A blue and white logo

Description automatically generated

CSE 351 Computer Networks  
Fall 2024/2025

Submitted to:

Dr. Ayman Bahaa

Eng. Noha Wahdan

Submitted by:

Mohamed Mostafa Mamdouh 21P0244

Gaser Zaghloul 21P0052

Ali Tarek 21P0123

Abdelrahman Sherif Hassan 21P0098

Table of Contents

[Introduction 4](#_Toc183544006)

[Required RFCs 6](#_Toc183544007)

[RFC 1034 6](#_Toc183544008)

[RFC 1035 6](#_Toc183544009)

[RFC 2181 6](#_Toc183544010)

[Project Planning and Design 7](#_Toc183544011)

[System Architecture 8](#_Toc183544012)

[Communication Protocols 13](#_Toc183544013)

[Steps 15](#_Toc183544014)

# 

# Introduction

What is DNS?

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.

Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses in IPv4 and IPv6.

Iterated query: The resolver starts by asking a root DNS server, which then refers it to a TLD (Top Level Domain) server. The TLD server then refers the resolver to the authoritative DNS server for the specific domain. The resolver continues this process until it gets the IP address.

Recursive query: the DNS resolver asks a single DNS server to handle the entire resolution process. The resolver sends a query to the DNS server, which then takes on the responsibility of querying other DNS servers if needed until it finds the final answer.

**DNS Hierarchy**

The DNS hierarchy is organized as an inverted tree structure with the following key levels:

**1.1 Root Level**

* **Root Domain**: Represented by a single dot (.) at the top of the hierarchy.
* The root domain does not have a name itself but serves as the starting point for DNS resolution.
* The **root servers** handle queries for top-level domains (TLDs) like .com, .org, or .edu.
* **Example**: A query for www.example.com starts at the root, which directs the query to the .com TLD.

**1.2 Top-Level Domain (TLD)**

* TLDs are the second level in the hierarchy and categorize domains.
* Examples include:
  + **Generic TLDs (gTLDs)**: .com, .org, .net
  + **Country Code TLDs (ccTLDs)**: .uk, .fr, .jp
  + **Sponsored TLDs**: .gov, .edu
* Managed by organizations like **ICANN** and delegated to **registry operators**.

**1.3 Second-Level Domain (SLD)**

* This is the domain name registered under a TLD, often representing an organization, company, or individual.
* **Example**: example in example.com.

**1.4 Subdomain**

* A subdomain is part of a larger domain and can be used for organizing resources.
* **Example**: www in www.example.com or blog in blog.example.com.

**1.5 Hostname**

* The most specific part of a domain name, pointing to a specific machine or service.
* **Example**: mail.example.com could point to a mail server.

Authoritative servers: These servers hold the actual DNS records for a domain. They are the definitive source of information for that domain. When a query is made, an authoritative server provides a response based on its own records. It can be maintained by organization or service provider.

Non-authoritative servers: These servers do not hold the DNS records themselves. Instead, they cache the information received from authoritative servers. When a query is made, a non-authoritative server provides a response based on its cached data. If the data is not in the cache, it queries an authoritative server to get the information.

A close-up of a server

Description automatically generated

DNS types:  
**A Record (Address Record):**

* **Purpose**: Maps a domain name to an IPv4 address.
* **Example**: example.com -> 93.184.216.34

**AAAA Record (IPv6 Address Record):**

* **Purpose**: Maps a domain name to an IPv6 address.
* **Example**: example.com -> 2606:2800:220:1:248:1893:25c8:1946

**CNAME Record (Canonical Name Record):**

* **Purpose**: Alias one domain name to another.
* **Example**: www.example.com -> example.com

**HINFO Record (Host Information Record):**

* **Purpose**: Provides details about a host's CPU and operating system.
* **Example**: host.example.com -> "Intel" "Linux"

**NS Record (Name Server Record):**

* **Purpose**: Specifies the authoritative DNS servers for a domain.
* **Example**: example.com -> ns1.example.com

**MX Record (Mail Exchange Record):**

* **Purpose**: Directs email to a mail server.
* **Example**: example.com -> mail.example.com

**PTR Record (Pointer Record):**

* **Purpose**: Maps an IP address to a domain name (reverse DNS lookup).
* **Example**: 192.0.2.1 -> server.example.com

**SOA Record (Start of Authority Record):**

* **Purpose**: Contains administrative information about a DNS zone.

Required RFCs

RFC 1034 Domain Names – Concepts and Facilities. This foundational RFC defines the concept of the Domain Name System (DNS), which is responsible for translating human-readable domain names into IP addresses. It outlines the basic structure and operation of DNS.

* Link: [RFC 1034 - Domain names - concepts and facilities](https://datatracker.ietf.org/doc/rfc1034/)

RFC 1035 Domain Names – Implementation and Specification. This RFC provides detailed specifications for the DNS protocol, including message formats, query types, and response structures. It focuses on how the DNS protocol operates in practice.

* Link: [RFC 1035 - Domain names - implementation and specification](https://datatracker.ietf.org/doc/rfc1035/)

RFC 2181 Clarifications to the DNS Specification. This RFC clarifies ambiguities and updates certain aspects of the original DNS specification (RFC 1034 and RFC 1035) to address issues encountered during deployment.

* Link: [RFC 2181 - Clarifications to the DNS Specification](https://datatracker.ietf.org/doc/rfc2181/)

# Project Planning and Design

The goal of this project is to analyze and implement a server agent for DNS protocol that is compliant with the RFC. The project should work seamlessly with common off-the shelf clients such as a typical browser or Windows DNS client agents. The project will be divided into four phases, each building on the previous one, to demonstrate compliance details with the RFC. The project should follow behaviors, scenarios, messages format and sequence, and produce RFC-compliant outcomes or error codes.

