COMPUTER NETWORKS

PROJECT REPORT

DOMAIN NAME SYSTEM (DNS)

GROUP 2, A3

SUBMISSION: 25/09/2024

TOPIC: DOMAIN NAME SYSTEM IMPLEMENTATION AND DEPLOYMENT

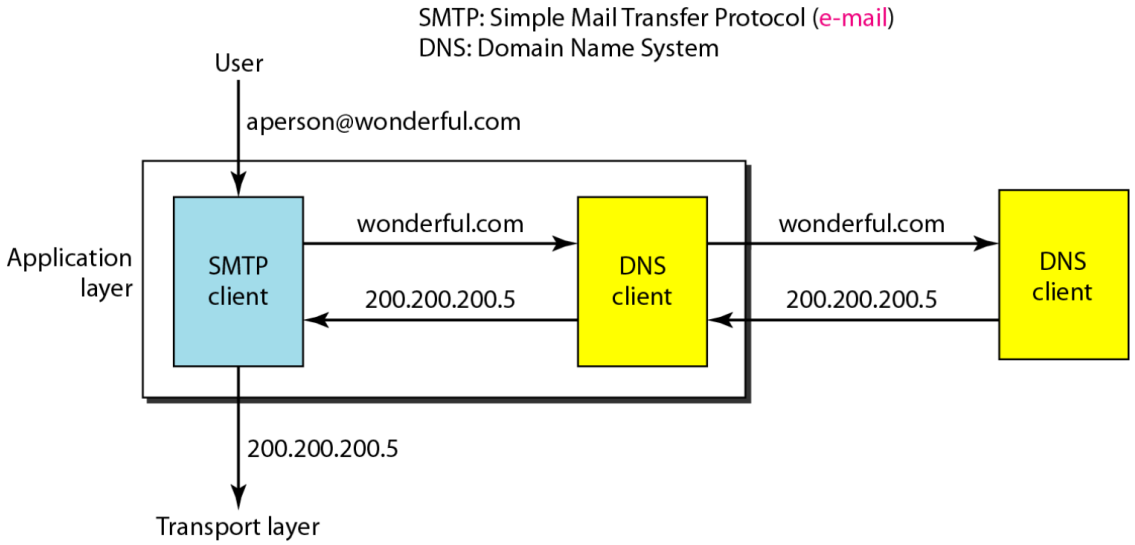
NAME	ROLL	WORK
SUBHOJIT BHATTACHERJEE	002210501104	Cache and it's replacement strategy.
SUJAL KYAL	002210501105	Deployment, Client-Server interface.
DEBJIT DHAR	002210501106	Authoritative servers, Local DNS.
SOHAM LAHIRI	002210501107	DNS packet format, Global DNS call and Client.

OVERALL FUNCTIONALITY

1. Client and DNS Server Interface
2. Client sends the query to the dns server in the required packet format.
3. The dns server tries to resolve the query based on the data in its cache.
4. The cache uses LRU replacement strategy and each entry in the cache has a ttl field.
5. If a cache miss occurs the dns server tries an iterative resolution using the root and the local authoritative servers(com, edu, gov and in).
6. Subsequent failure results in the global dns call.
7. Deployed in AWS.

THEORY:

The **Domain Name System (DNS)** is a hierarchical system that translates human-readable domain names (like www.example.com) into machine-readable IP addresses (like 200.200.200.5). This is essential because, while humans prefer easy-to-remember domain names, computers require numerical IP addresses to communicate.



---

1. Domain Name Space

The domain name space is a hierarchical structure that organizes domain names systematically. It consists of several levels:

a. Root Domain

- Represented by a dot (.).
- It is the starting point of the DNS hierarchy and managed by ICANN.
- Below the root domain are **Top-Level Domains (TLDs)**.

b. Top-Level Domains (TLDs)

- These are classified into:
  - **Generic TLDs (gTLDs):** e.g., .com, .org, .net.

- **Country Code TLDs (ccTLDs):** e.g., .uk (United Kingdom), .jp (Japan).

c. Second-Level Domains

- Domains under TLDs, often chosen by organizations or individuals (e.g., example in example.com).

d. Subdomains

- Parts of a domain name that fall under the second-level domain (e.g., www in www.example.com).

e. Fully Qualified Domain Name (FQDN)

- A complete domain name specifying its position in the hierarchy, including all levels (e.g., www.example.com. with the trailing dot representing the root).

