CSE 351 Computer Networks

Fall 2024/2025

Group 11

Submitted to:

Dr. Ayman Bahaa

Eng. Noha Wahdan

Submitted by:

Mohamed Mostafa Mamdouh 21P0244

Gaser Zaghloul Hassan 21P0052

Ali Tarek 21P0123

Abdelrahman Sherif 21P0098

Table of Contents

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.

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 DNS Servers

Authoritative servers are the definitive source of DNS information for a specific domain. These servers hold and manage the actual DNS records, such as A, CNAME, MX, and TXT records. When a query is directed to an authoritative server, it provides a response directly based on its maintained records, ensuring accuracy and reliability.

Key Characteristics:

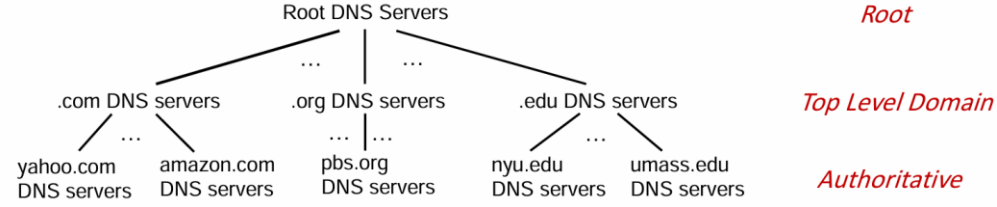
- Serve as the ultimate source for DNS information.
- Managed by the domain's owner, organization, or a designated service provider.
- Provide responses without needing to query other servers.

Non-Authoritative DNS Servers

Non-authoritative servers, on the other hand, do not store DNS records themselves. Instead, they function as intermediaries by caching information obtained from authoritative servers. When a query is received, a non-authoritative server checks its cache for the relevant data. If the requested data is unavailable or expired, the server queries an authoritative server to retrieve and store the information temporarily.

Key Characteristics:

- Depend on cached data for responses.
- Provide faster responses for cached queries but rely on authoritative servers for fresh data.
- Widely used in recursive resolution to enhance performance and reduce query load on authoritative servers.



Difference Between Authoritative and Non-Authoritative Responses:

Feature	Authoritative Response	Non-Authoritative Response
Source	Directly from an authoritative DNS server	From a caching or recursive resolver
Data Source	Zone file	Cached data
Authority Flag (AA)	1 (Authoritative)	0 (Non-Authoritative)
Example Use Case	Query to ns1.example.com for example.com	Query to a public DNS resolver like 8.8.8.8

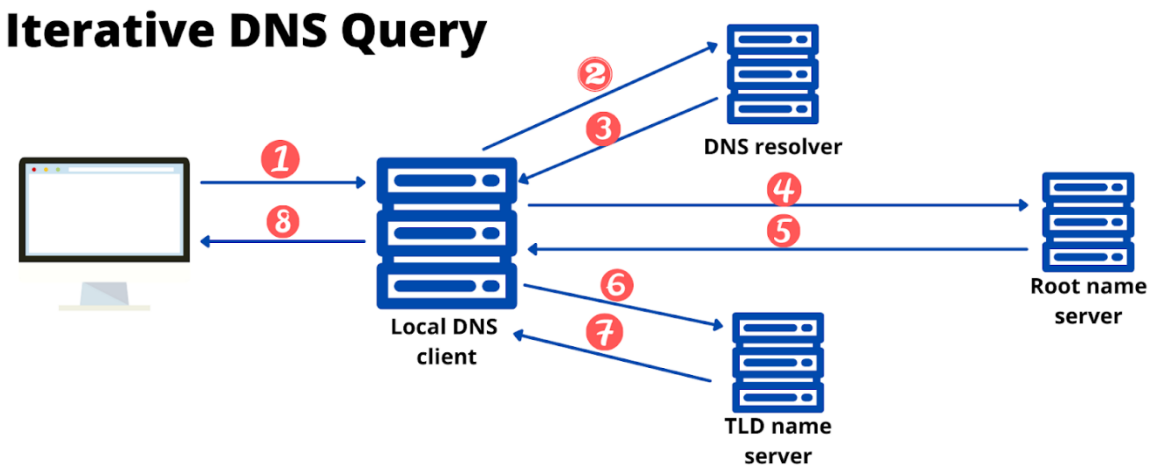
Queries can be of two types, Iterative or Recursive.

Iterative Query

In an iterative query, the DNS resolver takes an active role in the resolution process by querying multiple DNS servers step by step:

1. **Root Server**: The resolver starts by querying a root DNS server. The root server does not provide the final answer but directs the resolver to the appropriate **Top-Level Domain (TLD) server** (e.g., .com, .org).
2. **TLD Server**: The resolver queries the referred TLD server, which responds with the location of the **authoritative DNS server** for the requested domain.
3. **Authoritative Server**: The resolver queries the authoritative DNS server, which returns the requested resource record, such as an IP address.

The resolver continues this iterative process, handling multiple queries, until it retrieves the final answer.

