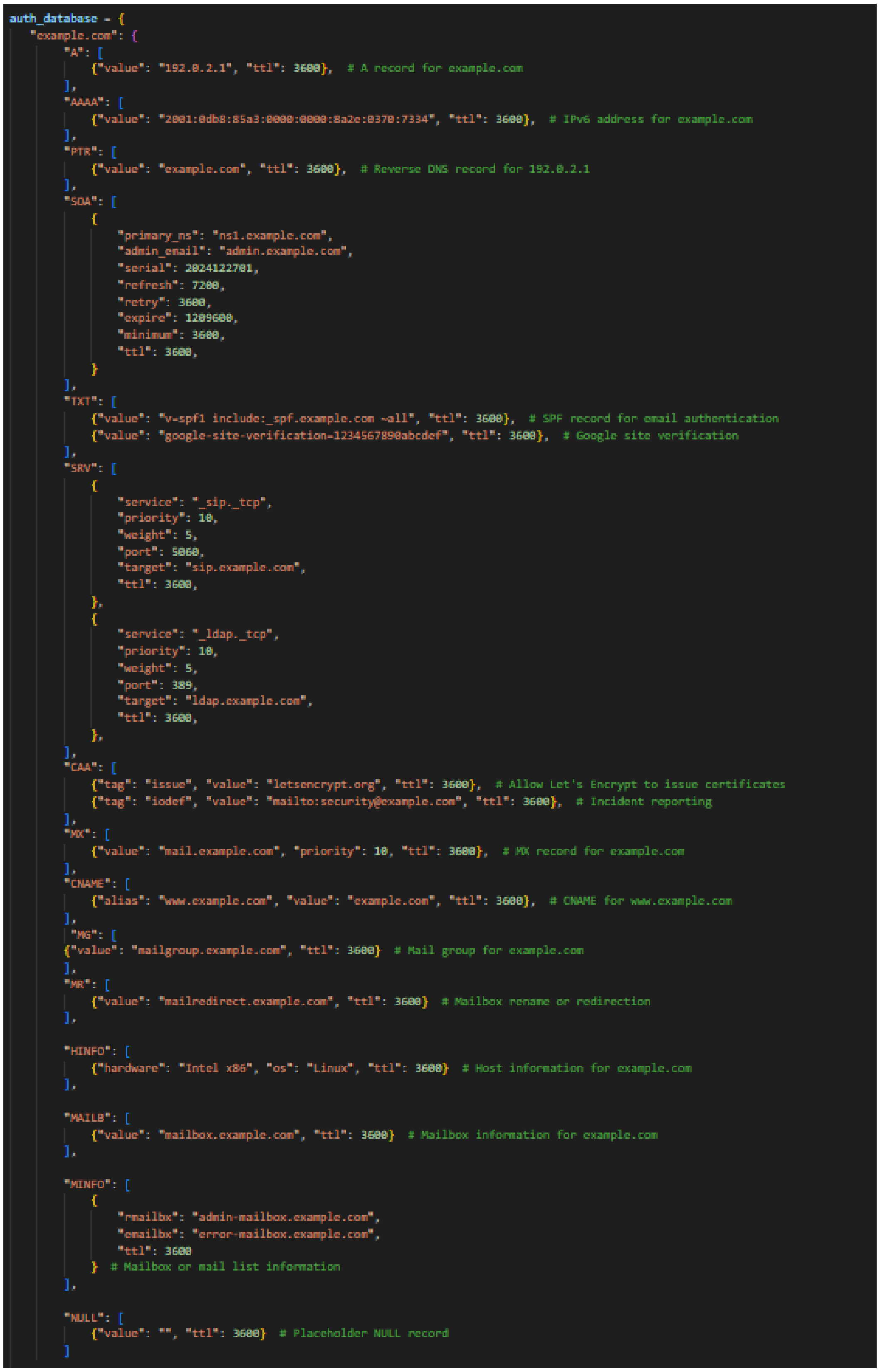


```
tld_database = {
    "example.com": {
        "NS": [
            {"value": b"ns1.auth-example.com", "ttl": 3600},
        ],
        "A": [
            {"name": "ns1.auth-example.com", "value": "127.0.0.1", "ttl": 3600},
        ]
    },
    "example.org": {
        "NS": [
            {"value": b"ns1.auth-example.org", "ttl": 3600},
        ],
        "A": [
            {"name": "ns1.auth-example.org", "value": "127.0.0.1", "ttl": 3600},
        ]
    },
    "example.net": {
        "NS": [
            {"value": b"ns1.auth-example.net", "ttl": 3600},
        ],
        "A": [
            {"name": "ns1.auth-example.net", "value": "127.0.0.1", "ttl": 3600},
        ]
    }
}
```

3. **Authoritative Server**:
    - **A Records**: Maps domain names to their corresponding IPv4 addresses.
    - **AAAA Records**: Maps domain names to their corresponding IPv6 addresses.
    - **MX Records**: Specifies the mail exchange servers for the domain.
    - **CNAME Records**: Provides aliasing information for the domain, mapping one domain name to another.
    - **PTR Records**: Used for reverse DNS lookup, mapping an IP address to a domain name.
    - **SOA Records**: Specifies the Start of Authority for a domain, containing administrative details such as the primary DNS server, the admin's email address, and various timers (refresh, retry, expire).
    - **TXT Records**: Stores arbitrary text data, often used for purposes like SPF (Sender Policy Framework) records or domain verification (e.g., Google, Facebook).
    - **SRV Records**: Specifies information about available services, including the target domain, port, and priority (e.g., for SIP, LDAP, or other services).
    - **CAA Records**: Specifies which certificate authorities (CAs) are allowed to issue SSL/TLS certificates for the domain.

- **NS Records**: Specifies the authoritative name servers for the domain.
- **MG Records:** Specifies a mailbox group or mail list. It provides information about a group of mailboxes that receive mail sent to a specific domain. Rarely used in modern implementations.
- **MR Records:** Used to specify a new domain for a renamed mailbox. This record type helps redirect email sent to an old domain to a new one. Rarely used today.
- **HINFO Records:** Provides information about the host, including the hardware type (CPU) and operating system. This record is largely informational and rarely used due to privacy and security concerns.
- **MAILB Records:** Indicates a mailbox domain name. This record type is obsolete and was defined in older DNS specifications (RFC 1035).
- **MINFO Records:** Specifies mailbox or mail list information. It contains two domain names:
  **Responsible mailbox (RMAILBX):** Indicates the mailbox responsible for the domain or mailing list.
  **Error mailbox (EMAILBX):** Indicates a mailbox to report errors related to the domain or mailing list.
- **NULL Records:** A record type that is defined but does not contain any information. It was originally intended for experimental purposes.

Screenshot for example.com, same is done for example.net & example.org:



```
auth_database = {
    "example.com": {
        "A": [
            {"value": "192.0.2.1", "ttl": 3600},  # A record for example.com
        ],
        "AAAA": [
            {"value": "2001:0db8:85a3:0000:0000:8a2e:0370:7334", "ttl": 3600},  # IPv6 address for example.com
        ],
        "PTR": [
            {"value": "example.com", "ttl": 3600},  # Reverse DNS record for 192.0.2.1
        ],
        "SOA": [
            {
                "primary_ns": "ns1.example.com",
                "admin_email": "admin.example.com",
                "serial": 2024122701,
                "refresh": 7200,
                "retry": 3600,
                "expire": 1209600,
                "minimum": 3600,
                "ttl": 3600,
            }
        ],
        "TXT": [
            {"value": "v=spf1 include:_spf.example.com ~all", "ttl": 3600},  # SPF record for email authentication
            {"value": "google-site-verification=1234567890abcdef", "ttl": 3600},  # Google site verification
        ],
        "SRV": [
            {
                "service": "_sip._tcp",
                "priority": 10,
                "weight": 5,
                "port": 5060,
                "target": "sip.example.com",
                "ttl": 3600,
            },
            {
                "service": "_ldap._tcp",
                "priority": 10,
                "weight": 5,
                "port": 389,
                "target": "ldap.example.com",
                "ttl": 3600,
            },
        ],
        "CAA": [
            {"tag": "issue", "value": "letsencrypt.org", "ttl": 3600},  # Allow Let's Encrypt to issue certificates
            {"tag": "iodef", "value": "mailto:security@example.com", "ttl": 3600},  # Incident reporting
        ],
        "MX": [
            {"value": "mail.example.com", "priority": 10, "ttl": 3600},  # MX record for example.com
        ],
        "CNAME": [
            {"alias": "www.example.com", "value": "example.com", "ttl": 3600},  # CNAME for www.example.com
        ],
         "MG": [
        {"value": "mailgroup.example.com", "ttl": 3600}  # Mail group for example.com
        ],
        "MR": [
            {"value": "mailredirect.example.com", "ttl": 3600}  # Mailbox rename or redirection
        ],

        "HINFO": [
            {"hardware": "Intel x86", "os": "Linux", "ttl": 3600}  # Host information for example.com
        ],

        "MAILB": [
            {"value": "mailbox.example.com", "ttl": 3600}  # Mailbox information for example.com
        ],

        "MINFO": [
            {
                "rmailbx": "admin-mailbox.example.com",
                "emailbx": "error-mailbox.example.com",
                "ttl": 3600
            }  # Mailbox or mail list information
        ],

        "NULL": [
            {"value": "", "ttl": 3600}  # Placeholder NULL record
        ]
```

## Advantages of Architecture

1. **Modularity**:
   - Each component (resolver, root, TLD, authoritative) operates independently, making it easy to debug and scale.
2. **Scalability**:
   - The hierarchical structure supports adding more TLD and authoritative servers without affecting the resolver or root server.
3. **Standards Compliance**:
   - The architecture adheres to the principles of DNS as outlined in RFC 1034 and 1035.
4. **Error Resilience**:
   - Each server handles errors gracefully, ensuring reliable query resolution.

# UDP Communication Protocol

## Introduction to UDP

The User Datagram Protocol (UDP) is a connectionless and lightweight transport layer protocol defined in **RFC 768**. Unlike TCP, UDP does not establish a connection before data transfer, making it faster and more efficient for small, time-sensitive transmissions. UDP operates by encapsulating data into datagrams, which are sent directly to the recipient without acknowledgment or retransmission. While this simplicity sacrifices reliability and error correction, it significantly reduces latency and overhead, making UDP ideal for applications where speed and low latency are critical.

## Why use UDP in DNS

The Domain Name System, as specified in **RFC 1035**, primarily relies on UDP for communication between clients, resolvers, and servers. The decision to use UDP for DNS is rooted in the following advantages:

1. **Speed and Efficiency**:
   - UDP enables fast query and response cycles without the overhead of establishing a connection.
   - DNS queries are typically small (less than 512 bytes in legacy implementations) and well-suited to UDP's datagram structure.