---

2. DNS Resolution Process

DNS resolution is the process of converting a domain name into an IP address. It involves two methods of query resolution:

a. Iterative Resolution

- The client contacts DNS servers one by one until it finds the answer.
- Process:
  1. The client sends a query to a local DNS resolver.
  2. If the resolver doesn't have the record, it queries the root server.
  3. The root server directs the resolver to a TLD server.
  4. The TLD server points to the authoritative server for the domain.
  5. The authoritative server provides the IP address.
- Example:
  - The client asks, "What is the IP for example.com?"

    - The resolver systematically queries each server until it receives the final IP.

b. Recursive Resolution

- The DNS resolver takes on the responsibility of querying multiple servers to provide the client with the final answer.
- Process:
    1. The client sends a query to a DNS resolver.
    2. The resolver recursively queries root, TLD, and authoritative servers.
    3. It returns the final IP address to the client.
- Example:
    - The client asks, "What is the IP for example.com?"
    - The resolver handles the entire resolution process and returns the answer without further queries from the client.

---

3. DNS Caching

DNS caching is a mechanism that stores previous DNS query results temporarily to reduce resolution time for subsequent queries.

a. Purpose of Caching

- Improves performance and reduces latency by avoiding repetitive queries.
- Reduces traffic to DNS servers.

b. Cache Levels

1. **Client-Side Cache**: Stored in the operating system or browser of the user's device.
2. **Resolver Cache**: Stored by DNS resolvers, such as ISP-provided resolvers or third-party resolvers like Google DNS.

3. **Authoritative Server Cache**: Authoritative servers may also cache records to respond faster to frequently queried domains.

c. Time-to-Live (TTL)

- Each DNS record has a TTL value that specifies how long it can be cached.
- Once the TTL expires, the cached record is considered stale, and a new query is made to ensure up-to-date information.

Summary of DNS Components and Operations

Component	Description
Root Servers	The first point of contact in the DNS hierarchy.
TLD Servers	Manage domains under specific TLDs like .com or .org.
Authoritative Servers	Provide definitive answers for specific domain names.
Resolvers	Intermediaries that query DNS servers on behalf of clients.
Cache	Stores DNS query results temporarily for faster lookups.

The DNS system ensures a reliable and efficient way to navigate the internet by converting domain names into IP addresses while balancing query resolution efficiency through caching and hierarchical querying.

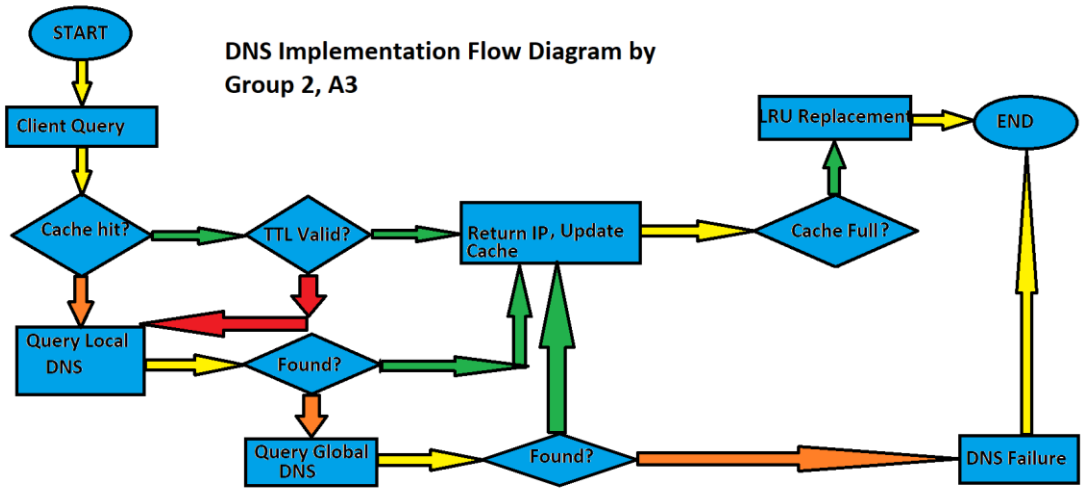
Sender Program:

This program is designed to simulate the process of error detection and correction in data communication. It reads binary data from a file and then simulates errors during transmission by modifying the data using different error types. The program supports two error detection schemes: Checksum and Cyclic Redundancy Check (CRC). Based on the user's choice, it computes the appropriate checksum or CRC remainder, appends it to the data, and then sends the modified data (with simulated errors) over a network socket to a receiver. This demonstrates how error detection methods help identify and potentially correct errors that occur during data transmission.

