University of Dhaka

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 4 : Distributed Database Management and Implementation of Iterative and Recursive Queries of DNS Records.

Submitted By:

Afser Adil Olin

Roll No : AE-47

Anika Tabassum

Roll No : Rk-61

Submitted On :

February 15, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

Contents

1 Introduction

The preliminary objective of this lab is to emulate the Domain Name Service (DNS) protocol and to understand the difference between iterative and recursive DNS resolution. The client can request IP address of his desired domain and the nameserver hierarchy will use the DNS resolution to return the IP address of the corresponding domain to the client if the domain name is valid.

1.1 Objectives

- We should design a distributed database architecture that is optimized for handling DNS records and resolving iterative and recursive queries.
- We should design and implement security mechanisms to protect the confidentiality and integrity of the data stored in the database.
- We should design and implement a user interface for accessing the database and submitting queries.
- We should measure the time it takes for the system to respond to different types of queries (iterative and recursive) under various conditions such as different data sizes, query loads, and network latencies.
- We should test the ability of the system to handle an increasing number of queries and data sizes, and measure the impact on response time and accuracy.

2 Theory

The preliminary objective of this lab is to emulate the Domain Name Service (DNS) protocol and to understand the difference between iterative and recursive DNS resolution. The client can request IP address of his desired domain and the nameserver hierarchy will use the DNS resolution to return the IP address of the corresponding domain to the client if the domain name is valid.

2.1 Search in DNS cache

When we ask our computer to resolve a hostname, the first place our computer looks is in its local DNS cache. If our computer doesn't already know the answer, it needs to perform a DNS query to find out.

2.2 Request to ISP's DNS servers

If the information is not stored locally, our computer queries our ISP's DNS servers.

2.3 Request to root nameservers

If those servers don't have the answer, they query the root nameservers. A nameserver is an always running computer that resolves queries about domain names, such as IP addresses.

2.4 Request to Top Level Domain nameservers

The root nameservers will look at the first part of our request, reading from right to left and direct our query to the Top-Level Domain (TLD) nameservers for .com (for google.com). Each TLD, such as .com, .org, and .bd, have their own set of nameservers, which act like a receptionist for each TLD.

2.5 Request to authoritative DNS servers

The TLD nameservers review the next part of our request and direct our query to the nameservers responsible for this specific domain. These authoritative nameservers are responsible for knowing all the information about a specific domain, which are stored in DNS records.

2.6 Retrieve the record

ISP's DNS server retrieves the record for google.com from the authoritative nameservers and stores the record in its local cache. If anyone else requests the host record for google.com, the ISP's servers will already have the answer and will not need to go through the lookup process again. All records have an expiration time. After a while, the server will need to ask for a new copy of the record to make sure the information doesn't become out-of-date.

2.7 Receive the answer

Armed with the answer, ISP's server returns the record back to our computer. Computer stores the record in its cache, reads the IP address from the record, then passes

3 Methodology

- We create a thread for each DNS server.
- We create a tree of DNS servers with a root node.
- Each server is provided with the reference to its parent server and child servers.
- The server awaits for request from client
- After receiving request, server looks for the requested domain in its current location and the child (if any). if not found, it will forward the request to the parent.
- Parent does similar search, if not found, it sends the request to its parents.
- If the domain can't be found in any of the DNS servers, then client will be prompted with an error message.
- If the domain is found in any of the servers, they will return to the requesting client.

4 Setting up the DNS server

This Python code is a simple DNS client that allows a user to enter a DNS query and receive the IP address of the corresponding domain.

4.1 Experimental Result

This python function that creates a DNS query for a given domain name. The function takes in a domain name parameter and returns the packed binary data representing the DNS query for that domain.

```
    #this functions helps to fetch dns records from
       dns_records.txt
    def getDnsRecords():
```

This python function helps to extract the IP address (A record) from the response by calling the get a() method on the parsed DNS record object. The returned value is a string with multiple fields separated by spaces, so the function splits the string and extracts the response and type of the record.

```
    #this function helps to send to the clients dns_records
    def handle_client(data, addr, server):
```

This python function for client helps to encode the dns record sent from the server.

```
    def encode_msg(message):
    #"Enter a domain name to send to the server: "
    #message = encode_msg(message)
    return packed_data
```