2. **Low Overhead**:
    - DNS servers handle millions of queries per day. UDP’s lightweight nature minimizes the resource requirements for both clients and servers.
3. **Stateless Communication**:
    - UDP's connectionless design allows DNS servers to handle multiple queries without maintaining state, enhancing scalability and reducing complexity.
4. **Standards Compliance**:
    - **RFC 1035** explicitly specifies UDP as the default protocol for DNS query transmission. TCP is only used for specific scenarios, such as zone transfers or when the response exceeds UDP's size limitations.

The use of UDP in this project reflects its critical role in DNS communication as established by **RFC 1035**. Its speed, efficiency, and statelessness make it the optimal protocol for DNS query resolution, balancing performance with simplicity while adhering to established standards.

# DNS Resolver

The DNS Resolver is a critical component in the Domain Name System (DNS) architecture. It serves as the intermediary between clients making DNS queries and the hierarchy of DNS servers (Root, TLD, and Authoritative). The resolver processes client queries, communicates with the appropriate servers, and returns the resolved data to the client. This section outlines the resolver's responsibilities, design, and implementation.

## Responsibilities

1. Receive and validate DNS queries from clients.
2. Query the Root Server to retrieve the nameserver information for the relevant TLD.
3. Query the appropriate TLD Server to fetch the nameserver details for the domain's authoritative server.
4. Query the Authoritative Server to retrieve the required DNS records for the client.
5. Handle and relay error responses, such as "Non-Existent Domain", "Unsupported Query Type" , “Server Failure”, ”Format Error in Query” or “Policy Restriction”
6. Extract critical data (e.g., IP addresses) from the responses to facilitate subsequent queries.

## Implementation

1. **Socket Setup:**
    - The resolver operates on 127.0.0.1 at port 53 to listen for incoming DNS queries.
    - It uses UDP sockets for fast, connectionless communication with clients and servers.

2. **Query Handling Logic:**
    - **Root Query:** The resolver sends the client's query to the Root Server (127.0.0.1:5354) and extracts the IP address of the TLD Server from the additional section of the response.
    - **TLD Query:** The resolver sends a query to the extracted TLD Server and retrieves the authoritative server's IP address.
    - **Authoritative Query:** The resolver queries the authoritative server and receives the final DNS record.
    - **Response Handling:** Each response is parsed to extract relevant data or handle errors (e.g., NXDOMAIN, NOTIMP).
3. **Functions:**
    - **extract_ip_from_response(response):**
      This function parses a DNS response to extract the IPv4 address from the Additional Section, specifically from an A (IPv4) record. It first skips the DNS header (12 bytes) and iterates through the Question Section, skipping over the domain name, query type, and query class fields. Then, it processes the Answer and Authority Sections by accounting for compressed or uncompressed domain names, skipping over the fields for type, class, TTL, and RDATA length. Finally, it parses the Additional Section, where it checks for A records (type 1). When it encounters an A record, it converts its RDATA (IPv4 address in bytes) into dotted-decimal format and returns the resulting IP address. If no A record is found in the Additional Section, the function returns None. This function is essential for extracting useful information from DNS responses, particularly when handling glue records or additional resource records that include IP addresses.
    - **check_for_error(response):**
      This function analyzes the DNS response to identify any errors based on the Response Code (Rcode), which is contained in the last 4 bits of the flags field (bytes 2 and 3 of the response). The function converts the two-byte flags into a 16-bit integer and isolates the Rcode using a bitwise AND operation with 0b1111. It then interprets the Rcode value and prints an appropriate message indicating the type of error, if any. Specifically, it checks for the following cases: 0 (No error), 1 (Format error in the query), 2 (Server failure), 3 (Non-existent domain), 4 (Unsupported query type), and 5 (Policy restriction). This function is crucial for debugging and understanding the status of DNS queries and responses, as it provides insights into any issues encountered during query processing. It returns the original response unchanged, allowing further processing if needed.
    - **handle_query(data, addr):**
      This function is a key component of a DNS resolver that processes client queries by interacting sequentially with the root, TLD, and authoritative servers. Initially, it extracts the domain name from the client's query using the get_binary_domain

function for debugging.
The function then forwards the query to the root server and receives its response. This response is analyzed using the check_for_error function to identify any issues, such as non-zero Rcodes, that would indicate errors in the query. If errors are detected, the root server's response is relayed back to the client, and the process halts.
If the root server's response is valid, the function extracts the IP address of the TLD server using the extract_ip_from_response function. The query is sent to the TLD server, and its response undergoes the same validation process as the root server's response. In case of errors, the response is sent to the client, and further processing is stopped. Otherwise, the function proceeds to the next step.
The resolver then sends the query to the authoritative server, whose IP address is retrieved from the TLD server's response. The authoritative server's response is validated for errors, and any issues are promptly communicated back to the client.
Finally, if all responses are valid, the resolver sends the authoritative server's response back to the client. Throughout the process, debugging information, such as the binary domain name, server IPs, and server responses, is logged to assist in monitoring and diagnosing issues during query resolution. This structured approach ensures reliable handling of DNS queries with robust error detection and debugging support.

4. **Error Management:**
    - Identifies errors in responses using the RCODE field and relays them to the client.
    - **No Error (RCODE = 0)**
    - **Format Error (RCODE = 1):** Indicates an invalid query format received by the server.
    - **Server Failure (RCODE = 2):** Indicates that the server was unable to process the query due to an internal error.
    - **Non-Existent Domain (RCODE = 3):** Indicates the queried domain does not exist in the server's database.
    - **Not Implemented (RCODE = 4):** Indicates the query type is not supported by the server.
    - **Refused (RCODE = 5):** Indicates that the server refuses to process the query for policy reasons.

## Key features

1. **Standard Compliance:** Fully adheres to RFC 1034 and RFC 1035 specifications for DNS message structure and query resolution.
2. **Dynamic Query Chaining:** Dynamically processes and forwards queries through the Root, TLD, and Authoritative servers.
3. **Simple and Modular Design:** Functions are modular, focusing on individual tasks like query parsing, response handling, and error reporting.
4. **Customizable:** The resolver can be easily adapted to support additional DNS record types or custom configurations.
5. **Threaded Architecture:** Uses threading to handle multiple client requests concurrently.
6. **Monitoring with Logging:** Integrated comprehensive logging using INFO and WARNING levels, enabling efficient debugging and monitoring of server activities.

# Root Server

The Root Server is a foundational component in the Domain Name System (DNS) simulation. It serves as the first point of contact for DNS queries, directing resolvers to the appropriate Top-Level Domain (TLD) servers. The Root Server responds with authoritative records for TLDs, such as nameserver (NS) records, and additional A records for those nameservers when available. It operates using UDP and processes requests asynchronously with threading.

## Responsibilities

1. Maintain and manage a database of NS and A records for all supported TLDs.
2. Respond to queries with authoritative NS records for the requested TLD.
3. Provide additional A records for the NS servers in the additional section of the response, if available.
4. Validate incoming DNS queries to ensure compliance with DNS specifications.
5. Handle error cases such as unsupported query types (NOTIMP) or non-existent TLDs (NXDOMAIN).
6. Generate and send error responses to the resolver or client for invalid or unresolvable queries.

## Implementation

1. **Socket Setup:**
    - **IP Address**: 127.0.0.1
    - **Port**: 5354
    - **Protocol**: UDP for efficient communication without persistent connections.
2. **Query Handling Logic:**

- **Query Validation**:
  - Ensures the query is well-formed and conforms to DNS protocol specifications.
  - Rejects invalid queries with a FORMERR response.
- **Database Lookup**:
  - Retrieves NS records for the requested TLD.
  - Includes A records for the nameservers in the additional section if available.
- **Response Construction**:
  - Constructs a DNS response including:

    **Header**: Transaction ID, flags, question count, answer count, authority count, and additional count.

    **Question Section**: Echoes the query.

    **Authority Section**: Contains NS records for the requested TLD.

    **Additional Section**: Contains A records for the nameservers (if available).

3. **Functions:**
   - **encode_domain_name(domain):**
     This function converts a domain name into its binary format as required by DNS protocols. It splits the domain into labels using the period (.) as a delimiter, then processes each label by prefixing it with a single byte that specifies its length, followed by the label itself encoded in UTF-8. These encoded labels are concatenated to form the binary representation of the domain, and a null byte (b'\0') is appended to indicate the end of the name. This ensures the domain name is correctly formatted for use in DNS queries, adhering to the specification where each label is length-prefixed and the sequence ends with a null byte.

   - **convert_records_to_binary(records):**
     This function is responsible for converting a list of DNS record dictionaries into a concatenated binary format. It iterates through each record, encoding various components such as the domain name (TLD), record type, class, time-to-live (TTL), and RDATA (typically the nameserver value) into binary representations. The function uses specific byte orderings (big-endian) for each component, including the length of the RDATA field. After encoding each part, it combines them into a single binary record, appends them to a bytearray, and finally returns the entire sequence of binary records as a bytes object. This process enables efficient handling and transmission of DNS records in binary format.

   - **get_tld(domain_name):**
     This function extracts the top-level domain (TLD) from a given domain name. It splits the domain name into individual parts using the dot (.) as a delimiter, creating a list

of domain labels. The TLD is then identified as the last part of the list (i.e., the last element). The function returns this last part, which represents the TLD, such as "com", "org", or "net".

- **get_records(tld):**
  This function retrieves the nameserver (NS) records for a given top-level domain (TLD) from the root_database. It first checks if the TLD exists in the root_database. If found, it extracts the associated NS records, which contain the nameserver's value and TTL (time to live). For each NS record, the function creates a dictionary representing the record with the following fields: name (encoded TLD), type (set to 2, which denotes an NS record), class (set to 1, representing the IN class), ttl (time to live for the record), and rdata (the nameserver value). These dictionaries are collected in a list, which is then returned as the function's output. If the TLD is not found in the root_database, the function returns an empty list, indicating that no records were found.

- **get_question_domain(data):**
  This function extracts the domain name and query type from a DNS query data packet. It starts by initializing an empty list to store domain name parts and a dictionary (query_type_map) to map query type codes (such as 1 for 'A', 28 for 'AAAA', etc.) to their corresponding string representations. The function skips the first 12 bytes of the data, which are reserved for the DNS header, and then iterates through the remaining data to extract the domain name. Each domain label is preceded by a byte that indicates its length, and the labels are decoded from bytes to strings and added to the domain_parts list. After all domain labels are extracted, they are joined together with dots to form the complete domain name. The function then reads the next two bytes to determine the query type, converting it into its string representation using the query_type_map. Finally, it returns the domain name and query type as a tuple.