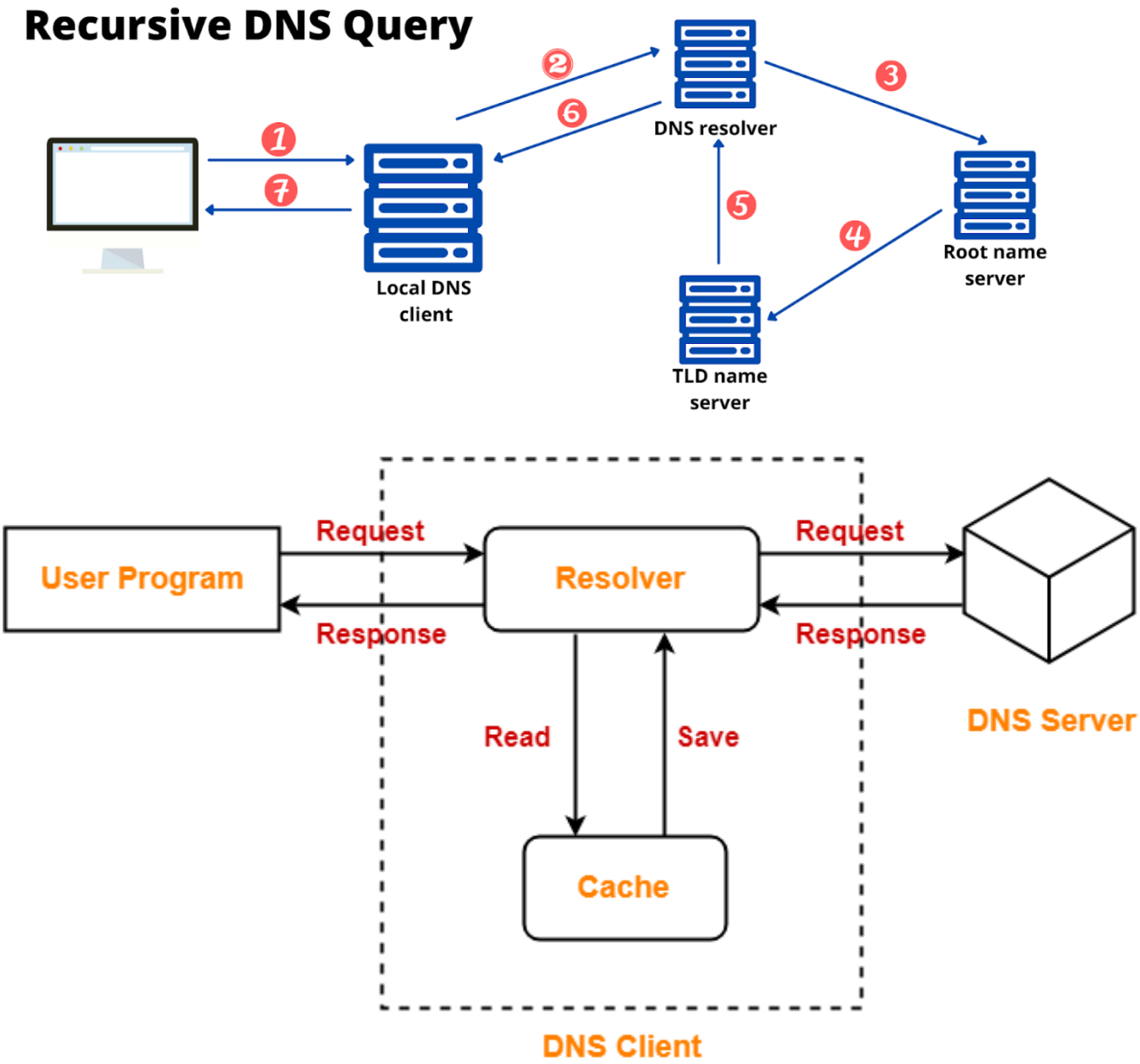


Recursive Query

In a recursive query, the resolver offloads the entire resolution responsibility to a single DNS server. The process unfolds as follows:

1. The resolver sends the query to a designated DNS server, typically an upstream caching resolver.
2. This server takes full responsibility for the query, recursively reaching out to other DNS servers (root, TLD, and authoritative) as needed.
3. Once the DNS server retrieves the final answer, it responds directly to the resolver.

Recursive queries simplify the resolver's role by relying on the DNS server to perform all necessary lookups, including contacting other servers and resolving referrals.



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

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

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

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.

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 DNS query types mentioned in RFC 1035.
- 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.
    - Handle recursive and iterative queries.
    - Generate appropriate responses for queries with correct format and TTL values.
- **Error Handling**:
    - Return RFC-compliant error codes.
- **Caching**:
    - Implement a caching mechanism to reduce response latency and improve efficiency.
    - Ensure consistency by respecting TTL values as specified in RFC 2181.

- **Security**:
    - Prevent unauthorized access or spoofing through use of ACL.

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, 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

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 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.

Communication Protocol Layer

This module handles encoding, decoding, and transmitting DNS messages in compliance with RFC 1035.

Key Functions

1. **Message Parsing and Construction**
    - Parses incoming DNS messages into their components: header, question, answer, authority, and additional sections.
    - Constructs outgoing DNS messages with the same structure.
2. **Protocol Handling**
    - **UDP**:
        - Default transport for most DNS queries.
        - Limits messages to 512 bytes unless EDNS (Extension Mechanisms for DNS) 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.
4. **Compression**
    - Reduces DNS response size using name compression techniques.
5. **Security Features**
    - Supports DNSSEC to sign outgoing responses and validate incoming queries.
    - Ensures 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 with responses or referrals.