The aim of this project is to develop an RFC-compliant DNS server agent that operates seamlessly with standard DNS clients. The server should handle queries, provide authoritative responses, and cache results as needed.

Its goal involves ensuring compliance with DNS protocol RFCs 1034, 1035 and 2181, and be able to handle valid queries and errors effectively.

The DNS server is the core component of the architecture that processes DNS queries and generates responses. It includes modules for parsing queries, accessing stored data, handling errors and constructing RFC-compliant responses.

This server will perform the following key functionalities:

* Resolve domain names to IP addresses (forward lookup).
* Translate IP addresses back to domain names (reverse lookup).
* Provide error codes and responses as per the DNS specifications.
* Support typical DNS query types (A, AAAA, CNAME, MX, etc.).
* Handle query and response messages in a robust, RFC-compliant manner.

**2. Project Scope**

The DNS server agent will support the following features:

**Key Functionalities**

* **Query Processing**:
  + Resolve queries for records such as A (IPv4), AAAA (IPv6), CNAME, NS, MX, and PTR.
  + Handle recursive and iterative queries.
  + Generate appropriate responses for queries with correct format and TTL values.
* **Error Handling**:
  + Return RFC-compliant error codes for scenarios like NXDOMAIN (non-existent domain), SERVFAIL, or format errors.
* **Caching**:
  + Implement a caching mechanism to reduce response latency and improve efficiency.
  + Ensure consistency by respecting TTL values as specified in RFC 2181.
* **Security**:
  + Integrate DNSSEC for validation of data integrity.
  + Prevent unauthorized access or spoofing through signed records.

## System Architecture

DNS server agent will consist of the following components:

**1. Resolver Module**

The **resolver** is responsible for handling incoming queries from clients (e.g., browsers, operating systems) and determining the best way to process them.

**Key Functions:**

1. **Query Parsing**:
   * Accepts DNS queries in a standardized format as described in RFC 1035.
   * Extracts key fields such as the domain name, query type (A, AAAA, etc.), and query class.
2. **Decision Making**:
   * If the requested domain data is available in the cache (and TTL is valid), server will respond immediately.
   * If not in the cache:
     + **Recursive Resolution**: The resolver queries other DNS servers to fully resolve the request on behalf of the client.
     + **Iterative Resolution**: The resolver provides referral information to the client to query the next server.
3. **Query Validation**:
   * Ensures that incoming queries are well-formed and RFC-compliant.
   * Returns error codes such as FORMERR (format error) for malformed queries or SERVFAIL for server-side issues.

**Communication Flow:**

* **Input**: Queries from clients.
* **Output**: Responses to clients, which could be answers, referrals, or errors.

**Interactions:**

* Uses the **cache module** to check for existing answers.
* Communicates with external authoritative servers during recursive lookups.

**2. Authoritative Server Module**

The **authoritative server** stores and serves data for specific zones, such as resource records (RRs) for example.com.

**Key Functions:**

1. **Zone Management**:
   * Stores resource records (e.g., A, AAAA, NS, MX) for zones it is authoritative for.
   * Updates zone information from zone files or external sources as needed.
2. **Authoritative Responses**:
   * Answers queries for domains under its management with authoritative data.
   * Ensures consistency by checking record validity before responding.
3. **Zone Transfers**:
   * Allows secondary servers to perform zone transfers (AXFR) to replicate data.
   * Implements access control for zone transfers to ensure only trusted servers can request data.
4. **SOA (Start of Authority) Management**:
   * Provides SOA records to indicate the authoritative server for a zone and define parameters like refresh intervals and TTLs.

**Communication Flow:**

* **Input**: Queries for domains it manages.
* **Output**: Authoritative answers to resolvers or clients.

**Interactions:**

* Interacts with the **communication layer** to encode/decipher DNS messages.
* Updates zone data from **master files** or other trusted sources.

**3. Cache Module**

The **cache** temporarily stores results of resolved queries to improve response time and reduce the need for repetitive queries.

**Key Functions:**

1. **Result Storage**:
   * Saves answers from external queries along with their TTL values.
   * Maintains a mapping of domain names to resource records.
2. **TTL Management**:
   * Automatically invalidates entries when their TTL expires, as specified in RFC 2181.
   * Refreshes records by querying the authoritative server if needed.
3. **Error Handling**:
   * Avoids caching of responses marked with NOERROR but containing no answer records, to prevent stale or misleading data.
4. **Performance Optimization**:
   * Reduces latency for repeated queries by serving from the cache.
   * Minimizes external network traffic by reusing cached data.

**Communication Flow:**

* **Input**: Resolved records from authoritative servers or upstream resolvers.
* **Output**: Answers to queries if they match valid cache entries.

**Interactions:**

* Works closely with the **resolver module** to check if a query can be satisfied locally.
* Communicates with external servers to refresh cached data when necessary.

**4. Communication Protocol Layer**

This module manages the encoding, decoding, and transmission of DNS messages as per RFC 1035.

**Key Functions:**

1. **Message Parsing and Construction**:
   * Parses incoming messages into components: header, question, answer, authority, and additional sections.
   * Constructs outgoing messages using the same structure.
2. **Protocol Handling**:
   * **UDP**:
     + Default transport for most DNS queries.
     + Limits messages to 512 bytes unless EDNS is supported.
   * **TCP**:
     + Used for responses larger than 512 bytes or for zone transfers.
   * Implements retry mechanisms for reliability.
3. **Error Messaging**:
   * Responds with appropriate error codes:
     + NXDOMAIN for non-existent domains.
     + SERVFAIL for server-side issues.
     + REFUSED for unauthorized requests.
4. **Compression**:
   * Implements message compression to reduce the size of DNS responses.
5. **Security Features**:
   * Uses DNSSEC to sign outgoing responses and validate incoming queries.
   * Ensures that responses are sent only to the origin of the query (to prevent spoofing).