- **get_flags(rcode):**
  This function constructs the DNS flags section of a DNS response based on the provided RCODE (Response Code). It starts by setting predefined values for various flag fields: QR (Query/Response) is set to '1' (indicating a response), opcode is '0000' (standard query), AA (Authoritative Answer) is '0', TC (Truncated) is '0', RD (Recursion Desired) is '0', RA (Recursion Available) is '0', and Z (reserved bits) is '000'. The RCODE is appended to the end of this binary string. The function then concatenates all these flag components into a single binary string, converts it to an integer, and then converts the integer to a two-byte binary format (using big-endian byte order). Finally, it returns the flags as a byte representation, which can be used in the DNS response.

- **get_rcode(tld, query_type, data):**
  Thisfunction is designed to determine the appropriate DNS response code (RCODE) based on the provided TLD (top-level domain), query type, and data. It first validates

the format of the query data using the validate_query_format function, returning an RCODE of 1 (FORMERR) if the data is invalid. Next, it checks if the query type is supported (e.g., 'A', 'CNAME', 'MX', etc.), returning 4 (NOTIMP) if the query type is unsupported. The function then checks if records exist for the given TLD using get_records. If no records are found, it returns 3 (NXDOMAIN), indicating that the domain does not exist. If all checks pass, it returns 0 (NOERROR), signaling that the query is valid and successful. Additionally, the function contains comments referencing other potential RCODEs for server failure (2) and policy restriction (5).

- **validate_query_format(data):**
  This function is designed to validate the structure of a DNS query. It first checks if the length of the query is at least 12 bytes, as the DNS header requires this minimum size; if the query is too short, a warning is logged, and the function returns False. Next, the function verifies that the query is not a response by checking the QR (Query/Response) bit in the DNS header. If the QR bit is set (indicating a response), a warning is logged, and the function returns False. The function then checks the number of questions in the query by inspecting the 5th and 6th bytes of the data, which represent the number of questions in big-endian format. If the number of questions is not exactly one, it logs a warning and returns False. If all checks pass, the function returns True, indicating that the query format is valid
- **build_response(data):**
  This function constructs a DNS response based on the incoming query data. It first extracts the domain name and query type from the question section using the get_question_domain function, and then it determines the top-level domain (TLD) using get_tld. It retrieves the corresponding NS records for the TLD via the get_records function. The function then extracts the transaction ID from the original query and generates the appropriate RCODE (Response Code) using the get_rcode function. The RCODE is used to generate DNS flags with the get_flags function. If the RCODE indicates a successful query ('0000'), the function constructs the DNS response by including the question section, authority section (converted from the NS records), and additional section (via get_additional_section). It also calculates the counts for the Question, Answer, Authority, and Additional sections in the DNS header. The response is composed by concatenating the DNS header, question section, authority section, and additional section. If the RCODE indicates an error, the function generates an error response using build_error_response. The constructed response is then returned.

- **build_error_response(data, rcode**):
  This function constructs a DNS error response when a query cannot be processed successfully. It begins by extracting the transaction ID from the original query to ensure the response is associated with the correct request. The function then generates the DNS flags based on the provided RCODE (Response Code) using the get_flags function. The header for the error response is constructed with the transaction ID, flags, and counts for the Question, Answer, Authority, and Additional sections, all set to their appropriate values (1 question, 0 answers, 0 authority, and 0

additional records). The question section of the original query is echoed in the response, which ensures the response maintains the same format as the query. Finally, the function combines the header and question section into the complete error response and returns it.

4. **Error Management:**
   - Identifies errors in responses using the RCODE field and relays them to the client.
   - **No Error (RCODE = 0)**
   - **Format Error (RCODE = 1):** Indicates an invalid query format received by the server.
   - **Server Failure (RCODE = 2):** Indicates that the server was unable to process the query due to an internal error.
   - **Non-Existent Domain (RCODE = 3):** Indicates the queried domain does not exist in the server's database.
   - **Not Implemented (RCODE = 4):** Indicates the query type is not supported by the server.
   - **Refused (RCODE = 5):** Indicates that the server refuses to process the query for policy reasons.

## Key features

1. **Standard Compliance**: Fully adheres to RFC 1034 and RFC 1035 specifications for DNS message formats and query resolution.
2. **Authoritative Responses**: Provides authoritative NS records for all supported TLDs.
3. **Dynamic Additional Records**: Includes A records for NS servers in the additional section, when available.
4. **Error Handling**: Comprehensive handling of various error scenarios with appropriate DNS error codes.
5. **Threaded Design**: Supports concurrent query handling using a multi-threaded architecture.
6. **Customizable**: The server's database and functionality can be extended to support additional TLDs or record types.
7. **Monitoring with Logging:** Integrated comprehensive logging using INFO and WARNING levels, enabling efficient debugging and monitoring of server activities.

# TLD Server

The Top-Level Domain (TLD) server is a critical part of the Domain Name System (DNS) simulation. It responds to DNS queries for specific domains by providing authoritative records, such as nameserver (NS) records, for domains under its control. This server operates using UDP and handles requests asynchronously through threading.

## Responsibilities

1. Maintain and manage a database of NS and A records for domains under its TLD.
2. Respond to queries with authoritative NS records for the requested domain.
3. Provide additional A records for the NS servers in the additional section of the response when available.
4. Validate incoming DNS queries to ensure compliance with DNS specifications.
5. Handle error cases, such as non-existent domains (NXDOMAIN) or unsupported query types (NOTIMP).
6. Generate and send error responses to the resolver or client for invalid or unresolvable queries.

## Implementation

1. **Socket Setup**:
    - The TLD Server listens on IP address 127.0.0.1 and port 5356.
    - It uses UDP sockets to process DNS queries and send responses efficiently without establishing a persistent connection.
2. **Query Handling Logic**:
    - **Query Validation**: The server verifies the structure and validity of incoming DNS queries. Invalid queries are rejected with a FORMERR response.
    - **Database Lookup**:
        - For valid queries, the server looks up NS records for the requested domain.
        - If available, A records corresponding to the NS records are included in the additional section.
    - **Response Construction**: A DNS response is constructed containing:
        - Transaction ID, flags, and other header fields.
        - The question section echoing the query.
        - Authority section with NS records.

    - Additional section with A records for the nameservers (if available).

3. **Functions**:

    - **encode_domain_name(domain)**:
      same implementation as root server
    - **convert_records_to_binary(records)**:
      same implementation as root server
    - **get_records(domain)**:
      This function retrieves nameserver (NS) records for a given domain from the tld_database. It first checks if the domain exists in the database. If the domain is found, it extracts the associated NS records, which consist of a list of dictionaries containing the nameserver values (value) and time-to-live (TTL) values (ttl). For each NS record, the function creates a dictionary representing the record, with fields for name (the domain itself), type (set to 2, indicating an NS record), class (set to 1, indicating the IN class), ttl (the TTL in seconds), and rdata (the nameserver value). These dictionaries are collected in a list, which is returned as the function's output. If the domain is not found in the tld_database, the function returns an empty list, indicating no records are available.
    - **get_question_domain(data)**:
      same implementation as root server
    - **get_flags(Rcode)**:
      same implementation as root server
    - **validate_query_format(data)**:
      same implementation as root server
    - **get_rcode(domain_name, query_type, data):**
      This function handles DNS query validation and response code generation based on several conditions. It first checks whether the provided query data is valid using the validate_query function, returning a response code of 1 (FORMERR) if the validation fails. Next, it ensures that the query_type is among a predefined list of accepted types (e.g., 'A', 'CNAME', 'MX', etc.). If the query type is not valid, it returns 4 (NOTIMP). The function then checks if records for the specified domain exist using the get_records function, returning 3 (NXDOMAIN) if no records are found. If all checks pass, it returns 0 (NOERROR). The function also includes comments indicating potential response codes for server failure (2) and policy restrictions (5).
    - **build_response(data)**:
      This function constructs a DNS response based on the incoming query data. It first extracts the domain name and query type from the question section using the get_question_domain function and retrieves the corresponding NS records with the get_records function. The transaction ID from the original query is preserved for the response. The function determines the appropriate RCODE (Response Code) using the get_rcode function and generates the DNS flags with the get_flags function.

If the RCODE indicates a successful query ('0000'), the function builds the DNS response by echoing the question section from the original query. It then creates the authority section by converting the NS records to binary using convert_records_to_binary and generates the additional section with get_additional_section. The counts for the Question, Answer, Authority, and Additional sections are calculated. These counts and the DNS header (which includes the transaction ID, flags, and section counts) are combined, followed by the question, authority, and additional sections to form the full response.
If the RCODE indicates an error, the function constructs an error response using build_error_response and returns it instead. The final DNS response, whether successful or error, is then returned by the function.

- **build_error_response(data, rcode)**:
  same implementation as root server

4. **Error Management**:
   - Identifies errors in responses using the RCODE field and relays them to the client.
   - **No Error (RCODE = 0)**
   - **Format Error (RCODE = 1):** Indicates an invalid query format received by the server.
   - **Server Failure (RCODE = 2):** Indicates that the server was unable to process the query due to an internal error.
   - **Non-Existent Domain (RCODE = 3):** Indicates the queried domain does not exist in the server's database.
   - **Not Implemented (RCODE = 4):** Indicates the query type is not supported by the server.
   - **Refused (RCODE = 5):** Indicates that the server refuses to process the query for policy reasons.

## Key features

1. **Standard Compliance**: Fully adheres to RFC 1034 and RFC 1035 specifications for DNS message formats and query resolution.
2. **Authoritative Responses**: Provides authoritative NS records for domains under its TLD.
3. **Dynamic Additional Records**: Includes corresponding A records for NS servers in the additional section when available.
4. **Error Handling**: Comprehensive handling of various error scenarios with appropriate DNS error codes.
5. **Threaded Design**: Supports concurrent query handling using a multi-threaded architecture.

6. **Customizable**: The server's database and functionality can be extended to support more TLDs or record types.

7. **Monitoring with Logging:** Integrated comprehensive logging using INFO and WARNING levels, enabling efficient debugging and monitoring of server activities.

# Authoritative Server

The Authoritative Server is a crucial component of the Domain Name System (DNS) simulation. It responds to DNS queries with authoritative answers, including resource records (e.g., A, MX, CNAME) for domains under its control. This server operates over UDP and handles multiple requests concurrently using threading.

## Responsibilities

1. Maintain and manage a database of DNS records, such as A, MX, and CNAME records, for domains under its authority.

2. Respond to queries with authoritative resource records for the requested domain and record type.

3. Validate incoming DNS queries for correct structure and format.

4. Handle error cases such as unsupported query types (NOTIMP) and non-existent domains (NXDOMAIN).

5. Construct and send appropriate DNS responses or error messages to the client.

## Implementation

1. **Socket Setup:**
   - The Authoritative Server listens on IP address 127.0.0.1 and port 5357.
   - It uses UDP sockets for stateless communication, enabling efficient handling of DNS queries.