Interactions

- Facilitates communication between internal modules and external systems.
- Interfaces with:

- The **resolver module** for query handling.
- The **authoritative server module** for data retrieval.

DNS Over UDP

1. **Default Transport Protocol**
   - DNS typically uses UDP on port 53 for queries and responses.
   - UDP is connectionless, meaning it does not establish a persistent connection.
2. **Why UDP?**
   - **Efficiency**: Faster than TCP because it avoids connection setup overhead.
   - **Small Responses**: Most DNS responses fit within the 512-byte limit of UDP.
3. **How It Works**
   - The client sends a query over UDP.
   - The server responds with the resource record if available.
   - If the response exceeds 512 bytes, the server sets the **TC (Truncated)** flag, prompting the client to retry over TCP.
4. **Limitations**
   - **No delivery guarantee**: Lost packets are not retransmitted.
   - **Vulnerabilities**: Susceptible to spoofing and amplification attacks.

DNS Over TCP

1. **When Is TCP Used?**
   - **Truncated Responses**: For responses exceeding 512 bytes, such as DNSSEC data or large zone transfers.
   - **Zone Transfers**: Full zone file transfers (e.g., AXFR, IXFR).
   - **Security Enforcements**: Some resolvers and firewalls enforce TCP for added security.
2. **Why TCP?**
   - **Reliable Transmission**: Ensures data delivery and retransmits if packets are lost.
   - **Larger Packets**: Supports responses without truncation.
   - **Security**: Less prone to spoofing due to connection establishment.
3. **How It Works**

- A TCP connection is established on port 53.
- The client sends the query, and the server responds with the full data.
- The connection is terminated after the exchange.

4. **Limitations**
   - **Higher Latency**: Connection setup requires additional time (SYN-ACK handshake).
   - **Increased Resource Usage**: Consumes more resources on both the client and server.

Security Considerations

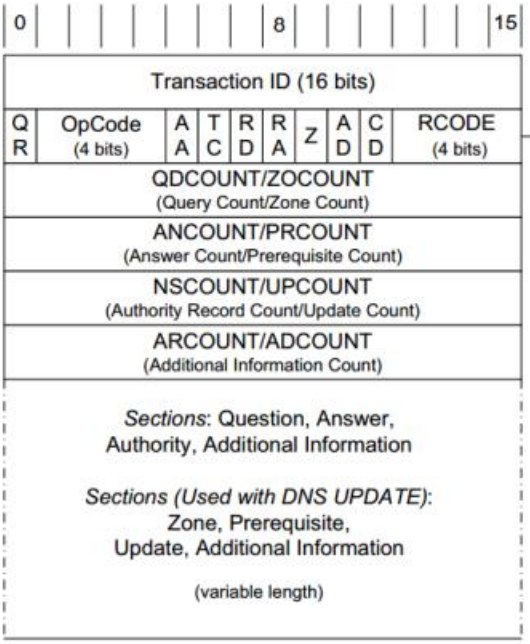
Each module incorporates security measures:

1. **Resolver Module**:
   - Verifies the legitimacy of incoming queries.
   - Protects against amplification attacks by limiting response size and rate.
2. **Authoritative Server Module**:
   - 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.