Structural Diagram

Implementable code at:

https://github.com/sujalkyal/my_dns_server.git



IMPLEMENTATION
(CODE SNIPPETS AND EXPLANATION)

LOCAL DNS RESOLUTION

```
AUTHORITATIVE_SERVERS = {
    "com": 5058,
    "edu": 5055,
    "gov": 5056,
    "in": 5057,
}

def query_authoritative_server(tld, domain_name):
    """Query an authoritative TLD server."""
    if tld not in AUTHORITATIVE_SERVERS:
        print(f"No authoritative server found for TLD: {tld}")
        return None

    server_address = ('0.0.0.0',
AUTHORITATIVE_SERVERS[tld])
    client = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
        # Serialize the DNS_MESSAGE before sending
        message = DNS_MESSAGE()
        message.create(domain_name)
        serialized_message = message.serialize()

        # Send the serialized message
        client.sendto(serialized_message, server_address)
        data, _ = client.recvfrom(512)
        response = DNS_MESSAGE.deserialize(data)
```

```
        return response.answer  # Expect the IP address in the
answer field
    except Exception as e:
        print(f"Error querying authoritative server: {e}")
        return None
    finally:
        client.close()



def handle_query(data, addr, server):
    message = DNS_MESSAGE.deserialize(data)
    domain_name = message.question
    print(f"Received query for: {domain_name} from {addr}")

    cached_ip = get_from_cache(domain_name)
    if cached_ip:
        print(f"Cache hit for {domain_name}. Sending IP:
{cached_ip}")
        response = DNS_MESSAGE()
        response.create(domain_name, cached_ip, 1)
        server.sendto(response.serialize(), addr)
        return

    tld = domain_name.split('.')[-1]
    ip_address = query_authoritative_server(tld, domain_name)
    if ip_address != 'NOT_FOUND':
        print(f"TLD server resolved {domain_name} to
{ip_address}")
        response = DNS_MESSAGE()
        response.create(domain_name, ip_address, 1)
```

```
        server.sendto(response.serialize(), addr)
        update_cache(domain_name, ip_address)
        return

    external_ip = query_external_root(domain_name)
    if external_ip:
        print(f"External DNS resolved {domain_name} to
{external_ip}")
        response = DNS_MESSAGE()
        response.create(domain_name, external_ip, 1)
        server.sendto(response.serialize(), addr)
        update_cache(domain_name, external_ip)
    else:
        print(f"Failed to resolve {domain_name}. Sending
NOT_FOUND.")
        response = DNS_MESSAGE()
        response.create(domain_name, "NOT_FOUND", 1)
        server.sendto(response.serialize(), addr)
```

EXPLANATION:

AUTHORITATIVE_SERVERS

Key Components of the Code

1. **AUTHORITATIVE_SERVERS**
    - A dictionary mapping top-level domains (TLDs) to their respective port numbers.
    - Example: The authoritative server for .com TLD operates on port 5058.
2. **query_authoritative_server(tld, domain_name)**
    - Queries the authoritative server for a given TLD to resolve a domain name.

Function Details

- **Input:**
  - tld: The top-level domain (e.g., com, edu).
  - domain_name: The domain name to resolve (e.g., example.com).
- **Process:**

1. Verifies if the TLD exists in the AUTHORITATIVE_SERVERS dictionary.
2. Creates a UDP client socket (socket.SOCK_DGRAM).
3. Constructs a DNS query using a custom DNS_MESSAGE class (assumed to be defined elsewhere).
4. Serializes the DNS query and sends it to the TLD's authoritative server.
5. Waits for a response, deserializes it, and extracts the answer.

- **Output:**
  - Returns the resolved IP address (or an error message if the query fails).

---

3. **handle_query(data, addr, server)**
   - Processes incoming client queries, checks cache, queries TLD authoritative servers, and provides responses.

Function Details

- **Input:**
  - data: The raw DNS query message sent by the client.
  - addr: The address of the client (IP and port).
  - server: The server socket handling the query.
- **Process:**

1.Deserialize Query:

- Converts the raw data into a readable format using the DNS_MESSAGE.deserialize method.
- Extracts the domain name being queried (domain_name).