2. **Query Handling Logic:**
   - **Query Validation:** Verifies the structure and compliance of incoming DNS queries with DNS protocol standards.
   - **Database Lookup:** Searches for records (A, MX, CNAME, etc.) in the database for valid queries.
   - **Response Construction:** Builds DNS responses containing:
     - Transaction ID, flags, and header fields.

- Question section echoing the query.
- Answer section with requested resource records.
- Additional sections for supplementary information (e.g., related records).

3. **Functions:**
   - **get_records(domain_name, record_type):**
     This function retrieves specific DNS records for a given domain name and record type from the auth_database. It first checks if the domain name exists in the database and if the specified record type (such as 'A', 'NS', etc.) is associated with that domain. If both conditions are met, it returns the corresponding records for the given domain name and record type. If the domain name or record type is not found in the database, the function returns an empty list, indicating that no records were found.

   - **build_records(domain_name, record_type, records):**
     This function constructs the DNS response body by encoding various types of DNS resource records in compliance with the DNS protocol's wire format. It iterates through a list of records for a specific domain and appends the corresponding binary data for each record type (e.g., A, AAAA, CNAME, NS, MX, PTR, SOA, TXT, SRV, CAA, and others). For each record, it encodes fields such as the domain name, record type, class, TTL (Time-To-Live), and record-specific data (e.g., IP addresses, hostnames, or text values). The function ensures proper formatting by converting data into byte representations, handling domain name labels, and adhering to the specifications for each record type. This allows the DNS server to generate accurate and standards-compliant responses.

   - **get_question_domain(data):**
     same implementation as Root server and TLD server

   - **get_flags(Rcode):**
     same implementation as Root server and TLD server

   - **get_rcode(domain_name, query_type, data):**
     same implementation as Root server and TLD server

   - **validate_query_format(data):**
     same implementation as Root server and TLD server

   - **build_response(data):**
     function constructs the DNS response message based on the original query (data) provided. It processes the query to generate the appropriate response by extracting

domain name and query type, getting corresponding records, and assembling the DNS response header, question section, and answer section. Here's a breakdown:
**Extract domain name and query type**: The function retrieves the domain name and query type using the get_question_domain function.
**Retrieve records**: The get_records function fetches the records based on the domain name and query type.
**Transaction ID**: The original transaction ID from the request is preserved.
**Rcode and flags**: The get_rcode and get_flags functions are used to determine the response code and flags.
**Answer count**: Based on the number of records returned, the answer count is set.
**DNS header**: The DNS header is constructed, including the transaction ID, flags, question count, answer count, authority count, and additional count.
**Question section**: The question section from the original request is echoed in the response.
**Answer section**: The build_records function is called to construct the answer section for the requested records.
**Error response**: If the response code is not "0000", an error response is returned using the build_error_response function.

The function returns the assembled DNS response in binary format.

- **build_error_response(data, rcode):**
  same implementation as Root server and TLD server

4. **Error Management:**
   - Identifies errors in responses using the RCODE field and relays them to the client.
   - **No Error (RCODE = 0)**
   - **Format Error (RCODE = 1):** Indicates an invalid query format received by the server.
   - **Server Failure (RCODE = 2):** Indicates that the server was unable to process the query due to an internal error.
   - **Non-Existent Domain (RCODE = 3):** Indicates the queried domain does not exist in the server's database.
   - **Not Implemented (RCODE = 4):** Indicates the query type is not supported by the server.
   - **Refused (RCODE = 5):** Indicates that the server refuses to process the query for policy reasons.

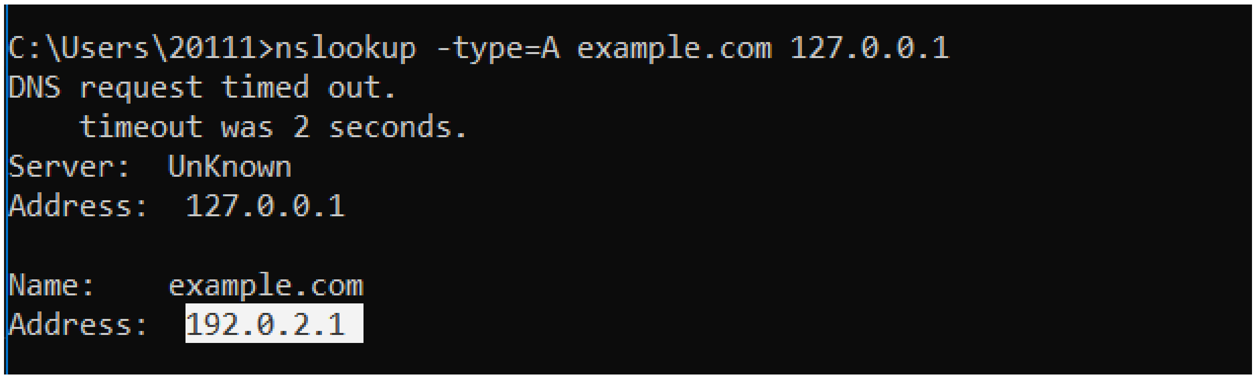
## Key Features

1. **Standard Compliance:** Fully adheres to DNS protocol specifications (RFC 1034, RFC 1035).
2. **Authoritative Responses:** Provides authoritative answers for all domains in its database.
3. **Resource Record Support**: Supports multiple record types, including A, MX, and CNAME.
4. **Error Handling:** Comprehensive error handling with appropriate RCODEs for various scenarios.
5. **Threaded Design:** Handles concurrent queries using a multi-threaded architecture for efficiency.
6. **Extensibility:** Easily extendable to support additional record types or domains.
7. **Monitoring with Logging:** Integrated comprehensive logging using INFO and WARNING levels, enabling efficient debugging and monitoring of server activities.

## Testing Scenarios

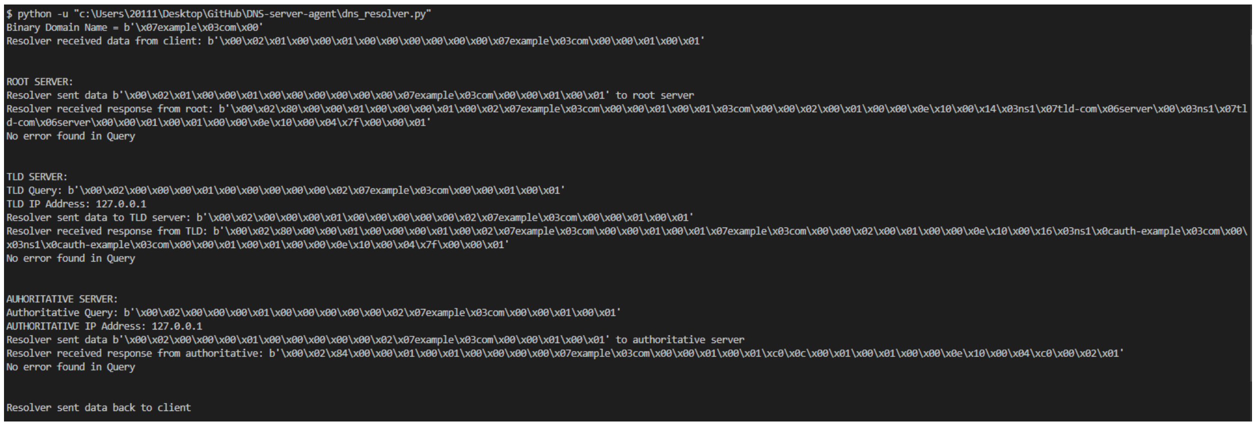
### Test Case 1

**Description:** Valid tld in root server's database (com)
**Input**: nslookup -type=A example.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=A example.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

Name:    example.com
Address:  192.0.2.1
```

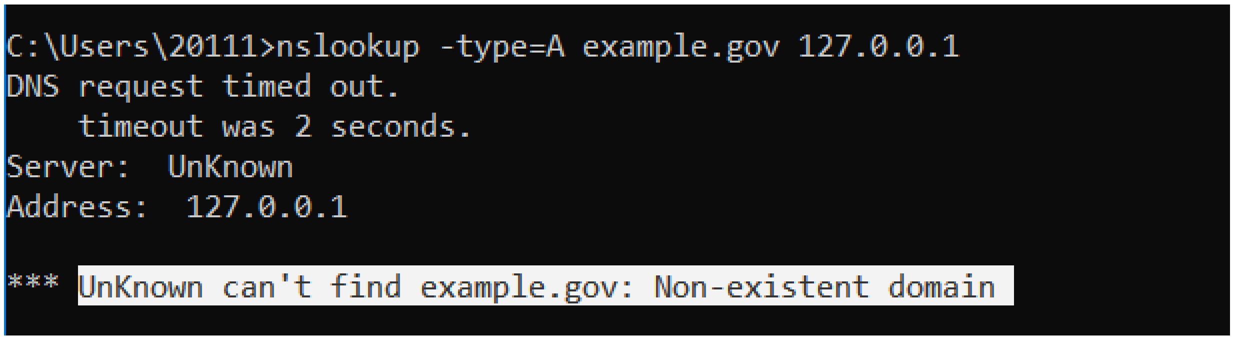
**DNS Resolver's Logging:**



```
$ python -u "c:\Users\20111\Desktop\GitHub\DNS-server-agent\dns_resolver.py"
Binary Domain Name = b'\x07example\x03com\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x01\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x01\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x01\x00\x01\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-com\x06server\x00\x03ns1\x07tl
d-com\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03com\x00\x00\x01\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03com\x00\x00\x01\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x01\x00\x01\x07example\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03com\x00\
x03ns1\x0cauth-example\x03com\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03com\x00\x00\x01\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03com\x00\x00\x01\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x07example\x03com\x00\x00\x01\x00\x01\xc0\x0c\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\xc0\x00\x02\x01'
No error found in Query


Resolver sent data back to client
```

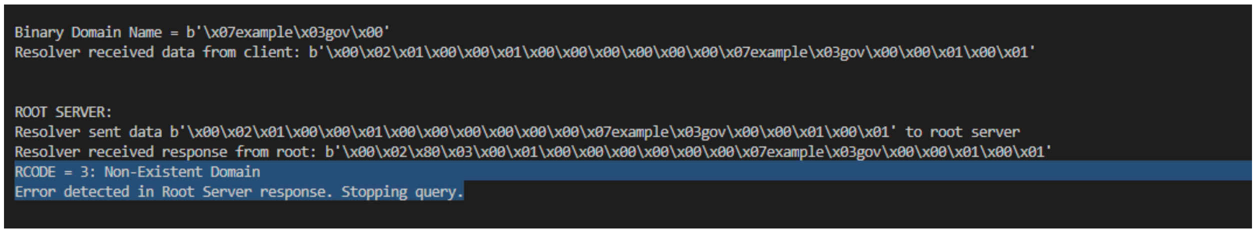
## Test Case 2

**Description:** Invalid tld in root server's database (gov)
**Input:** nslookup -type=A example.gov 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=A example.gov 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

*** UnKnown can't find example.gov: Non-existent domain
```