**Communication Flow:**

* **Input**: Raw DNS packets from clients or upstream servers.
* **Output**: Encoded DNS packets containing responses or referrals.

**Interactions:**

* Facilitates communication between all modules and external systems.
* Interfaces with the **resolver module** for query handling and the **authoritative server module** for data retrieval.

| **Error Code** | **Meaning** | **Cause** |
| --- | --- | --- |
| NOERROR | Query successful | No issues with processing. |
| FORMERR | Format Error | Malformed query message. |
| SERVFAIL | Server Failure | Internal server error or data unavailability. |
| NXDOMAIN | Non-existent Domain | Queried domain does not exist. |
| NOTIMP | Not Implemented | Unsupported query type or operation. |
| REFUSED | Query Refused | Policy or authorization restriction. |
| YXDOMAIN | Name Exists | Dynamic update error. |
| YXRRSET | RR Set Exists | Dynamic update error. |
| NXRRSET | RR Set Does Not Exist | Dynamic update error. |
| NOTAUTH | Not Authorized | Server not authoritative for the zone. |
| NOTZONE | Not in Zone | Query outside the managed zone. |

**Security Considerations**

Each module incorporates security measures:

1. **Resolver Module**:
   * Verifies the legitimacy of incoming queries.
   * Protects against amplification attacks by limiting response size and rate.
2. **Authoritative Server Module**:
   * Implements DNSSEC for signing responses.
   * Restricts zone transfers to trusted servers.
3. **Cache Module**:
   * Ensures data consistency by honoring TTL values.
   * Prevents cache poisoning by validating data integrity.
4. **Communication Protocol Layer**:
   * Uses secure protocols and validates source addresses to avoid spoofed queries.

## Communication Protocols

**4. Communication Protocols**

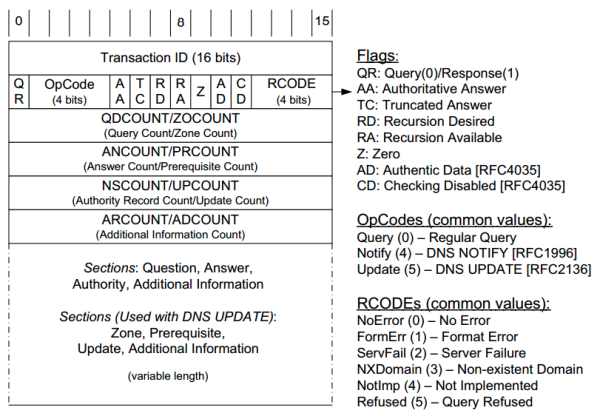
The DNS server agent will follow the communication protocols and message formats outlined in RFC 1035.

**4.1. DNS Message Format**

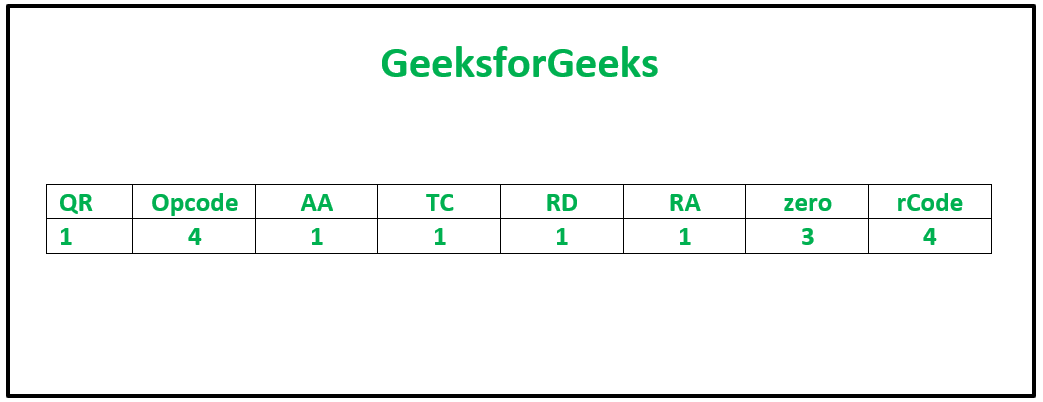
Each message will include:

* **Header**: Contains fields like query ID, flags (e.g., recursion desired, truncated message), and counters for questions, answers, authority, and additional records. A DNS header is 12 bytes (96 bits) long and consists of the following fields:

| **Field Name** | **Size** | **Description** |
| --- | --- | --- |
| **ID** | 16 bits | A unique identifier for matching queries and responses. |
| **Flags** | 16 bits | Control flags and status indicators (broken into subfields). |
| **QDCOUNT** | 16 bits | Number of questions in the Question section. |
| **ANCOUNT** | 16 bits | Number of resource records in the Answer section. |
| **NSCOUNT** | 16 bits | Number of resource records in the Authority section. |
| **ARCOUNT** | 16 bits | Number of resource records in the Additional section. |



Flags:



| **Field** | **Size** | **Meaning** |
| --- | --- | --- |
| **QR** | 1 bit | Indicates whether the message is a query (0) or a response (1). |
| **Opcode** | 4 bits | Defines the query type (e.g., standard, inverse, update). |
| **AA** | 1 bit | Indicates if the server is authoritative for the domain. |
| **TC** | 1 bit | Indicates if the message was truncated (size limit exceeded), indicating the client may need to re-query using TCP. |
| **RD** | 1 bit | Indicates if recursion is desired by the client (1) or not (0). |
| **RA** | 1 bit | Indicates if recursion is available from the server. (1) or not (0). |
| **Zero** | 3 bits | Reserved for future use by potential extensions, always set to 0. |
| **RCODE** | 4 bits | Indicates the status or error code for the response (e.g., NOERROR, NXDOMAIN). |

**Opcode Possible Values**:

* 0000 (Standard Query): The most common query type, asking for a specific resource record.
* 0001 (Inverse Query): Used for reverse lookups (obsolete, replaced by PTR records).
* 0010 (Status Request): Requests status information about the DNS server (also deprecated).
* 0111 (Update): Used for DNS updates, introduced with dynamic DNS (RFC 2136).