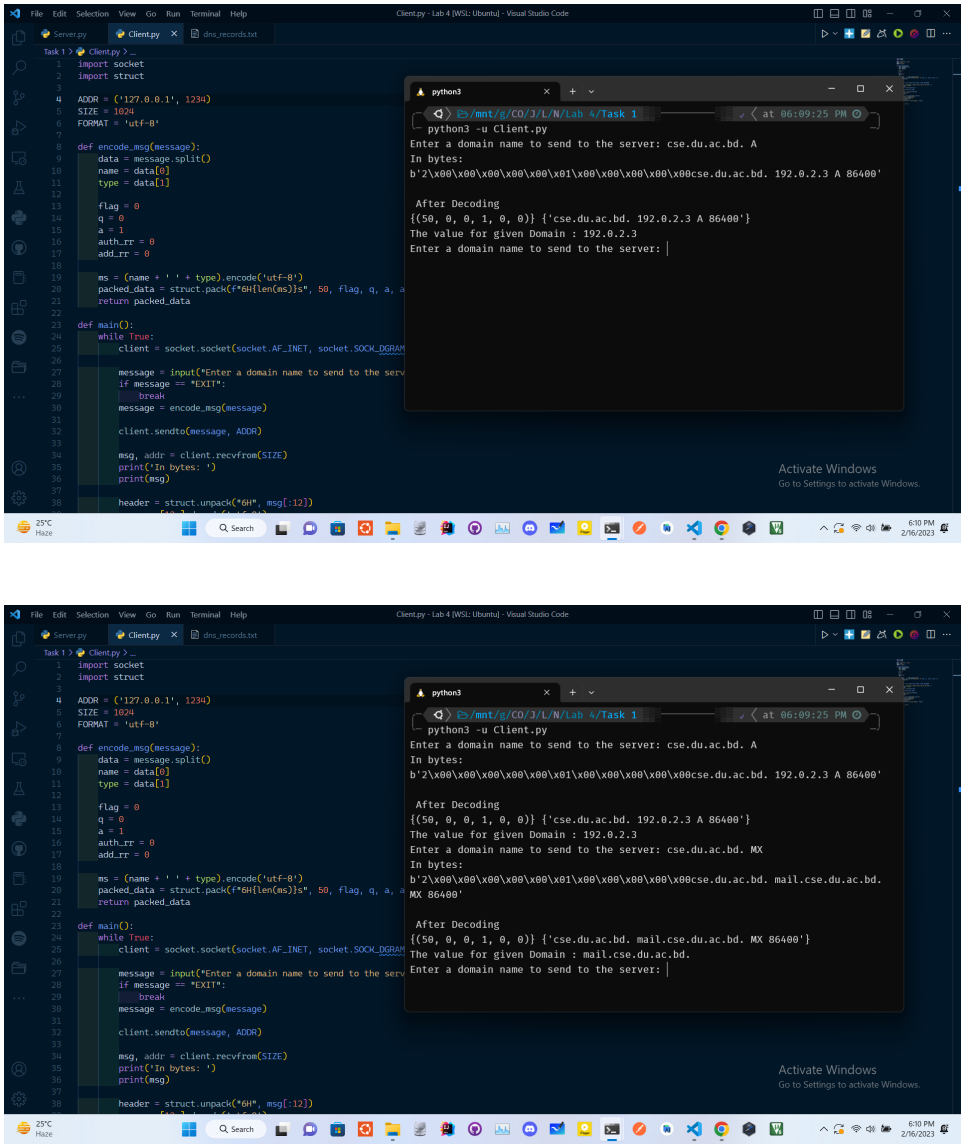

Figure 1: Client Side View

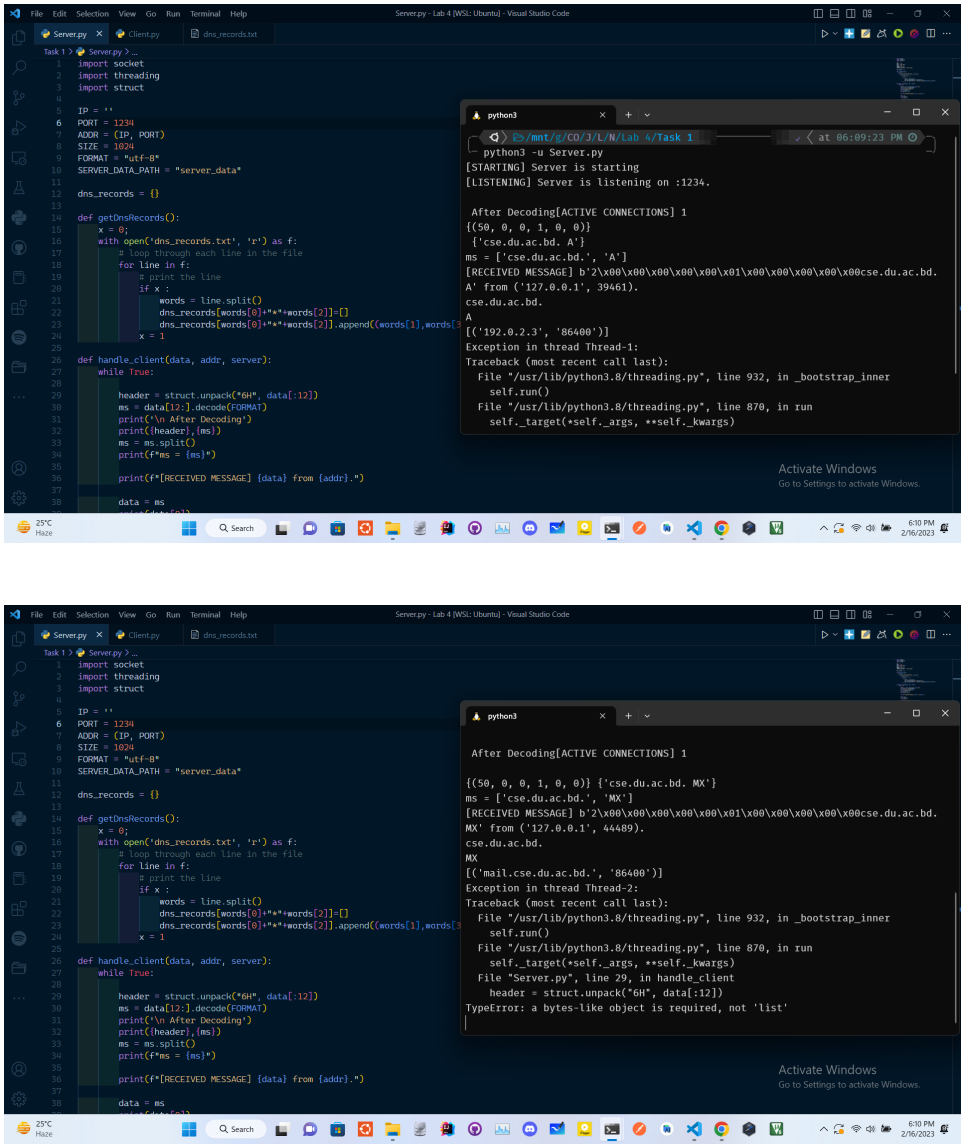

Figure 2: Server Side View

5 Iterative DNS resolution

Iterative DNS query is a type of DNS resolution process in which a DNS client directly queries the DNS root servers or a specific DNS server for each level of the DNS hierarchy until it finds the authoritative server that can provide the answer to the query. Our Python code enables a user to enter a domain name to look up, sends a DNS query to a specified DNS server, receives the DNS response from the server, and prints the IP address of the domain.

5.1 Experimental Result

Listing 1: For Authoratitive Server

```
9     def getDnsRecords():  # to fetch dns records
10    def handle_client(data, addr, server): #this function
          handles the threading for sending to the client the
          dns records
```

Listing 2: For TLD Server

```
11    def getDnsRecords():   # to fetch dns records
12
13    def handle_client(data, addr, server): #this function
          handles the threading for sending to the client the
          dns records
```

Listing 3: For Root Server

```
14    def getDnsRecords():  # to fetch dns records
15    def handle_client(data, addr, server): #this function
          handles the threading for sending to the client the
          dns records
```

Listing 4: For Client Side

```
16    #this functions helps the clients to request for dns
          records from Rootserver or TLD or Authorative server
17     def encode_msg(message):
```

```
18    #this functions helps the clients to decode message sent
          from the server
19      def decode_msg(msg):
```