2.Cache Lookup:

- Checks if the domain's IP address is in the cache using get_from_cache(domain_name).
- If found, sends the cached IP to the client and returns.

3. Query Authoritative Server:

- Determines the TLD from the domain name (e.g., example.com → com).
- Queries the corresponding authoritative server using query_authoritative_server(tld, domain_name).
- If a valid IP is returned, it is sent to the client, and the cache is updated.

4.Fallback to External Root DNS:

- If the authoritative server fails to resolve the domain, the function queries an external root DNS using query_external_root(domain_name) (assumed to be defined elsewhere).
- The returned IP address is sent to the client and cached.

5. Error Handling:

- If no resolution succeeds, the server responds with "NOT_FOUND".

Flow Summary

1. **Receiving Query:**
    - A client sends a DNS query for domain_name.
2. **Cache Lookup:**
    - If cached, the server sends the cached IP back to the client.
3. **Query Authoritative Server:**
    - The server extracts the TLD and queries the corresponding authoritative server.
    - If resolved, the result is cached and sent to the client.
4. **Fallback:**
    - If the authoritative server cannot resolve the query, the server queries an external root DNS.
5. **Error Response:**
    - If all queries fail, the server responds with "NOT_FOUND".

AUTHORITATIVE SERVERS:

COM SERVER:

```
import socket
import pickle


# Domain data for .com TLD
COM_DOMAINS = {
    "example.com": "93.184.216.34",
    "localtest.com": "127.0.0.1",
    "abcd.com": "108.98.98.2",
}
def tld_server(port, domain_data):
    server = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
        server.bind(('0.0.0.0', port))
        print(f"TLD Server is running on port {port}...")


        while True:
            data, addr = server.recvfrom(512)
            #print(f"Raw data received: {data}")


            try:
```

```
            # Attempt to deserialize the data
            message = DNS_MESSAGE.deserialize(data)
        except Exception as e:
            print(f"Deserialization failed: {e}")
            continue  # Skip to the next iteration if deserialization fails

        domain_name = message.question
        print(f"TLD Server received query for: {domain_name}")

        # Search for the domain in the TLD's database
        ip_address = domain_data.get(domain_name, "NOT_FOUND")

        # Prepare and send the response
        response = DNS_MESSAGE()
        response.create(domain_name, ip_address, type=1)
        server.sendto(response.serialize(), addr)

    finally:
        server.close()
        print("TLD Server shut down.")
```

```
if __name__ == "__main__":
    tld_server(port=5058, domain_data=COM_DOMAINS)
```

EDU SERVER:

```
import socket
import pickle


# Domain data for .edu TLD
EDU_DOMAINS = {
    "example.edu": "192.0.2.45",
    "university.edu": "198.51.100.23"
}
def tld_server(port, domain_data):
    server = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
        server.bind(('0.0.0.0', port))
        print(f"TLD Server is running on port {port}...")


        while True:
            data, addr = server.recvfrom(512)
            #print(f"Raw data received: {data}")
```

```
            try:
                # Attempt to deserialize the data
                message = DNS_MESSAGE.deserialize(data)
            except Exception as e:
                print(f"Deserialization failed: {e}")
                continue  # Skip to the next iteration if
deserialization fails

            domain_name = message.question
            print(f"TLD Server received query for:
{domain_name}")

            # Search for the domain in the TLD's database
            ip_address = domain_data.get(domain_name,
"NOT_FOUND")

            # Prepare and send the response
            response = DNS_MESSAGE()
            response.create(domain_name, ip_address, type=1)
            server.sendto(response.serialize(), addr)

    finally:
        server.close()
        print("TLD Server shut down.")
```

```
if __name__ == "__main__":
    tld_server(port=5055, domain_data=EDU_DOMAINS)
```

GOV SERVER:

```
import socket
import pickle

# Domain data for .gov TLD
GOV_DOMAINS = {
    "example.gov": "203.0.113.10",
    "agency.gov": "198.51.100.55"
}

def tld_server(port, domain_data):
    server = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
        server.bind(('0.0.0.0', port))
        print(f"TLD Server is running on port {port}...")

        while True:
```

```
        data, addr = server.recvfrom(512)
        #print(f"Raw data received: {data}")

        try:
            # Attempt to deserialize the data
            message = DNS_MESSAGE.deserialize(data)
        except Exception as e:
            print(f"Deserialization failed: {e}")
            continue  # Skip to the next iteration if deserialization fails

        domain_name = message.question
        print(f"TLD Server received query for: {domain_name}")

        # Search for the domain in the TLD's database
        ip_address = domain_data.get(domain_name, "NOT_FOUND")

        # Prepare and send the response
        response = DNS_MESSAGE()
        response.create(domain_name, ip_address, type=1)
        server.sendto(response.serialize(), addr)
```

```
    finally:
        server.close()
        print("TLD Server shut down.")

if __name__ == "__main__":
    tld_server(port=5056, domain_data=GOV_DOMAINS)
```