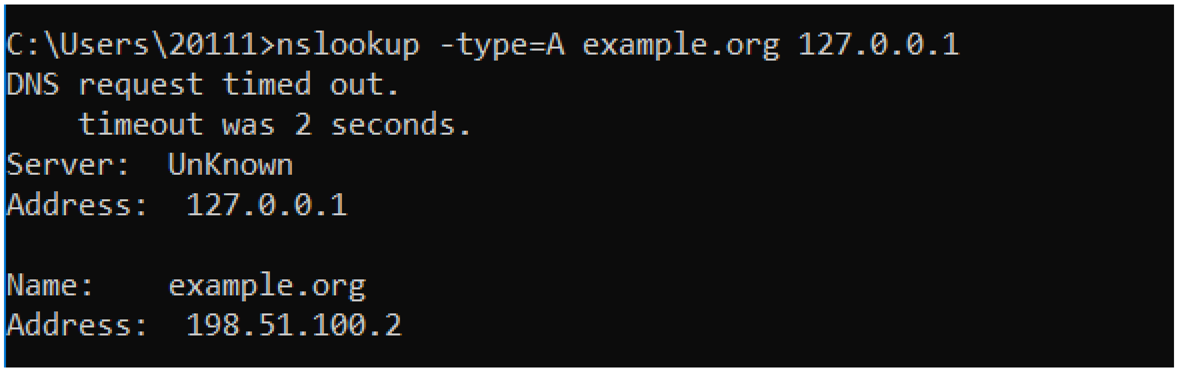
**DNS Resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03gov\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03gov\x00\x00\x01\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03gov\x00\x00\x01\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x03\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03gov\x00\x00\x01\x00\x01'
RCODE = 3: Non-Existent Domain
Error detected in Root Server response. Stopping query.
```

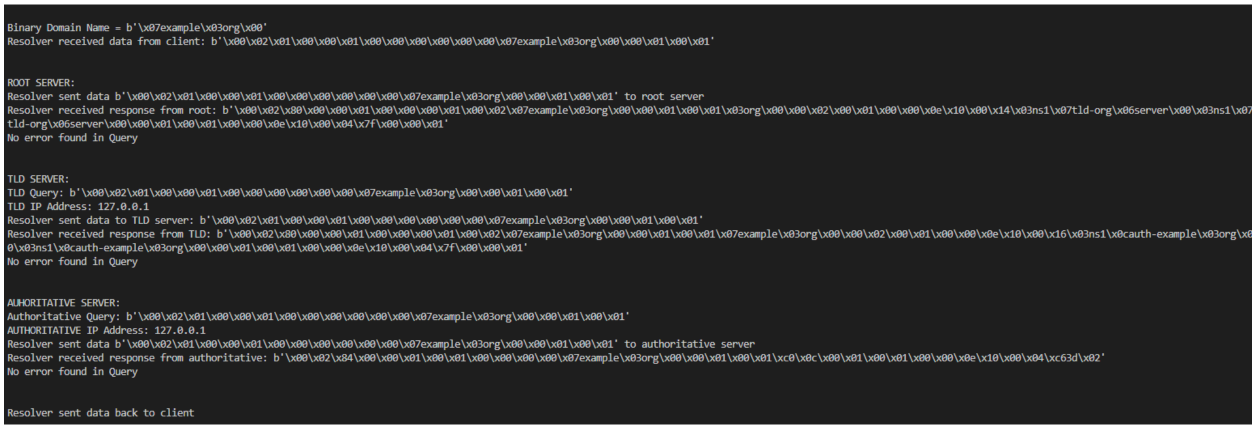
## Test Case 3

**Description:** Valid domain in TLD server's database (example.org)
**Input:** nslookup -type=A example.org 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=A example.org 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

Name:    example.org
Address:  198.51.100.2
```

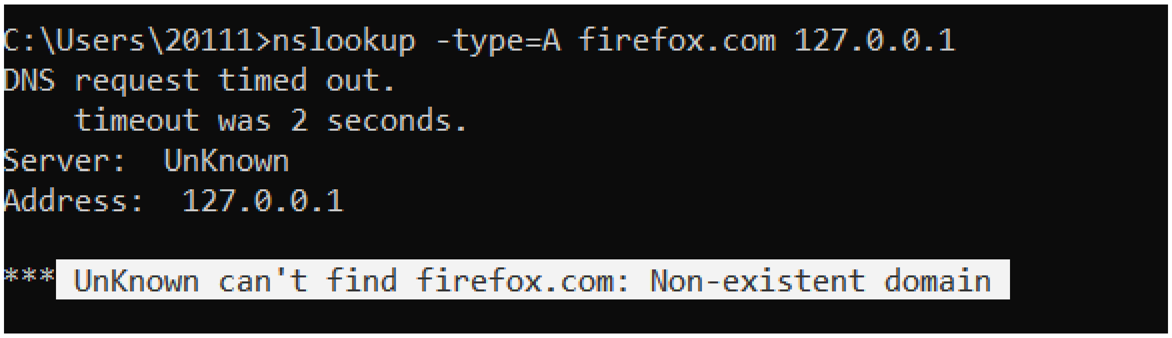
**DNS Resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03org\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00\x01\x00\x01\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-org\x06server\x00\x03ns1\x07
tld-org\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00\x01\x00\x01\x07example\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03org\x0
0\x03ns1\x0cauth-example\x03org\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x07example\x03org\x00\x00\x01\x00\x01\xc0\x0c\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\xc63d\x02'
No error found in Query


Resolver sent data back to client
```

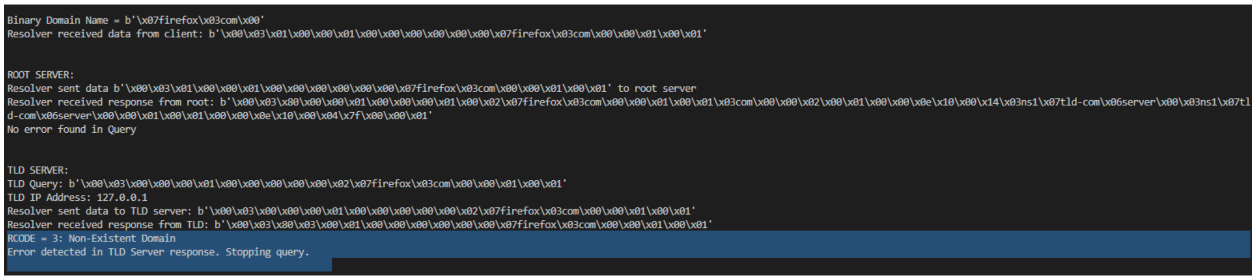
## Test Case 4

**Description:** Invalid domain in TLD server's database (firefox.com)
**Input:** nslookup -type=A fireforx.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=A firefox.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

*** UnKnown can't find firefox.com: Non-existent domain
```

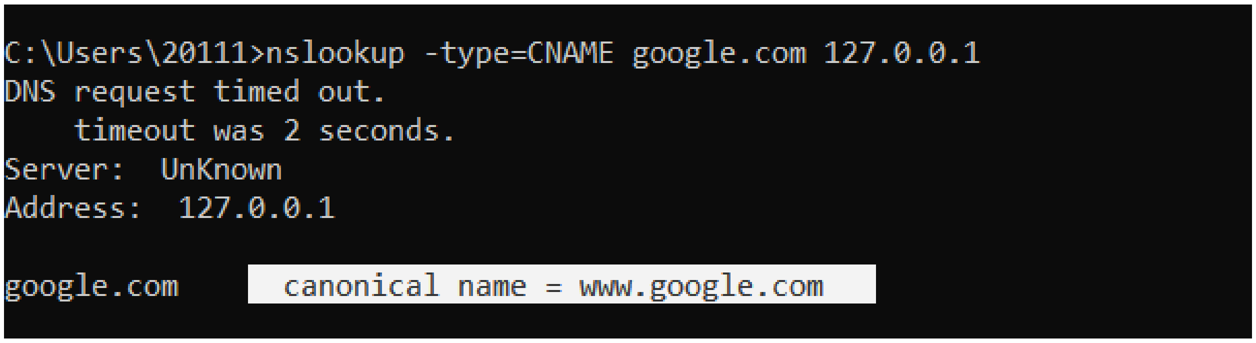
**DNS Resolver's Logging:**



```
Binary Domain Name = b'\x07firefox\x03com\x00'
Resolver received data from client: b'\x00\x03\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07firefox\x03com\x00\x00\x01\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x03\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07firefox\x03com\x00\x00\x01\x00\x01' to root server
Resolver received response from root: b'\x00\x03\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07firefox\x03com\x00\x00\x01\x00\x01\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-com\x06server\x00\x03ns1\x07tl
d-com\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x03\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07firefox\x03com\x00\x00\x01\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x03\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07firefox\x03com\x00\x00\x01\x00\x01'
Resolver received response from TLD: b'\x00\x03\x80\x03\x00\x01\x00\x00\x00\x00\x00\x00\x07firefox\x03com\x00\x00\x01\x00\x01'
RCODE = 3: Non-Existent Domain
Error detected in TLD Server response. Stopping query.
```