**Errors Defined in the DNS Protocol (RFC 1035)**

**Response Codes (RCODEs)**

These are 4-bit fields in the DNS header used to indicate the outcome of a query:

1. **NOERROR (0)**:
   * **Meaning**: The query was successful.
   * **Action**: Normal response; no error.
2. **FORMERR (1)**:
   * **Meaning**: Format Error.
   * **Cause**: The query was not properly formed (e.g., invalid DNS message format).
   * **Action**: Return this error code to the client.
3. **SERVFAIL (2)**:
   * **Meaning**: Server Failure.
   * **Cause**: The server is unable to process the query due to internal issues (e.g., misconfiguration or unavailability of required data).
   * **Action**: Report this to the client; may include retry instructions.
4. **NXDOMAIN (3)**:
   * **Meaning**: Non-Existent Domain.
   * **Cause**: The domain name queried does not exist in the DNS hierarchy.
   * **Action**: The response should include the authoritative status and this code.
5. **NOTIMP (4)**:
   * **Meaning**: Not Implemented.
   * **Cause**: The server does not support the requested query type or operation.
   * **Action**: Indicate that the requested feature is unsupported.
6. **REFUSED (5)**:
   * **Meaning**: Query Refused.
   * **Cause**: The server refuses to process the query, often due to policy (e.g., restricted zones or unauthorized clients).
   * **Action**: Inform the client and include a reason if appropriate.
7. **YXDOMAIN (6)**:
   * **Meaning**: Name Exists when it should not.
   * **Cause**: Used in dynamic updates when the update attempts to add a record to an existing domain name that should not exist.
   * **Action**: Return this error for incorrect updates.
8. **YXRRSET (7)**:
   * **Meaning**: RR Set Exists when it should not.
   * **Cause**: Used in dynamic updates when the update tries to add resource records to an RR set that already exists.
   * **Action**: Return this error for incorrect updates.
9. **NXRRSET (8)**:
   * **Meaning**: RR Set does not exist.
   * **Cause**: Used in dynamic updates when the update attempts to delete a non-existent RR set.
   * **Action**: Inform the client of the error.
10. **NOTAUTH (9)**:
    * **Meaning**: Not Authorized.
    * **Cause**: The server is not authoritative for the requested zone.
    * **Action**: Refuse the operation and report the error.
11. **NOTZONE (10)**:
    * **Meaning**: Not in Zone.
    * **Cause**: A name used in the operation is outside the zone being managed.
    * **Action**: Indicate that the name does not belong to the specified zone.

| **Error Code** | **Meaning** | **Cause** |
| --- | --- | --- |
| NOERROR | Query successful | No issues with processing. |
| FORMERR | Format Error | Malformed query message. |
| SERVFAIL | Server Failure | Internal server error or data unavailability. |
| NXDOMAIN | Non-existent Domain | Queried domain does not exist. |
| NOTIMP | Not Implemented | Unsupported query type or operation. |
| REFUSED | Query Refused | Policy or authorization restriction. |
| YXDOMAIN | Name Exists | Dynamic update error. |
| YXRRSET | RR Set Exists | Dynamic update error. |
| NXRRSET | RR Set Does Not Exist | Dynamic update error. |
| NOTAUTH | Not Authorized | Server not authoritative for the zone. |
| NOTZONE | Not in Zone | Query outside the managed zone. |

**3. Errors Related to Resource Record Handling (RFC 1034, RFC 2181)**

1. **TTL Consistency (RFC 2181)**:
   * **Cause**: RR sets with inconsistent TTL values.
   * **Action**: Treat this as an error and either discard the RR set or use the lowest TTL in the set.
2. **Duplicate Records**:
   * **Cause**: Identical records (same name, class, type, and data) appear in a zone file.
   * **Action**: Suppress duplicates.
3. **Zone Integrity**:
   * **Cause**: Misconfigured zones, such as missing NS or SOA records.
   * **Action**: Report an internal error and potentially refuse queries for the affected zone.

**5. Errors Related to DNSSEC (RFC 2181)**

1. **Signature Validation Failure**:
   * **Cause**: The DNSSEC signature for a response does not validate.
   * **Action**: Set the AD flag to 0 and return SERVFAIL.
2. **Missing DNSSEC Records**:
   * **Cause**: DNSSEC is enabled, but required records (e.g., RRSIG, DS) are missing.
   * **Action**: Return SERVFAIL.
3. **Validation Disabled by Client**:
   * **Cause**: The client sets the CD (Checking Disabled) flag.
   * **Action**: Skip DNSSEC validation and return the data as is.

* **Question Section**: The query sent by the client, specifying the domain name, type of record (e.g., A, CNAME), and class (usually IN for internet).
* **Answer Section**: Includes resource records (RRs) that answer the query.
* **Authority Section**: Lists authoritative name servers for the queried domain.
* **Additional Section**: Provides extra information, such as IP addresses for authoritative servers.

Note that the content, but not the format, of these sections varies with header opcode.