4. Communication Protocols

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

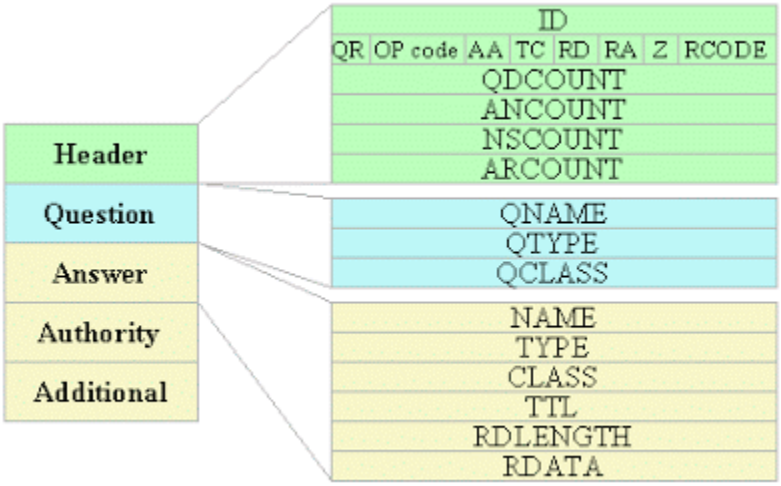
DNS message format

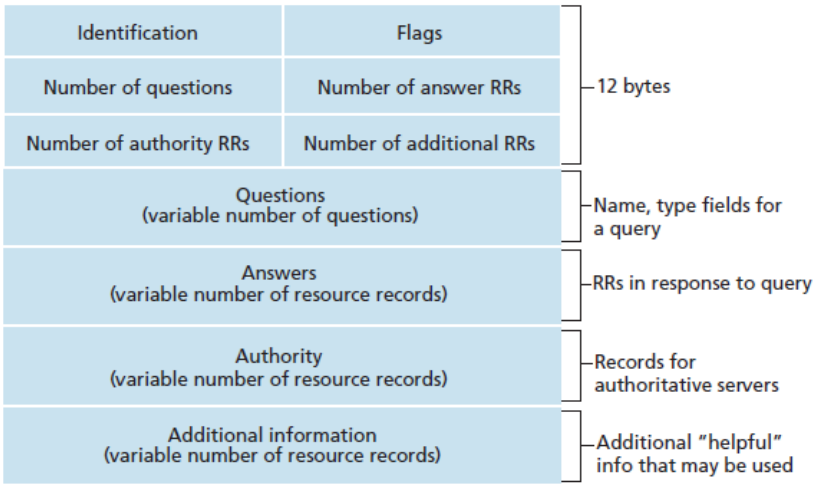


Flags:
QR: Query(0)/Response(1)
AA: Authoritative Answer
TC: Truncated Answer
RD: Recursion Desired
RA: Recursion Available
Z: Zero
AD: Authentic Data [RFC4035]
CD: Checking Disabled [RFC4035]

OpCodes (common values):
Query (0) – Regular Query
Notify (4) – DNS NOTIFY [RFC1996]
Update (5) – DNS UPDATE [RFC2136]

RCODEs (common values):
NoError (0) – No Error
FormErr (1) – Format Error
ServFail (2) – Server Failure
NXDomain (3) – Non-existent Domain
NotImp (4) – Not Implemented
Refused (5) – Query Refused





Header section

The header Contains fields like query ID, flags, 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.

1. ID (Transaction ID)

- **Size**: 16 bits
- **Purpose**: A unique identifier to match queries with responses.
- **Value**: A random number generated by the client for each query.

2. **Flags**

Bit(s)	Field	Description
0	QR (Query/Response)	Indicates whether the message is a query (0) or a response (1).
1-4	Opcode	Specifies the type of query. Common values: 0000 (Standard Query), 0001 (Inverse Query), 1000 (Update).
5	AA (Authoritative Answer)	Set to 1 if the responding server is authoritative for the domain (used in responses only).
6	TC (Truncated)	Set to 1 if the message was truncated due to size limits.
7	RD (Recursion Desired)	Set to 1 if the client requests recursive resolution.
8	RA (Recursion Available)	Set to 1 if the server supports recursion (used in responses only).
9	Z (Reserved)	Reserved for future use; must be 0.

Bit(s)	Field	Description
10	AD (Authenticated Data)	Set to 1 if the data in the response has been authenticated (used with DNSSEC).
11	CD (Checking Disabled)	Set to 1 to indicate that the client wants to disable DNSSEC validation.
12-15	RCODE (Response Code)	Indicates the status of the response.

- **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).

3. Question Count (QDCOUNT)

- **Size**: 16 bits
- **Purpose**: Specifies the number of questions in the **Question Section**.
- **Value**: Typically 1 for standard queries.

4. Answer Record Count (ANCOUNT)

- **Size**: 16 bits
- **Purpose**: Specifies the number of answers in the **Answer Section** (used in responses).

5. Authority Record Count (NSCOUNT)

- **Size**: 16 bits
- **Purpose**: Specifies the number of records in the **Authority Section** (used in responses for delegation or referrals).

6. Additional Record Count (ARCOUNT)

- **Size**: 16 bits
- **Purpose**: Specifies the number of records in the **Additional Section** (used in responses for extra information, like glue records).

Summary of DNS Header Layout

Offset	Field	Size	Description
0-15	ID	16 bits	Unique transaction ID to match queries and responses.
16	QR	1 bit	Query (0) or Response (1).
17-20	Opcode	4 bits	Query type (e.g., Standard, Inverse, Update).
21	AA	1 bit	Authoritative Answer flag.
22	TC	1 bit	Truncated flag.
23	RD	1 bit	Recursion Desired flag.
24	RA	1 bit	Recursion Available flag.
25	Z	1 bit	Reserved for future use.
26	AD	1 bit	Authenticated Data flag (DNSSEC).
27	CD	1 bit	Checking Disabled flag (DNSSEC).
28-31	RCODE	4 bits	Response Code indicating success or error status.
32-47	QDCOUNT	16 bits	Number of questions in the Question Section.
48-63	ANCOUNT	16 bits	Number of answers in the Answer Section.
64-79	NSCOUNT	16 bits	Number of authority records in the Authority Section.
80-95	ARCOUNT	16 bits	Number of additional records in the Additional Section.

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.

Question section

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.
- **Required QTYPE Values**:

TYPE	Value	Meaning
A	1	A host address
NS	2	An authoritative name server
MD	3	A mail destination (Obsolete - use MX)
MF	4	A mail forwarder (Obsolete - use MX)
CNAME	5	The canonical name for an alias
SOA	6	Marks the start of a zone of authority
MB	7	A mailbox domain name (EXPERIMENTAL)
MG	8	A mail group member (EXPERIMENTAL)
MR	9	A mail rename domain name (EXPERIMENTAL)
NULL	10	A null RR (EXPERIMENTAL)
WKS	11	A well-known service description
PTR	12	A domain name pointer