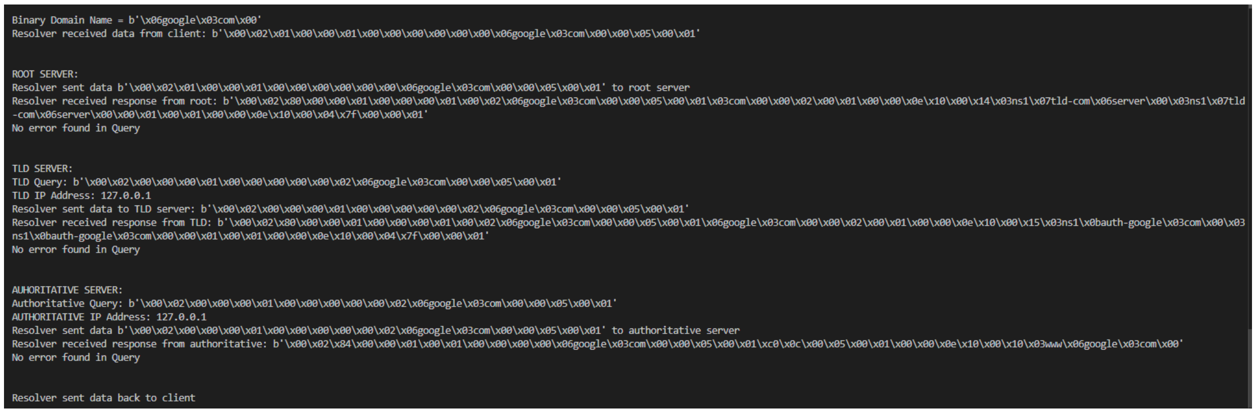
## Test Case 5

**Description:** Valid Query Type supported by DNS system ( CNAME )
**Input:** nslookup -type=CNAME google.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=CNAME google.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

google.com      canonical name = www.google.com
```

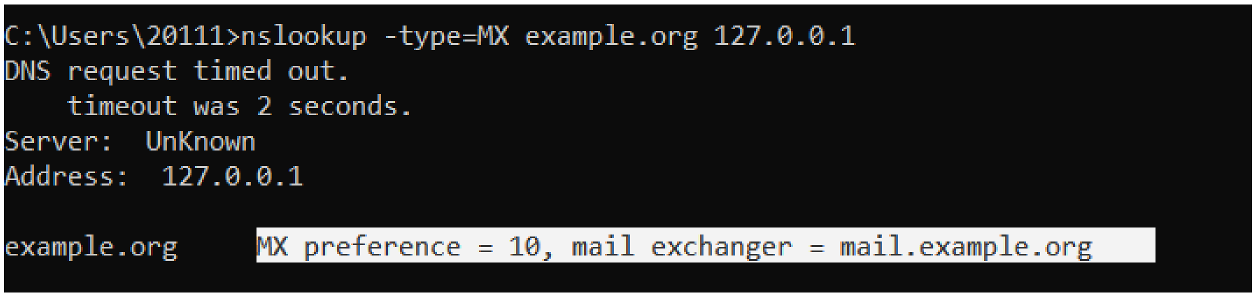
**DNS Resolver's Logging:**



```
Binary Domain Name = b'\x06google\x03com\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x06google\x03com\x00\x00\x05\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x06google\x03com\x00\x00\x05\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x06google\x03com\x00\x00\x05\x00\x01\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-com\x06server\x00\x03ns1\x07tld
-com\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x06google\x03com\x00\x00\x05\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x06google\x03com\x00\x00\x05\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x06google\x03com\x00\x00\x05\x00\x01\x06google\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x15\x03ns1\x0bauth-google\x03com\x00\x03
ns1\x0bauth-google\x03com\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x06google\x03com\x00\x00\x05\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x06google\x03com\x00\x00\x05\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x06google\x03com\x00\x00\x05\x00\x01\xc0\x0c\x00\x05\x00\x01\x00\x00\x0e\x10\x00\x10\x03www\x06google\x03com\x00'
No error found in Query


Resolver sent data back to client
```

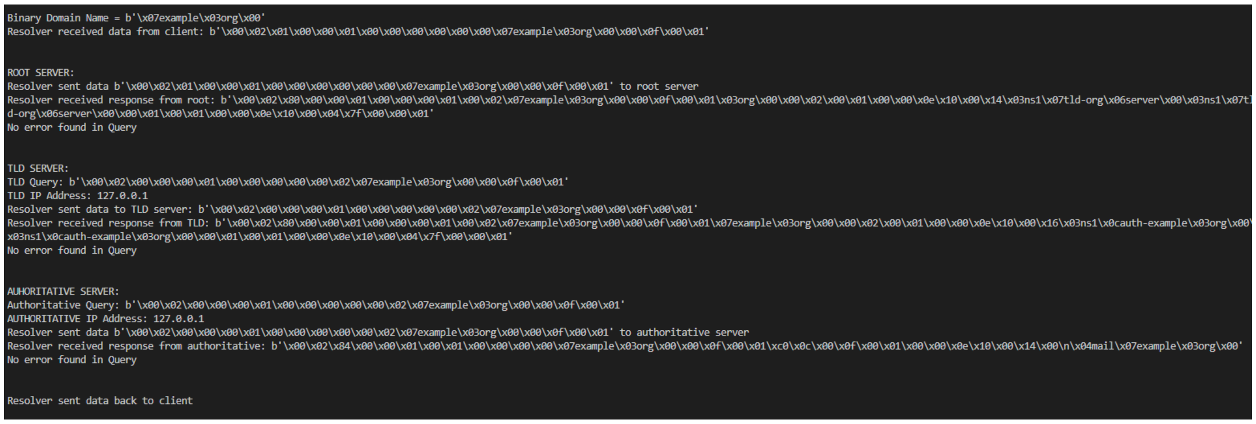
## Test Case 6

**Description:** Valid Query Type supported by DNS system (MX)
**Input:** nslookup -type=MX example.org 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=MX example.org 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

example.org     MX preference = 10, mail exchanger = mail.example.org
```

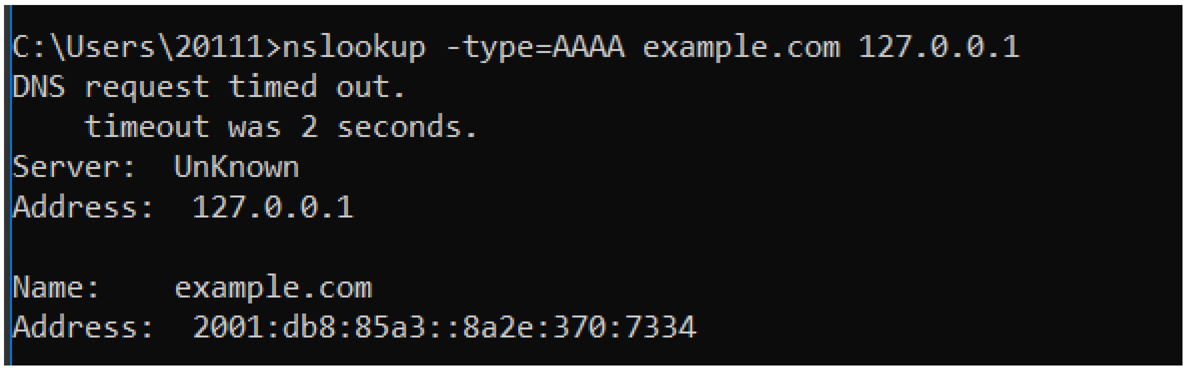
**DNS Resolver’s Logging:**



```
Binary Domain Name = b'\x07example\x03org\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x0f\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00\x0f\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-org\x06server\x00\x03ns1\x07tl
d-org\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01\x07example\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03org\x00
x03ns1\x0cauth-example\x03org\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x07example\x03org\x00\x00\x0f\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x07example\x03org\x00\x00\x0f\x00\x01\xc0\x0c\x00\x0f\x00\x01\x00\x00\x0e\x10\x00\x14\x00\n\x04mail\x07example\x03org\x00'
No error found in Query


Resolver sent data back to client
```

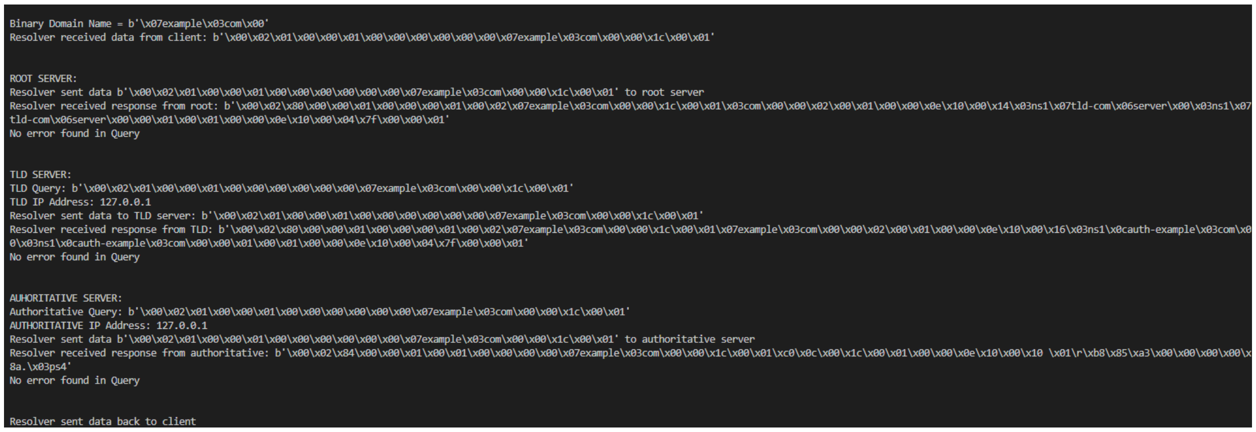
## Test Case 7

**Description:** Valid Query Type supported by DNS system (AAAA)
**Input:** nslookup -type=AAAA example.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=AAAA example.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

Name:    example.com
Address:  2001:db8:85a3::8a2e:370:7334
```

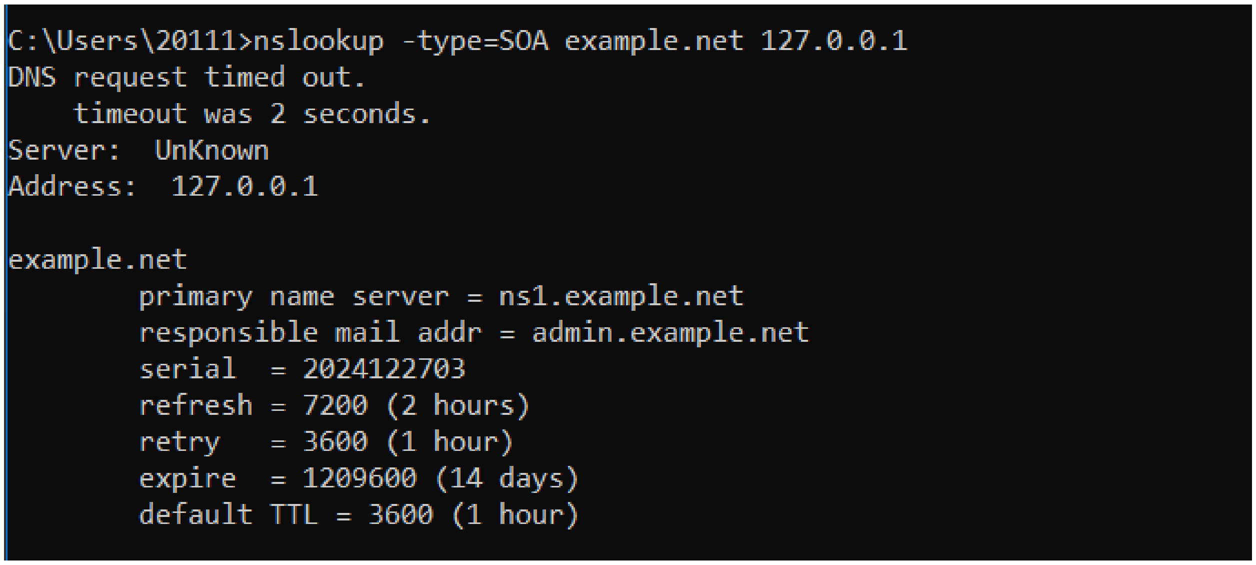
**DNS resolver’s Logging:**