In DNS queries, the **Question Section** specifies what information the client is requesting. This section contains three key fields:

**1. QNAME (Queried Name)**

* **Purpose**: Specifies the domain name being queried.
* **Format**:
  + A fully qualified domain name (FQDN) is represented as a series of labels (e.g., www.example.com).
  + Each label is preceded by its length and ends with a null byte (length of 0).

**2. QTYPE (Query Type)**

* **Purpose**: Indicates the type of record the client is requesting for the domain name.
* **Format**: A 16-bit field representing the requested record type.
* **Common QTYPE Values**:

| **QTYPE** | **Value** | **Meaning** |
| --- | --- | --- |
| A | 1 | IPv4 address record |
| AAAA | 28 | IPv6 address record |
| CNAME | 5 | Canonical name (alias) |
| MX | 15 | Mail exchange record |
| NS | 2 | Name server record |
| PTR | 12 | Reverse DNS record |
| SOA | 6 | Start of Authority record |
| TXT | 16 | Text record |
| ANY | 255 | All available records |

**3. QCLASS (Query Class)**

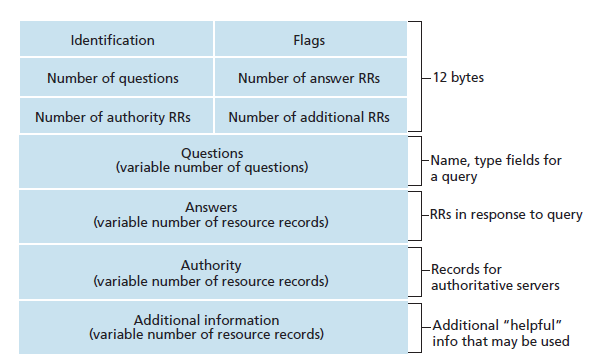
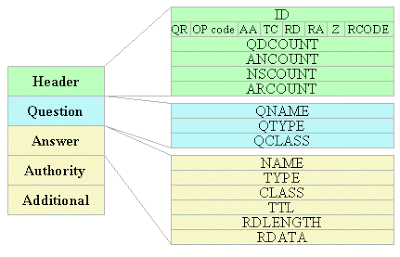
* **Purpose**: Specifies the protocol family or class of the query.
* **Format**: A 16-bit field representing the class.
* **Common QCLASS Values**:

| **QCLASS** | **Value** | **Meaning** |
| --- | --- | --- |
| IN | 1 | Internet (most common class) |
| CH | 3 | Chaosnet (debugging, legacy) |
| HS | 4 | Hesiod (used in Project Athena) |
| ANY | 255 | All available classes |

**Example DNS Query Breakdown**

A query for the A record of www.example.com in the Internet class would look like this:

| **Field** | **Value** | **Description** |
| --- | --- | --- |
| **QNAME** | www.example.com | The domain name being queried. |
| **QTYPE** | 1 | Requesting an IPv4 address (A record). |
| **QCLASS** | 1 | Querying the Internet class (IN). |



**4.2. Supported Query Types**

The server will support:

* **A**: IPv4 address record.
* **AAAA**: IPv6 address record.
* **CNAME**: Canonical name for aliasing domains.
* **MX**: Mail exchange server.
* **NS**: Name server for a zone.
* **PTR**: Reverse DNS lookup.

**4.3. Transport Protocols**

* **UDP** (default): Used for most DNS queries.
* **TCP**: Used for responses exceeding 512 bytes or zone transfers.

Responses include a TTL field, which specifies how long clients can cache the data.

**UDP** will be used as the default transport protocol on port 53, which is used for most DNS queries due to its low latency.

**Workflow Example**

1. Client sends a query: "What is the IP address of example.com?"
2. Query Processor parses the query and checks the database.
3. Database returns the IP address (93.184.216.34).
4. Response Generator constructs a response and includes the Answer Section with the IP.
5. Error Handler intervenes if the query is invalid or the domain is not found.
6. The response is sent back to the client using UDP.

**Inter-module Communication**

The modules are tightly integrated to ensure smooth operation:

1. **Resolver ↔ Cache**:
   * The resolver queries the cache before forwarding the request.
   * Cache sends a valid response if the query matches a valid record.
2. **Resolver ↔ Authoritative Server**:
   * The resolver forwards queries for domains managed by the authoritative server directly.
   * Authoritative server provides definitive answers.
3. **All Modules ↔ Communication Layer**:
   * Handles the low-level transmission of messages to/from external clients or servers.

**3.2. Interaction Between Components**

1. **Client Interaction**:
   * The resolver accepts DNS queries from clients (e.g., browsers, operating systems).
   * It parses and validates the query format before deciding how to handle it.
2. **Cache Lookup**:
   * If a query matches an entry in the cache and is still valid (TTL not expired), the response is returned directly.
   * If not, the query is forwarded to an authoritative server or another resolver.
3. **Recursive vs. Iterative Resolution**:
   * Recursive queries: The server contacts other DNS servers to resolve the query fully on behalf of the client.
   * Iterative queries: The server provides referrals to other DNS servers for the client to continue the resolution process.
4. **Response Handling**:
   * The server constructs a response message based on the query and either returns the result or a relevant error code.
   * Responses are checked for compliance with DNS standards (e.g., case-insensitivity of domain names, proper error handling).

**5. User Authentication Mechanisms**

Authentication will prevent unauthorized access and ensure secure data transmission.

**Proposed Methods:**

1. **DNSSEC Integration**:
   * Use digital signatures to validate the authenticity of responses.
   * Validate using public key cryptography as specified in DNSSEC standards.
2. **Access Control Lists (ACLs)**:
   * Restrict which IP addresses or subnets can send queries to the server.
   * Limit zone transfers to specific trusted servers.
3. **Logging and Monitoring**:
   * Maintain logs of incoming queries and responses to detect and prevent abuse (e.g., DNS amplification attacks).