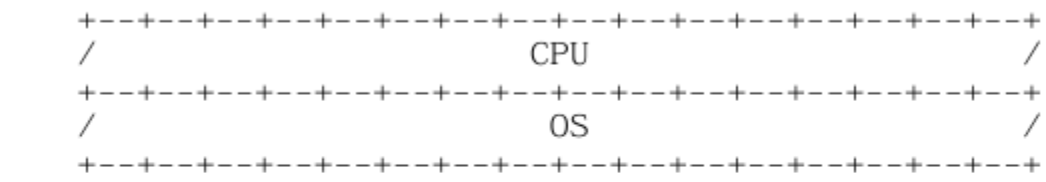
TYPE	Value	Meaning
HINFO	13	Host information
MINFO	14	Mailbox or mail list information
MX	15	Mail exchange
TXT	16	Text strings
AXFR	252	Request for transfer of an zone.
MAILB	253	Request for mailbow-related records (MB, MG or MR).
MAILA	254	Request for mail agents RRs (obsolete).
*	255	Request for all records

3.3.1. CNAME RDATA format

where:

CNAME A <domain-name> which specifies the canonical or primary name for the owner. The owner name is an alias.

3.3.2. HINFO RDATA format



where:

CPU A <character-string> which specifies the CPU type.

OS A <character-string> which specifies the operating system type.

3.3.3. MB RDATA format (EXPERIMENTAL)

where:

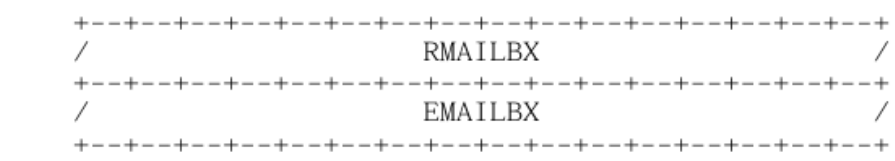
MADNAME A <domain-name> which specifies a host which has the specified mailbox.

3.3.6. MG RDATA format (EXPERIMENTAL)

where:

MGMNAME A <domain-name> which specifies a mailbox which is a member of the mail group specified by the domain name.

3.3.7. MINFO RDATA format (EXPERIMENTAL)

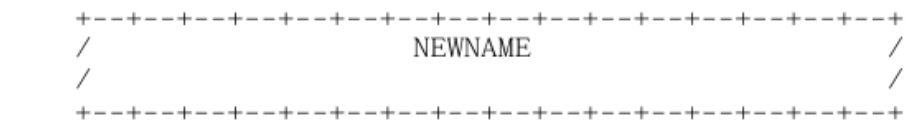


where:

RMAILBX A <domain-name> which specifies a mailbox which is responsible for the mailing list or mailbox. If this domain name names the root, the owner of the MINFO RR is responsible for itself. Note that many existing mailing lists use a mailbox X-request for the RMAILBX field of mailing list X, e.g., Msgroup-request for Msgroup. This field provides a more general mechanism.

EMAILBX A <domain-name> which specifies a mailbox which is to receive error messages related to the mailing list or mailbox specified by the owner of the MINFO RR (similar to the ERRORS-TO: field which has been proposed). If this domain name names the root, errors should be returned to the sender of the message.

3.3.8. MR RDATA format (EXPERIMENTAL)

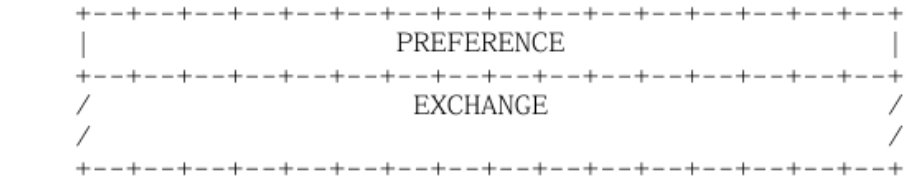


where:

NEWNAME A <domain-name> which specifies a mailbox which is the proper rename of the specified mailbox.

MR records cause no additional section processing. The main use for MR is as a forwarding entry for a user who has moved to a different mailbox.

3.3.9. MX RDATA format

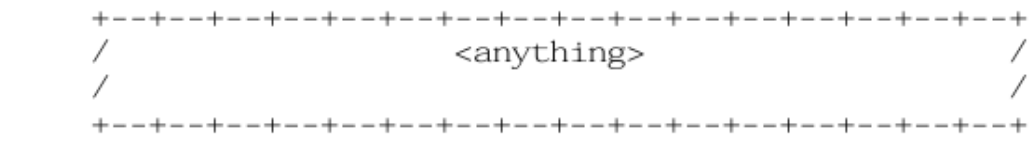


where:

PREFERENCE A 16 bit integer which specifies the preference given to this RR among others at the same owner. Lower values are preferred.

EXCHANGE A <domain-name> which specifies a host willing to act as a mail exchange for the owner name.

3.3.10. NULL RDATA format (EXPERIMENTAL)



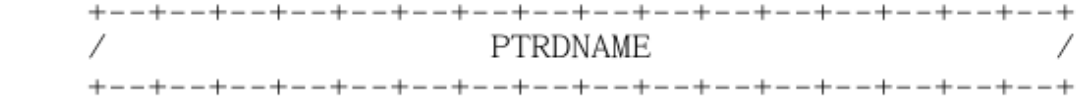
Anything at all may be in the RDATA field so long as it is 65535 octets or less.