```
Binary Domain Name = b'\x07example\x03com\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x1c\x00\x01\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-com\x06server\x00\x03ns1\x07
tld-com\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x1c\x00\x01\x07example\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03com\x0
0\x03ns1\x0cauth-example\x03com\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x07example\x03com\x00\x00\x1c\x00\x01\xc0\x0c\x00\x1c\x00\x01\x00\x00\x0e\x10\x00\x10 \x01\r\xb8\x85\xa3\x00\x00\x00\x00\x
8a.\x03ps4'
No error found in Query


Resolver sent data back to client
```

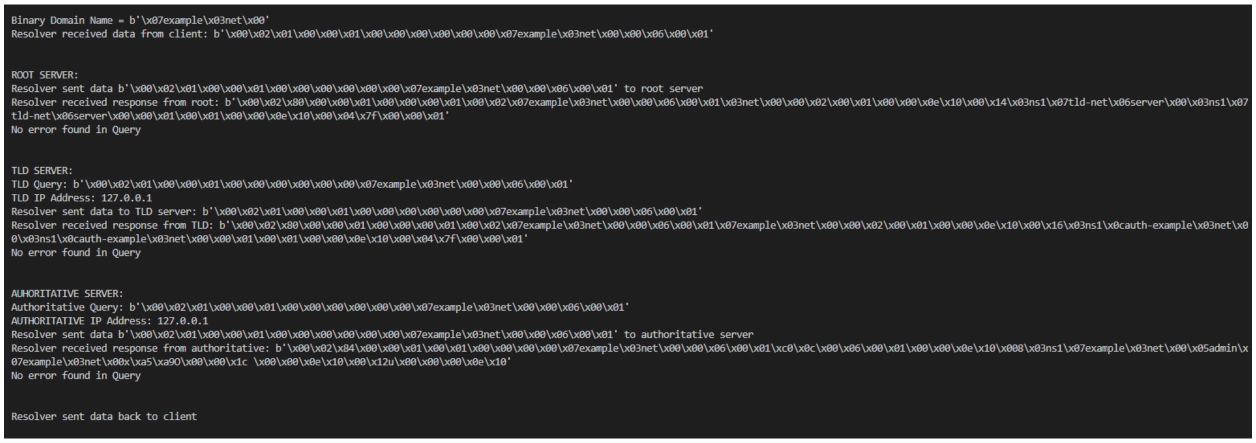
## Test Case 8

**Description:** Valid Query Type supported by DNS system (SOA)
**Input:** nslookup -type=SOA example.net 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=SOA example.net 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

example.net
        primary name server = ns1.example.net
        responsible mail addr = admin.example.net
        serial  = 2024122703
        refresh = 7200 (2 hours)
        retry   = 3600 (1 hour)
        expire  = 1209600 (14 days)
        default TTL = 3600 (1 hour)
```

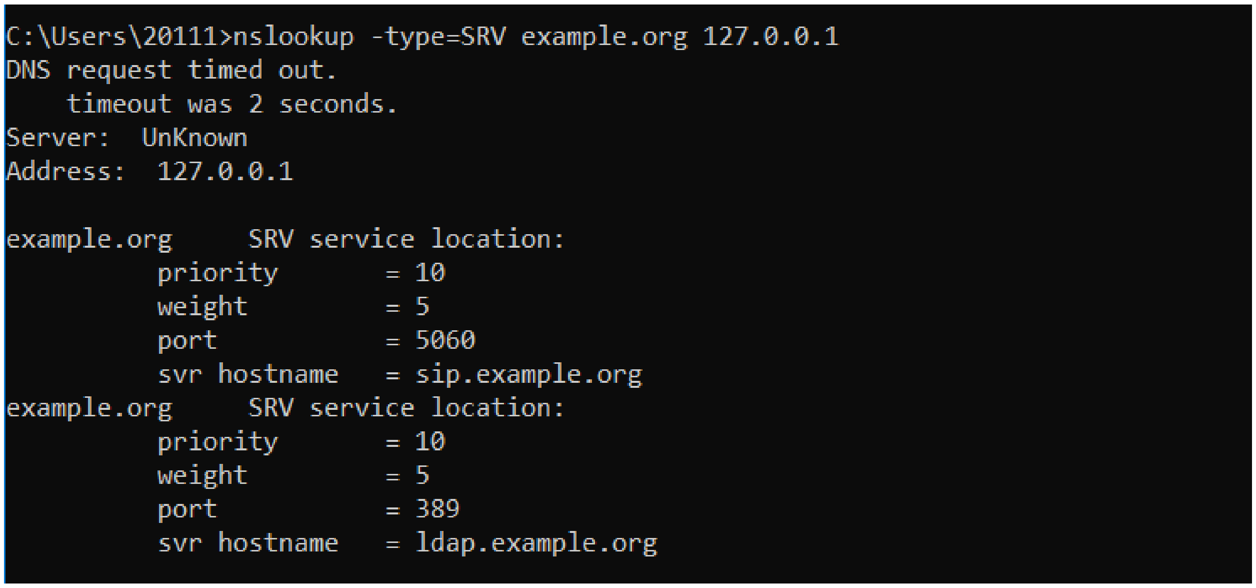
**DNS resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03net\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03net\x00\x00\x06\x00\x01\x03net\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-net\x06server\x00\x03ns1\x07
tld-net\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03net\x00\x00\x06\x00\x01\x07example\x03net\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03net\x0
0\x03ns1\x0cauth-example\x03net\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x01\x00\x00\x00\x00\x07example\x03net\x00\x00\x06\x00\x01\xc0\x0c\x00\x06\x00\x01\x00\x00\x0e\x10\x008\x03ns1\x07example\x03net\x00\x05admin\x
07example\x03net\x00x\xa5\xa9O\x00\x00\x1c \x00\x00\x0e\x10\x00\x12u\x00\x00\x00\x0e\x10'
No error found in Query


Resolver sent data back to client
```

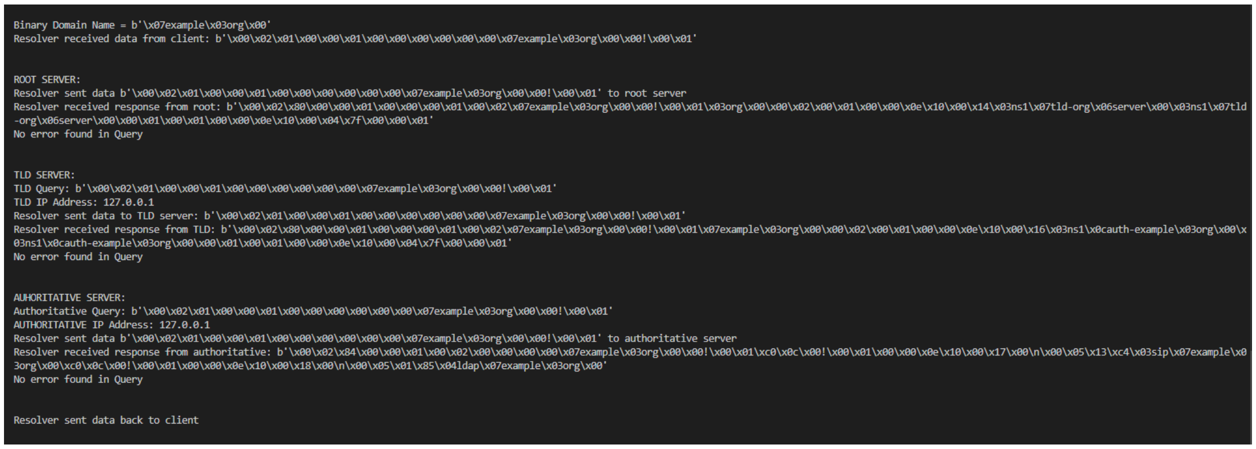
## Test Case 9

**Description:** Valid Query Type supported by DNS system (SRV)
**Input:** nslookup -type=SRV example.org 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=SRV example.org 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

example.org     SRV service location:
          priority       = 10
          weight         = 5
          port           = 5060
          svr hostname   = sip.example.org
example.org     SRV service location:
          priority       = 10
          weight         = 5
          port           = 389
          svr hostname   = ldap.example.org
```

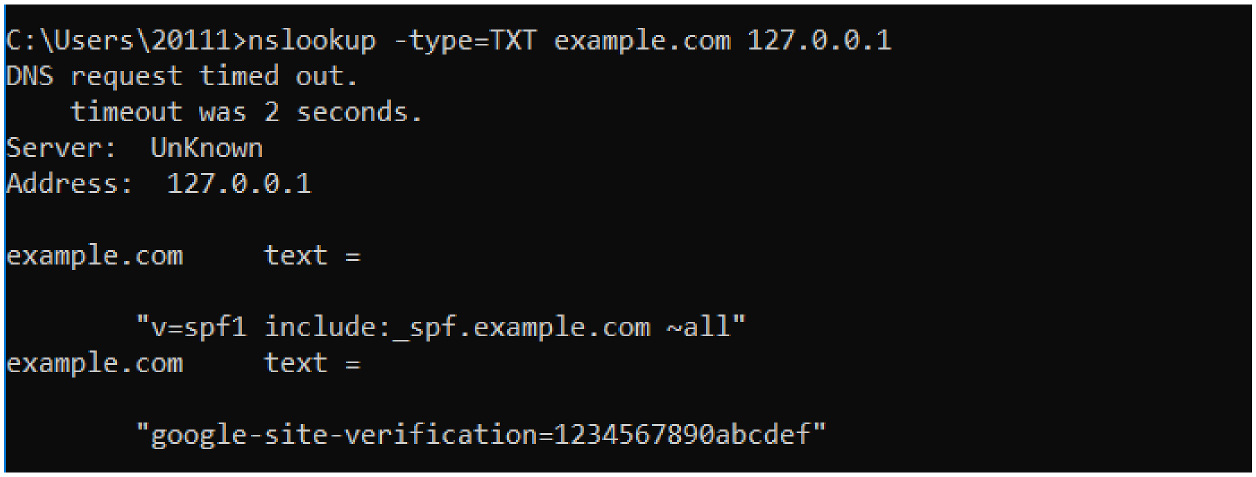
**DNS resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03org\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00!\x00\x01\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-org\x06server\x00\x03ns1\x07tld
-org\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03org\x00\x00!\x00\x01\x07example\x03org\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03org\x00\x
03ns1\x0cauth-example\x03org\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x02\x00\x00\x00\x00\x07example\x03org\x00\x00!\x00\x01\xc0\x0c\x00!\x00\x01\x00\x00\x0e\x10\x00\x17\x00\n\x00\x05\x13\xc4\x03sip\x07example\x0
3org\x00\xc0\x0c\x00!\x00\x01\x00\x00\x0e\x10\x00\x18\x00\n\x00\x05\x01\x85\x04ldap\x07example\x03org\x00'
No error found in Query