**6. Compliance with RFCs**

The server agent will adhere to the following key guidelines:

* **RFC 1034**:
  + Hierarchical domain name space structure.
  + Iterative and recursive resolution strategies.
* **RFC 1035**:
  + Standard DNS message formats and protocols.
  + Handling of resource records, caching, and TTL values.
* **RFC 2181**:
  + Clarifications on TTL uniformity, authoritative responses, and resource record sets (RRSets).

**7. Implementation Plan**

1. **Develop Core Components**:
   * Build resolver, authoritative server, and caching functionality.
   * Implement basic query handling for A, CNAME, and NS records.
2. **Implement RFC-Compliant Communication**:
   * Construct DNS messages as per RFC 1035 specifications.
   * Verify interoperability with common DNS clients.
3. **Add Security Features**:
   * Integrate DNSSEC for data integrity.
   * Configure ACLs for restricted access.
4. **Testing and Validation**:
   * Test against off-the-shelf DNS clients (e.g., dig, Windows DNS client).
   * Validate compliance with DNS standards using tools like dnsperf or custom test scripts.

## Steps

1. **Define User Authentication Mechanisms**

Objective here is to restrict access to sensitive operations like zone transfers or dynamic DNS updates. This can be done through the use of **ACLs** (Access Control Lists), where rules are placed on client IP address to allow or deny specific actions.   
Only IPs in a trusted subnet can perform updates.  
Public clients can query but not modify DNS records.

1. Outline the overall

**Reverse DNS Lookup** is the process of resolving an IP address back to its associated domain name. It is the opposite of the typical DNS lookup, which translates a domain name into an IP address.

**How It Works**

1. **Pointer Record (PTR)**:
   * Reverse DNS lookups rely on **PTR records** in the DNS.
   * A PTR record maps an IP address to a domain name.
2. **Reverse Mapping Zone**:
   * To support reverse lookups, IP addresses are organized into a special reverse mapping domain called **in-addr.arpa** for IPv4 and **ip6.arpa** for IPv6.
   * The IP address is reversed, and the resulting domain name is appended with .in-addr.arpa (for IPv4) or .ip6.arpa (for IPv6).
   * Example:
     + IPv4: 192.0.2.1 becomes 1.2.0.192.in-addr.arpa.
     + IPv6: 2001:db8::1 becomes 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.8.b.d.0.1.0.0.2.ip6.arpa.
3. **Query Process**:
   * The resolver queries the reverse DNS zone for the PTR record associated with the reversed IP address.
   * The authoritative server responds with the domain name.

**Example**

1. An application queries for the reverse DNS of the IP 192.0.2.1.
2. The resolver converts the IP into the reverse domain: 1.2.0.192.in-addr.arpa.
3. It queries the DNS system for the PTR record of 1.2.0.192.in-addr.arpa.
4. The DNS server responds with the domain name: server.example.com.

**Use Cases**

1. **Email Server Validation**:
   * Mail servers often perform reverse DNS lookups to verify the legitimacy of the sending server's IP address.
   * Example: An email from 192.0.2.1 should resolve back to the domain claiming to send the email.
2. **Network Diagnostics**:
   * Tools like ping or traceroute often perform reverse lookups to show domain names instead of raw IP addresses.
3. **Logging and Analytics**:
   * Reverse lookups can help translate IP addresses in logs into meaningful domain names for easier analysis.

In DNS, **Resource Record (RR) Class** defines the network or protocol family to which the resource record applies. It is a field in every DNS resource record and is used to differentiate between various types of data stored in the DNS.

**Key RR Classes**

1. **IN (Internet)**:
   * **Purpose**: The most common class, used for internet-based DNS records.
   * **Example Use**: Resolving domain names to IPv4/IPv6 addresses, mail exchanges, and other internet-specific information.
   * **Typical Records**: A, AAAA, CNAME, MX, NS, PTR, etc.
2. **CH (Chaosnet)**:
   * **Purpose**: Originally used for the Chaosnet networking protocol, primarily for debugging and system-specific data.
   * **Example Use**: Querying server-specific information like version details.
   * **Example Record**: dig @server CHAOS TXT version.bind (queries the server's version).

**Structure in an RR**

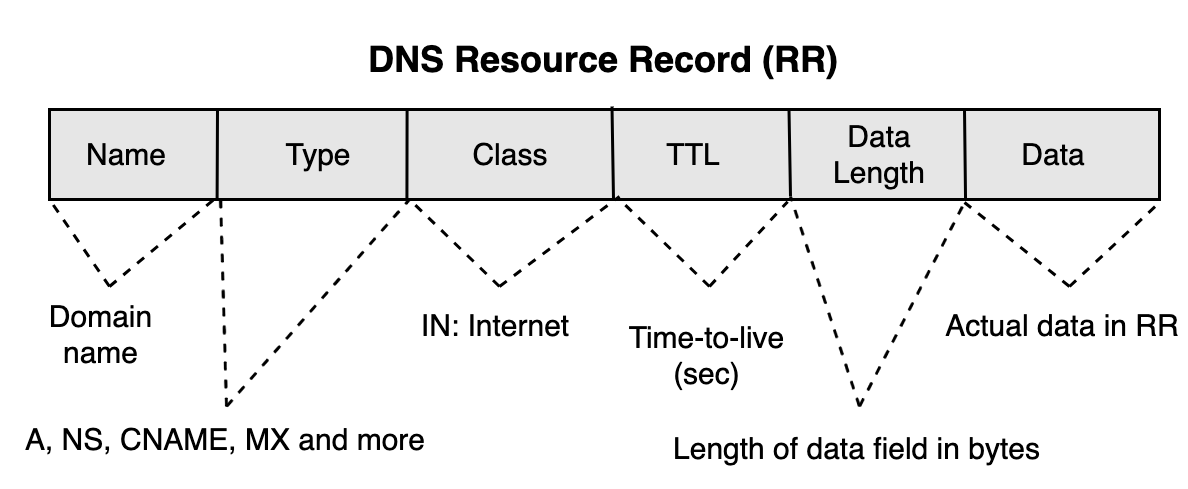
The **CLASS** field is a 16-bit value in a DNS resource record. It identifies which class the record belongs to. For most practical purposes, the value IN (Internet) is used.