3.3.11. NS RDATA format

where:

NSDNAME A <domain-name> which specifies a host which should be authoritative for the specified class and domain.

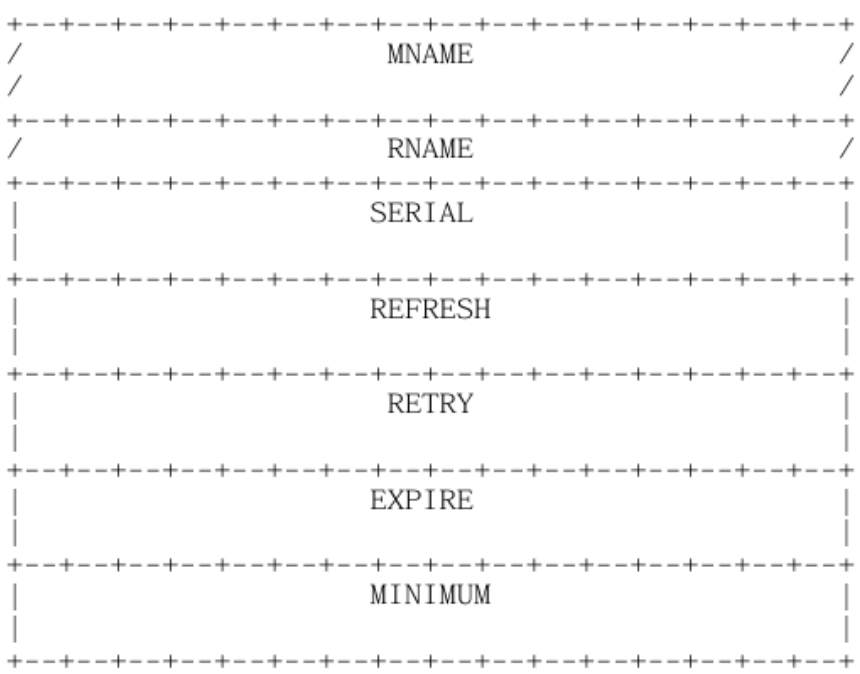
3.3.12. PTR RDATA format



where:

PTRDNAME A <domain-name> which points to some location in the domain name space.

3.3.13. SOA RDATA format



where:

MNAME	The <domain-name> of the name server that was the original or primary source of data for this zone.
RNAME	A <domain-name> which specifies the mailbox of the person responsible for this zone.
SERIAL	The unsigned 32 bit version number of the original copy of the zone. Zone transfers preserve this value. This value wraps and should be compared using sequence space arithmetic.
REFRESH	A 32 bit time interval before the zone should be refreshed.
RETRY	A 32 bit time interval that should elapse before a failed refresh should be retried.
EXPIRE	A 32 bit time value that specifies the upper limit on the time interval that can elapse before the zone is no longer authoritative.

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)
CS	2	The CSNET class (obsolete)
CH	3	Chaosnet (debugging, legacy)
HS	4	Hesiod (used in Project Athena)
*	255	Any class

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).

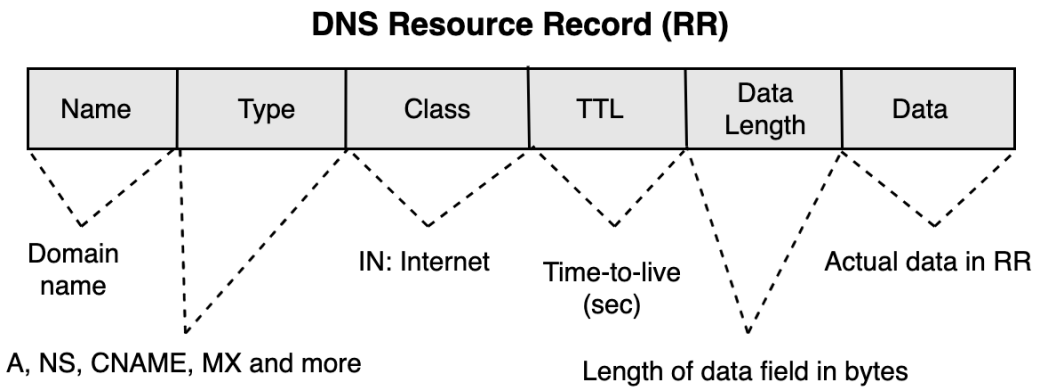
In DNS, **Resource Record (RR) Class** defines the network or protocol family to which the resource record applies. It is a field in every DNS resource record and is used to differentiate between various types of data stored in the DNS.

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.



- **NAME**: The domain name to which the record applies.
- **TYPE**: The type of DNS record.
- **CLASS**: The protocol family.
- **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.

This format is used in **answer, authority** and **additional sections**.

Format of DNS Messages Sent Between Components

DNS messages (queries and responses) are exchanged using a specific format defined in the DNS protocol. The message is structured as follows:

---

1. User to Resolver

- **Type of Query**: Recursive Query
- **Message Format**:
    - **Header**:
        - Query ID: A unique 16-bit identifier for the query.
        - Flags:
            - QR = 0 (indicating a query).
            - RD = 1 (Recursion Desired) to request the resolver to perform the entire resolution process.
        - Counts:
            - Questions = 1 (the query sent by the user).
            - Answers = 0.
            - Authority = 0.
            - Additional = 0.
    - **Question Section**

---

2. Resolver to Root Server

- **Type of Query**: Iterative Query
- **Message Format**:
    - **Header**:
        - Query ID: A unique ID to match responses.
        - Flags:
            - QR = 0 (indicating a query).
            - RD = 0 (Recursion Not Desired) since it's an iterative query.
        - Counts:
            - Questions = 1.
            - Answers = 0.