Resolver sent data back to client
```

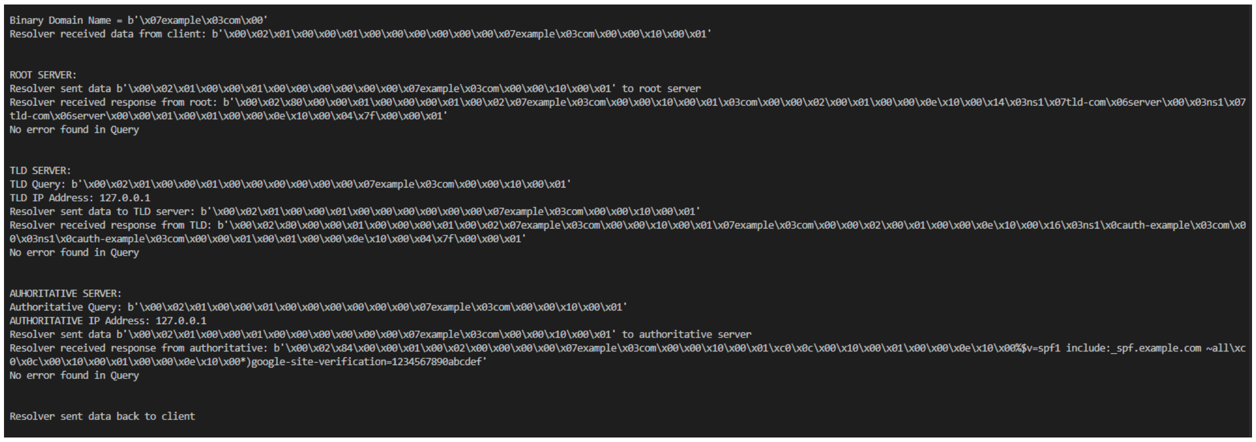
## Test Case 10

**Description:** Valid Query Type supported by DNS system (TXT)
**Input:** nslookup -type=TXT example.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=TXT example.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

example.com     text =

        "v=spf1 include:_spf.example.com ~all"
example.com     text =

        "google-site-verification=1234567890abcdef"
```

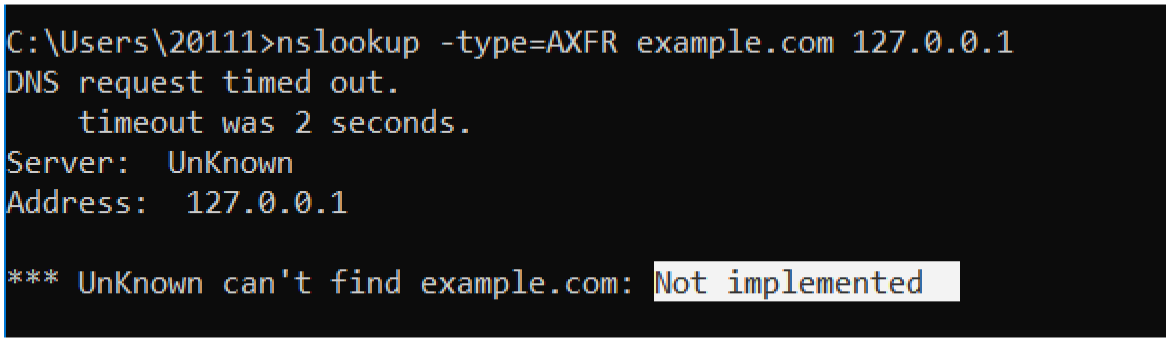
**DNS resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03com\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x10\x00\x01\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x14\x03ns1\x07tld-com\x06server\x00\x03ns1\x07
tld-com\x06server\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


TLD SERVER:
TLD Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01'
TLD IP Address: 127.0.0.1
Resolver sent data to TLD server: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01'
Resolver received response from TLD: b'\x00\x02\x80\x00\x00\x01\x00\x00\x00\x01\x00\x02\x07example\x03com\x00\x00\x10\x00\x01\x07example\x03com\x00\x00\x02\x00\x01\x00\x00\x0e\x10\x00\x16\x03ns1\x0cauth-example\x03com\x0
0\x03ns1\x0cauth-example\x03com\x00\x00\x01\x00\x01\x00\x00\x0e\x10\x00\x04\x7f\x00\x00\x01'
No error found in Query


AUHORITATIVE SERVER:
Authoritative Query: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01'
AUTHORITATIVE IP Address: 127.0.0.1
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01' to authoritative server
Resolver received response from authoritative: b'\x00\x02\x84\x00\x00\x01\x00\x02\x00\x00\x00\x00\x07example\x03com\x00\x00\x10\x00\x01\xc0\x0c\x00\x10\x00\x01\x00\x00\x0e\x10\x00%$v=spf1 include:_spf.example.com ~all\xc
0\x0c\x00\x10\x00\x01\x00\x00\x0e\x10\x00*)google-site-verification=1234567890abcdef'
No error found in Query


Resolver sent data back to client
```

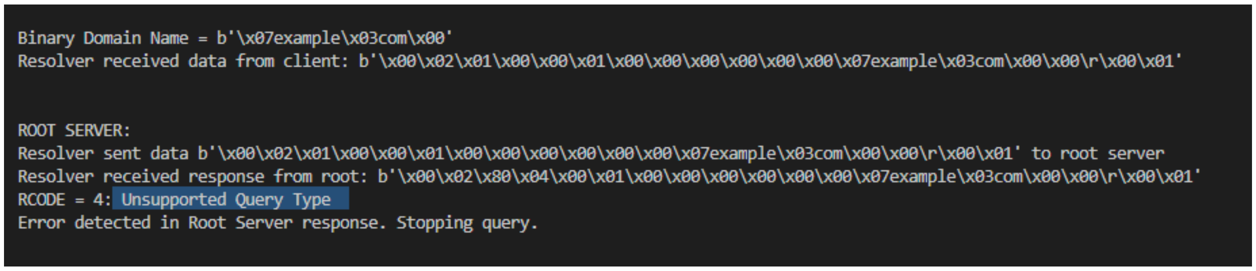
## Test Case 11

**Description:** Unsupported Query type by DNS system (AXFR)
**Input:** nslookup -type=AXFR example.com 127.0.0.1
**Output:**



```
C:\Users\20111>nslookup -type=AXFR example.com 127.0.0.1
DNS request timed out.
    timeout was 2 seconds.
Server:  UnKnown
Address:  127.0.0.1

*** UnKnown can't find example.com: Not implemented
```

**DNS resolver's Logging:**



```
Binary Domain Name = b'\x07example\x03com\x00'
Resolver received data from client: b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\r\x00\x01'


ROOT SERVER:
Resolver sent data b'\x00\x02\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\r\x00\x01' to root server
Resolver received response from root: b'\x00\x02\x80\x04\x00\x01\x00\x00\x00\x00\x00\x00\x07example\x03com\x00\x00\r\x00\x01'
RCODE = 4: Unsupported Query Type
Error detected in Root Server response. Stopping query.
```

# Installation, Configuration & Usage

## Installation

**Download the Code:**
Clone or download the repository containing the four server Python files.

**Set Up the Environment:**

Ensure that Python is installed on your system.

Install all required dependencies:

- Socket module
- Logging module
- Threading module
- Ipaddress module

## Configuration

**Server IP Address Configuration:**

By default, the servers are configured to bind to 127.0.0.1 (localhost). This allows the servers to communicate on the same device.

If you wish to allow communication across devices on the same Wi-Fi network, update the server binding IP addresses in the code to either:

- The local IP address of the device running the servers (e.g., 192.168.x.x).
- 0.0.0.0, which binds the servers to all available network interfaces.

**Database Updates:**

Each server maintains its own database. Ensure that these databases are correctly populated with the necessary data.

Add or modify records in the databases according to your use case. The format and location of these databases will depend on the implementation details specified in the project documentation.

### Usage

**Starting the Servers:**

Run all four server files simultaneously. Each server must be active and properly configured before handling any requests.

**Sending Requests:**

Use tools like nslookup or dig to send DNS queries to your server setup.

- nslookup <domain> <server-ip>
- dig <domain> @<server-ip>

Requests can be sent:

- From the same computer where the servers are running.
- From another device connected to the same Wi-Fi network. Ensure that the device running the servers is accessible by the client device.

# Conclusion

The Domain Name System (DNS) is a cornerstone of the internet, seamlessly translating human-readable domain names into machine-readable IP addresses. This critical service underpins modern internet communication by enabling users to access websites, send emails, and use various online services without needing to remember complex numerical IP addresses. To maintain consistency and reliability across the global DNS infrastructure, standards like RFCs 1034, 1035, and 2181 provide comprehensive guidelines for its design, implementation, and operation.

RFC 1034 and RFC 1035 establish the fundamental framework for DNS, detailing its hierarchical structure, query/response mechanisms, and essential record types like A, MX, and CNAME. Meanwhile, RFC 2181 complements these by addressing operational clarifications, particularly in areas such as data integrity, caching behavior, and resource record verification. Adhering to these standards ensures that DNS systems remain interoperable, robust, and scalable across diverse networks and implementations.

This project demonstrates the practical implementation of a DNS system, developed using Python and leveraging UDP as the transport protocol, consistent with the specifications of RFCs 1034 and 1035. The system comprises four independent servers designed to communicate over a shared network. These servers process and respond to DNS queries, adhering to the hierarchical delegation model defined in the RFCs. The project includes support for multiple record types, such as A, MX, and CNAME, and ensures accurate responses, including authoritative answers and proper error handling with RCODEs for invalid or unsupported requests.

Additionally, the implementation incorporates advanced features like multi-threading for handling concurrent queries efficiently and logging mechanisms to aid in debugging, monitoring, and auditing. Logging provides essential insights during development and real-time operation, offering detailed information about query handling, errors, and system status. This emphasis on maintainability and extensibility ensures the system's readiness for future enhancements, such as supporting additional record types or domains.

In conclusion, this project not only fulfills the functional requirements of a DNS system but also underscores the importance of adhering to established standards. By implementing DNS features according to RFCs 1034, 1035, and 2181, it ensures interoperability and reliability, providing a robust foundation for further exploration and refinement in the ever-evolving field of network systems. This practical exercise highlights the enduring significance of these RFCs in guiding DNS implementation and fostering a cohesive internet infrastructure

## References

- IETF Data tracker
  https://datatracker.ietf.org/

- Open AI ChatGPT
  https://chatgpt.com

- GeeksforGeeks
  https://www.geeksforgeeks.org

- Fortinet
  https://www.fortinet.com