Example of RR with class:  
example.com. 3600 IN A 192.0.2.1

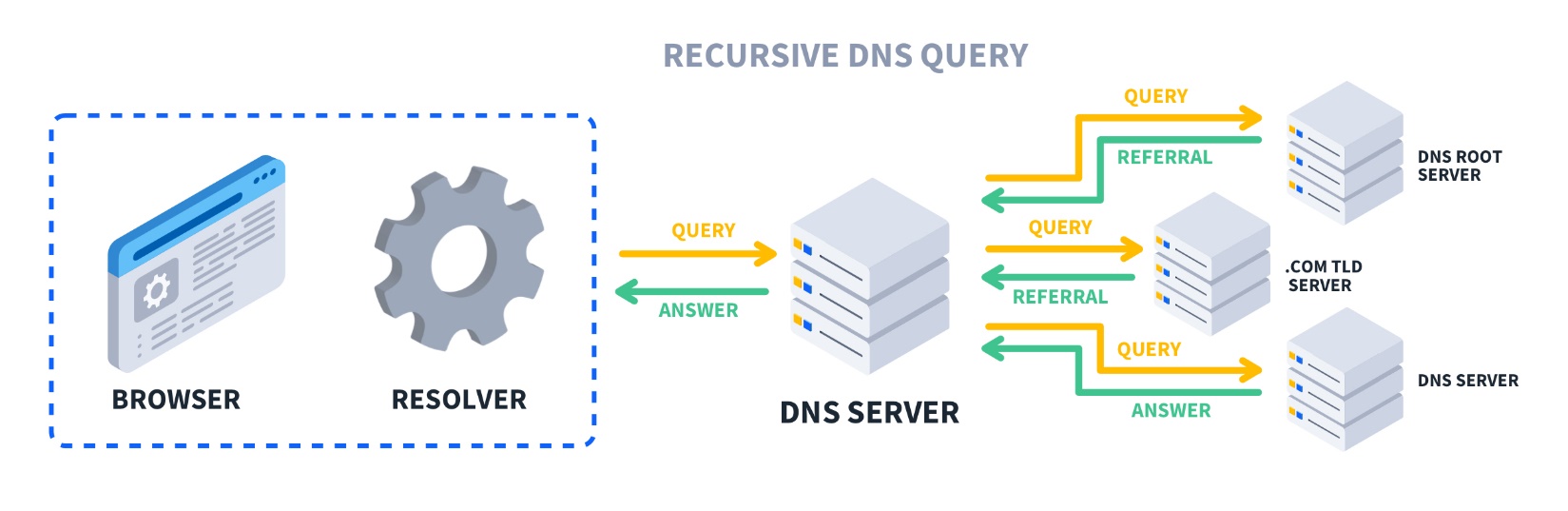
* **3600**: TTL (Time to Live).
* **IN**: RR class indicating this record belongs to the Internet.
* **A**: Record type mapping the domain to an IPv4 address.