        - Authority = 0.
        - Additional = 0.
    - **Question Section**

---

3. Root Server to Resolver

- **Type of Response**: Iterative Response
- **Message Format**:
    - **Header**:
        - Query ID: Matches the query ID sent by the resolver.
        - Flags:
            - QR = 1 (indicating a response).
            - RA = 0 (Recursion Not Available).
        - Counts:
            - Questions = 1.
            - Answers = 0.
            - Authority = 1 (contains the TLD name servers).
            - Additional = 1 (contains the IP addresses of TLD name servers, if available).
    - **Question Section**: Echoes the resolver's query.
    - **Authority Section**: Lists TLD name servers (e.g., ns1.tld.com).
    - **Additional Section**: Provides IP addresses of the TLD name servers.

---

4. Resolver to TLD Name Server

- **Type of Query**: Iterative Query
- **Message Format**:
    - Same as the query to the root server, but now:
        - QNAME: The domain name excluding subdomains (e.g., example.com).
        - QTYPE: Typically NS or A.
    - **Header**: RD = 0.

5. TLD Name Server to Resolver

- **Type of Response**: Iterative Response
- **Message Format**:
    - Similar to the root server's response:
        - Authority Section: Lists authoritative name servers for the domain (e.g., ns1.example.com).
        - Additional Section: IP addresses of authoritative name servers (if available).

---

6. Resolver to Authoritative Name Server

- **Type of Query**: Iterative Query
- **Message Format**:
    - **Question Section**

---

7. Authoritative Name Server to Resolver

- **Type of Response**: Final Response
- **Message Format**:
    - **Answer Section**: Contains the requested resource record (e.g., A record with the IP address of www.example.com).
    - **Authority Section**: May include additional authoritative information.
    - **Additional Section**: Includes any related records (e.g., glue records for faster resolution).

---

8. Resolver to User

- **Type of Response**: Recursive Response
- **Message Format**:
    - **Answer Section**: Contains the resolved IP address of the domain (e.g., 192.0.2.1 for www.example.com).
    - Other sections are typically omitted unless additional data is required.

Errors Defined in the DNS Protocol

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.

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.

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, 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.

Inter-module Communication

The modules are loosely coupled to ensure abstraction:

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.

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).

User Authentication Mechanisms

Authentication will prevent unauthorized access and ensure secure data transmission.

Proposed Methods:

1. **Access Control Lists (ACLs)**:
    - Restrict which IP addresses or subnets can send queries to the server.
    - Limit zone transfers to specific trusted servers.
2. **Logging and Monitoring**:
    - Maintain logs of incoming queries and responses to detect and prevent abuse (e.g., DNS amplification attacks).

Implementation Plan

1. **Develop Core Components**:
    - Build resolver, root server, TLD server, authoritative server, and caching functionality.
    - Implement basic query handling for 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.

An **authoritative zone** is a segment of the Domain Name System (DNS) namespace for which a specific DNS server has the authority to manage and provide definitive responses. The DNS server responsible for an authoritative zone is called an **authoritative name server**.

Glue records are **special DNS resource records** that provide the IP addresses of name servers to prevent circular dependencies during DNS resolution. They are critical when the **name server for a domain** is within the domain it manages.

DNS query resolution process algorithm on an authoritative name server or a recursive resolver

1. Set or clear the value of recursion available in the response depending on whether the name server is willing to provide recursive service. If recursive service is available and requested via the RD bit in the query, go to step 5, otherwise step 2.

2. Search the available authoritative zones for the zone which is the nearest ancestor to QNAME. If such a zone is found, go to step 3, otherwise step 4.

3. Start matching down, label by label, in the zone. The matching process can terminate several ways:

    a. If the whole of QNAME is matched, we have found the node. If the data at the node is a CNAME, and QTYPE doesn't match CNAME, copy the CNAME RR into the answer section of the response, change QNAME to the canonical name in the CNAME RR, and go back to step 1. Otherwise, copy all RRs which match QTYPE into the answer section and go to step 6.

    b. If a match would take us out of the authoritative data, we have a referral. This happens when we encounter a node with NS RRs marking cuts along the bottom of a zone. Copy the NS RRs for the subzone into the authority section of the reply. Put whatever addresses are available into the additional section, using glue RRs if the addresses are not available from authoritative data or the cache. Go to step 4.

    c. If at some label, a match is impossible (i.e., the corresponding label does not exist), look to see if a the "*" label exists.

    If the "*" label does not exist, check whether the name we are looking for is the original QNAME in the query or a name we have followed due to a CNAME. If the name is original, set an authoritative name error in the response and exit. Otherwise just exit. If the "*" label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the "*" label. Go to step 6.

4. Start matching down in the cache. If QNAME is found in the cache, copy all RRs attached to it that match QTYPE into the answer section. If there was no delegation from authoritative data, look for the best one from the cache, and put it in the authority section. Go to step 6.