IN SERVER:

```
import socket
import pickle

# Domain data for .in TLD
IN_DOMAINS = {
    "example.in": "203.119.86.101",
    "company.in": "198.51.100.33"
}
def tld_server(port, domain_data):
    server = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
    try:
        server.bind(('127.0.0.1', port))
        print(f"TLD Server is running on port {port}...")
```

```
        while True:
            data, addr = server.recvfrom(512)
            #print(f"Raw data received: {data}")

            try:
                # Attempt to deserialize the data
                message = DNS_MESSAGE.deserialize(data)
            except Exception as e:
                print(f"Deserialization failed: {e}")
                continue  # Skip to the next iteration if
deserialization fails

            domain_name = message.question
            print(f"TLD Server received query for:
{domain_name}")

            # Search for the domain in the TLD's database
            ip_address = domain_data.get(domain_name,
"NOT_FOUND")

            # Prepare and send the response
            response = DNS_MESSAGE()
            response.create(domain_name, ip_address, type=1)
```

```
            server.sendto(response.serialize(), addr)

    finally:
        server.close()
        print("TLD Server shut down.")

if __name__ == "__main__":
    tld_server(port=5057, domain_data=IN_DOMAINS)
```

EXPLANATIONS:

1. Shared Components Across All TLD Servers

a. Domain Data Dictionaries

Each TLD server has its dictionary containing the mapping of domain names to their corresponding IP addresses.

b. tld_server Function

This function starts a UDP server for a specific TLD. It listens for incoming DNS queries, processes them, and responds with the resolved IP address or a "NOT_FOUND" message.

1. **Input Parameters**:
    - port: The port number the server listens on (e.g., 5058 for .com).
    - domain_data: The dictionary of domain-to-IP mappings for the TLD.
2. **Steps in the Function**:

- **Setup**:
  - A UDP socket is created using socket.socket(socket.AF_INET, socket.SOCK_DGRAM).
  - The socket binds to 0.0.0.0 (or 127.0.0.1 for .in TLD), allowing it to listen for incoming queries.
- **Listening Loop**:
  - The server enters an infinite loop to handle incoming DNS queries.
  - Each incoming query:
    1. **Deserialize Query**:
       - The raw data received from the client is converted into a DNS_MESSAGE object using DNS_MESSAGE.deserialize(data).
    2. **Process Query**:
       - The domain name is extracted from the question field.
       - The domain's IP address is looked up in domain_data.
    3. **Create Response**:
       - A DNS_MESSAGE object is created with the domain name and either:
         - The resolved IP address, or

        - "NOT_FOUND" if the domain is not in the TLD's dictionary.

    4. **Send Response**:
        - The response message is serialized and sent back to the client using server.sendto().

- **Shutdown**:
    - The server socket is closed when the loop exits.

3. **Key Console Outputs**:
    - Indicates when the server starts, receives queries, and processes them.
    - Logs deserialization errors and the result of the domain lookup.

2. TLD Server Implementations

Each TLD server runs as a standalone process, handling domains specific to its TLD.

3. Execution Flow

1. **Start TLD Server**:
    - Each TLD server script is executed independently (e.g., python tld_server_com.py).
    - Each server binds to its respective port and begins listening.

2. **Query Processing**:
    - A client sends a DNS query to the appropriate TLD server (e.g., .com server at port 5058).

    - The server:
        - Deserializes the query.
        - Looks up the domain in its domain_data.
        - Creates and sends a response (resolved IP or "NOT_FOUND").
3. **Error Handling**:
    - If deserialization of the query fails, the server logs the error and continues.
    - If the domain is not found in the TLD database, a "NOT_FOUND" response is sent.

Limitations

- No hierarchical query handling (e.g., no root server directing queries to TLD servers).
- Limited to predefined domain-to-IP mappings (static data).
- No caching or optimization to reduce repeated queries.