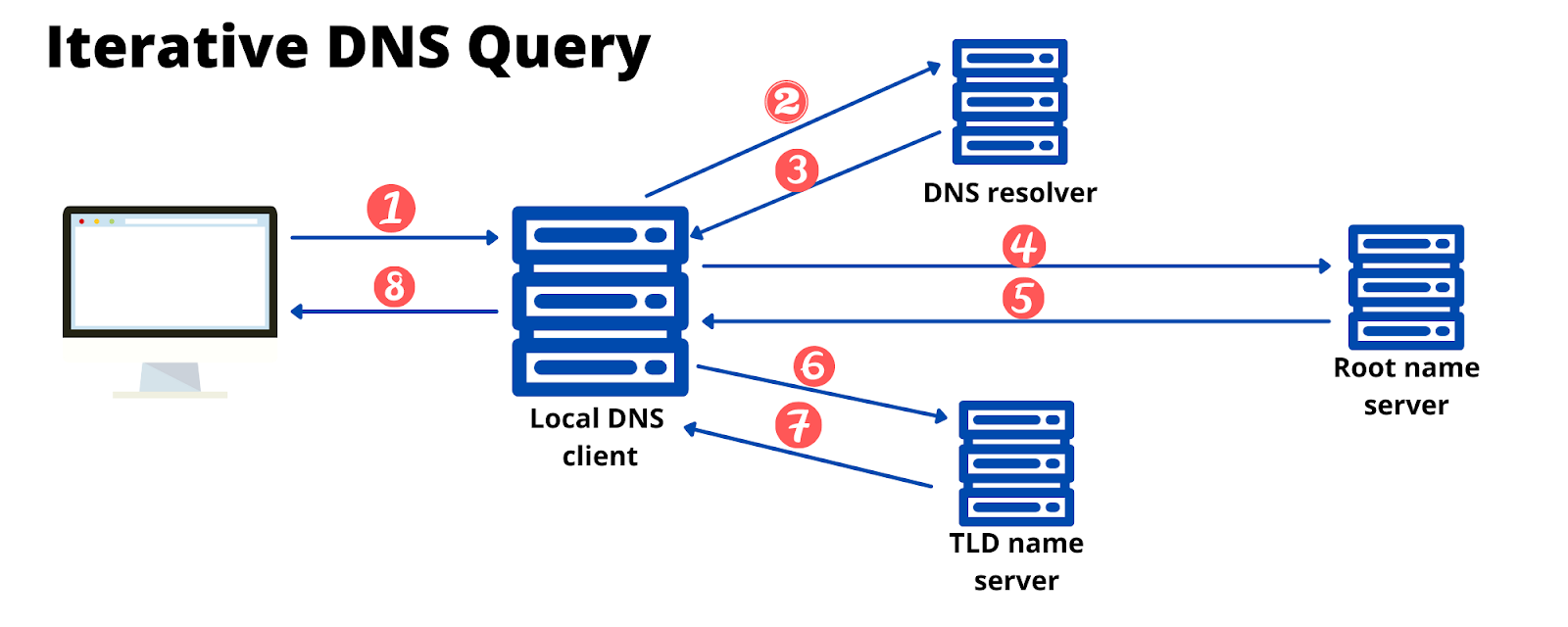
**Use Cases**

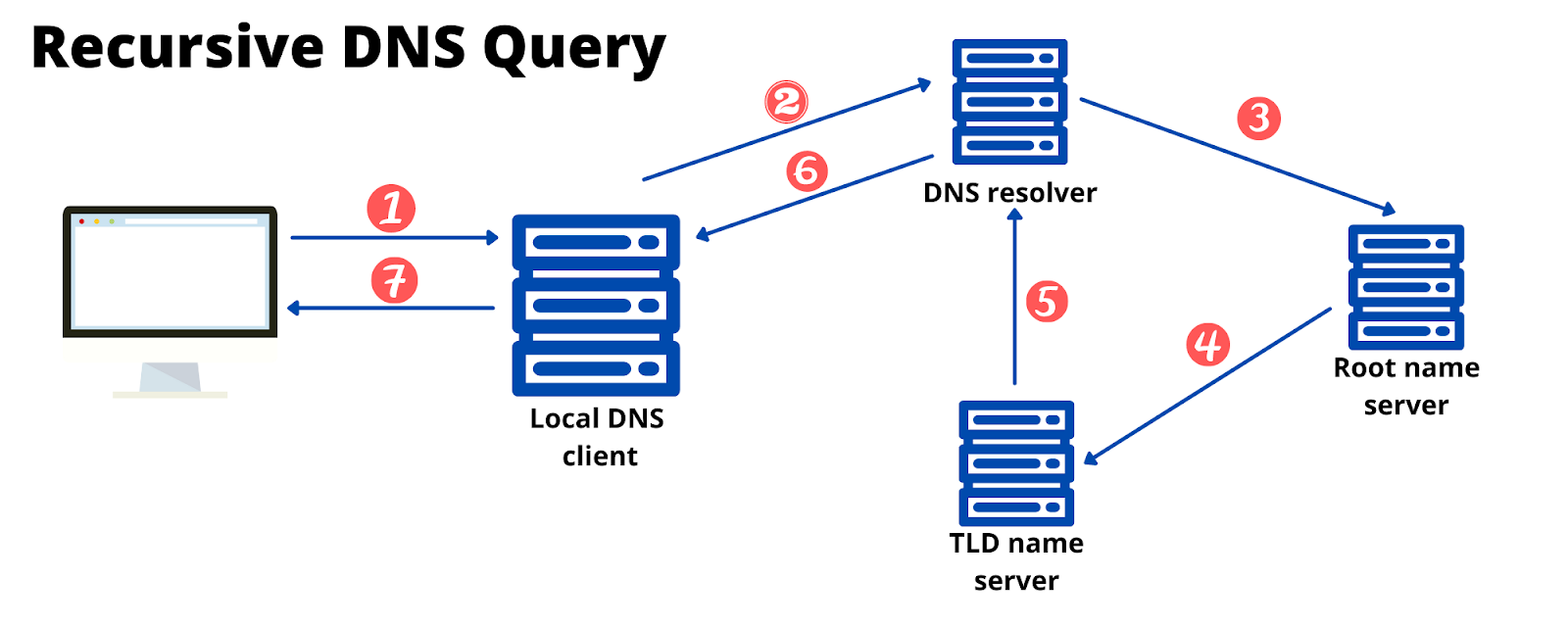
1. **Protocol Differentiation**:
   * RR classes allow DNS to handle multiple network protocols, though most use cases involve the IN class.
2. **Debugging and Metadata**:
   * The CH class is used for server-specific data retrieval, like software version queries.

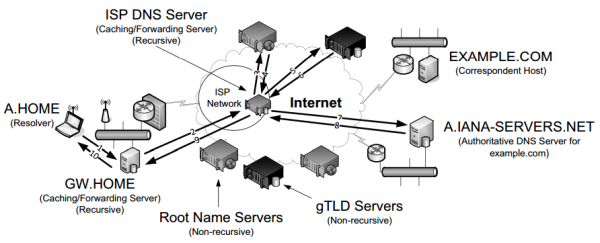


* **NAME**: The domain name to which the record applies.
* **TYPE**: The type of DNS record (e.g., A, MX, CNAME).
* **CLASS**: The protocol family (e.g., IN for Internet).
* **TTL**: Time-to-live value, defining how long the record can be cached.
* **RDLENGTH**: Length of the RDATA field in bytes.
* **RDATA**: The actual resource data, whose format depends on the record type.









A diagram of a system

Description automatically generated