5. Using the local resolver or a copy of its algorithm to answer the query. Store the results, including any intermediate CNAMEs, in the answer section of the response.

6. Using local data only, attempt to add other RRs which may be useful to the additional section of the query. Exit

Types of Nodes in the IN-ADDR.ARPA Domain

1. **Address Nodes**
    - These nodes are used to represent full host addresses in the **IN-ADDR.ARPA** domain.
    - **Address nodes** are at the **deepest level** of the hierarchy, where all four labels are specified. Each label corresponds to one octet of the IP address.
    - For example, for the IP address **10.2.0.52**, the **address node** is 52.0.2.10.IN-ADDR.ARPA, where the labels (52, 0, 2, and 10) correspond to the four octets of the IP address, and the domain ends in .IN-ADDR.ARPA.
    - These nodes store **PTR (pointer) records**, which link the IP address to its associated hostname (for example, a PTR record linking 52.0.2.10.IN-ADDR.ARPA to MILNET-GW.ISI.EDU).
2. **Network Nodes**
    - **Network nodes** represent **network numbers** (not individual host addresses) within the **IN-ADDR.ARPA** domain. These nodes exist at higher levels in the hierarchy.
    - A **network node** corresponds to an IP address block, often representing a range of IP addresses assigned to a particular network. Network nodes typically correspond to 1, 2, or 3 octets of an IP address.
    - For example, the network node for **10** in the **IN-ADDR.ARPA** domain would be 10.IN-ADDR.ARPA. This node would hold **PTR records** for all the gateways within the **10.x.x.x** network, pointing to their hostnames. If a query is made for 10.IN-ADDR.ARPA, it will return the PTR records pointing to the gateway hostnames within that network.
    - **Network nodes** are typically used to point to the **primary hostnames of gateways** that are connected to that particular network. Since gateways are connected to multiple networks, a gateway may have several associated network nodes, each corresponding to a different network that the gateway is a part of.
3. **Delegation of Zones**
    - The **IN-ADDR.ARPA** domain allows for the **delegation of zones** at different levels of the hierarchy, enabling the management of IP address blocks by different entities.
    - For instance, **10.IN-ADDR.ARPA** could be a zone managed by a specific organization, while **26.IN-ADDR.ARPA** could represent a separate zone for another network, such as MILNET.
    - The **network nodes** (like 10.IN-ADDR.ARPA or 26.IN-ADDR.ARPA) represent delegated zones, and they can point to other lower-level nodes or even other zones that manage further details about a network's infrastructure.

Delegation is the process of assigning responsibility for a portion of the DNS namespace (e.g., a subdomain) to another set of name servers. It is implemented using **NS records**, which specify the authoritative servers for the delegated zone. Glue records (A or AAAA records) may be added in the parent zone to provide the IP addresses of these name servers if they belong to the delegated domain.

Link Between Delegation and SOA

1. **Delegation**: Occurs at the parent zone (e.g., example.com delegating sub.example.com).
    - Parent zone contains **NS records** for the subdomain pointing to its name servers.
2. **SOA Record**: Resides in the subdomain's authoritative zone file (e.g., sub.example.com.zone) and provides the management and update details for that zone.

Together, delegation defines the handoff of responsibility, while the SOA record governs the operation of the delegated zone.

A **stub resolver** is a simple, lightweight component of the DNS resolution process that operates on client devices, such as a computer or smartphone. It is responsible for initiating DNS queries but relies on a more capable DNS resolver (usually a recursive resolver) to complete the task.

Characteristics of Stub Resolvers

1. **Minimal Functionality**:
    - A stub resolver is not designed to perform full DNS resolution. Instead, it forwards DNS queries to a **recursive resolver**.
    - It is typically embedded in operating systems or network applications.
2. **Limited Capability**:
    - Stub resolvers do not have their own cache for DNS records.
    - They only perform simple tasks, like forming queries and passing responses back to the application.
3. **Dependency on Recursive Resolvers**:
    - The stub resolver relies entirely on a recursive resolver to handle the DNS resolution process, including querying root, TLD, and authoritative name servers.

How Stub Resolvers Work

1. **Query Formation**:
    - An application, such as a web browser, requests a DNS resolution (e.g., translating www.example.com to its IP address).
    - The stub resolver generates a DNS query packet and sends it to a recursive resolver (usually configured via the network's DHCP or manually set, like Google DNS 8.8.8.8).

2. **Delegation to Recursive Resolver**:
    - The recursive resolver takes over and performs the complete DNS resolution process, starting from the root server if necessary.
    - Once the recursive resolver has the answer, it returns the response to the stub resolver.
3. **Response Delivery**:
    - The stub resolver forwards the response to the application that made the request.

Master files refer to **zone files** stored on **primary (master) DNS servers**. These files contain the complete set of **resource records (RRs)** for a DNS zone, defining the authoritative information for the domains managed by that server. Each line represents a RR, with fields for name, TTL, class, type and value. Secondary DNS servers obtain a copy via zone transfers (AXFR or IXFR). A **zone transfer** is a process where a secondary DNS server requests the full zone file from a primary DNS server to synchronize records. Zone transfers require **TCP** because:

1. TCP ensures reliability for large data transfers.
2. The zone file often exceeds the size limits of UDP.