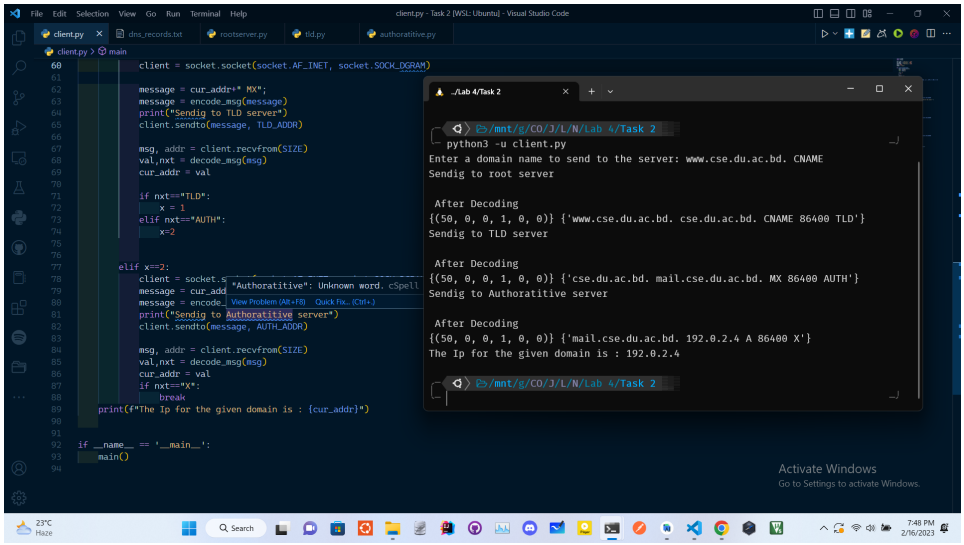

Figure 3: Client Side View

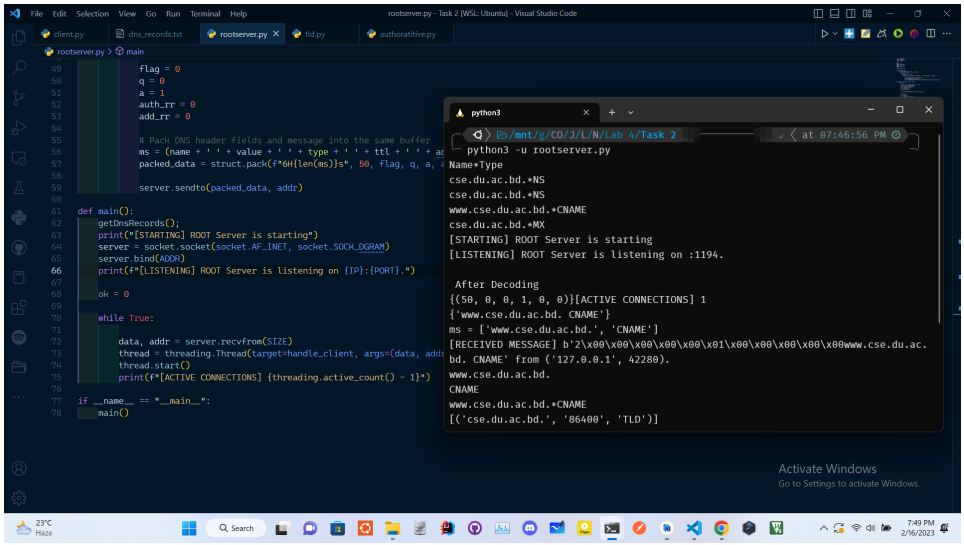

Figure 4: Root Server Side View

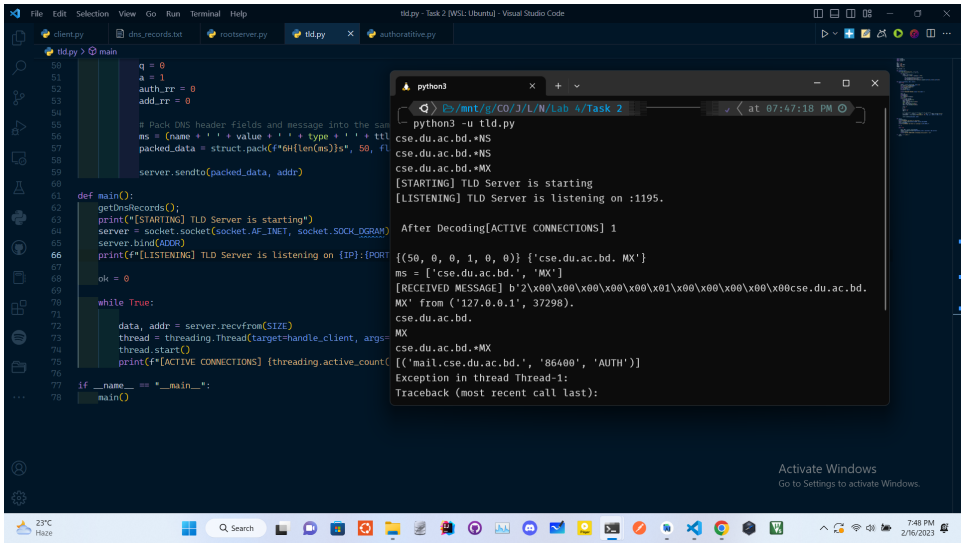

Figure 5: TLD Server Side View

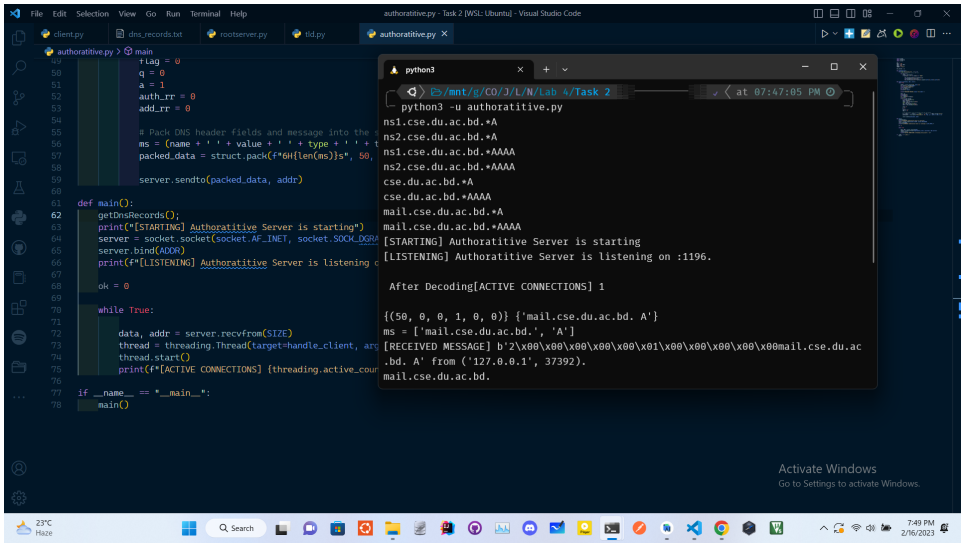

Figure 6: Authoratitive Server Side View

5.2 Advantages

- Faster Response Times: Iterative DNS queries typically result in faster response times, as the DNS server can respond with the best available information it has without waiting for other DNS servers to respond.
- More Resilient: Iterative queries are more resilient to network failures, as the client can retry the query with a different name server if one name server is not responding.
- Efficient Resource Utilization: Iterative queries can be more efficient in terms of resource utilization, as they avoid the need for recursive queries that can generate large amounts of network traffic and put more load on DNS servers.

5.3 Disadvantages

- Incomplete Responses: Since iterative queries do not rely on other DNS servers to provide a complete response, there is a risk that the DNS server may not have the complete information the client needs. In this case, the DNS server will respond with the best information it has, which may be incomplete or inaccurate.
- Increased Network Traffic: Since iterative queries require the client to send multiple requests to different DNS servers, it can result in increased network traffic, which can slow down the resolution process. This can be particularly problematic in high-traffic environments.
- Vulnerability to Cache Poisoning: Iterative queries can be vulnerable to cache poisoning attacks, in which an attacker can send fake responses to the client in an attempt to redirect them to a malicious website. To mitigate this risk, iterative queries should be made only to trusted name servers and over secure channels, such as DNS over HTTPS or DNS over TLS.

5.4 Security Implications

Iterative DNS queries can have security implications, including vulnerability to cache poisoning attacks, information leakage, malware propagation, and DNS hijacking. To mitigate these risks, clients should only query trusted name servers, use secure channels, and use security software to detect and block suspicious DNS queries. While iterative queries can provide faster

response times and better resource utilization, it is important to be aware of these security risks and take appropriate steps to mitigate them.

6 Recursive DNS resolution

When a DNS client sends a recursive query to a DNS server, the server will first check its cache to see if it has a record of the requested domain name. If the server does not have a cached record, it will query other DNS servers in a hierarchical fashion until it either finds the answer or reaches the authoritative DNS server for the requested domain Our python script enables a user to enter a domain name and recursively make queries to find it's IP address.It implements a simple DNS resolver client that can perform DNS queries by sending UDP packets to a DNS server.

6.1 Experimental Result

Listing 5: For Authoratitive Server

```
20    def getDnsRecords():  # to fetch "Authoratitve type dns
          records
21    def encode_msg(message): #to sort out the request from
          the client
22    return packed_data #encode the request in dns format
23
24    def handle_client(data, addr, server): #this function
          handles the threading for sending to the client the
          dns records
```

Listing 6: For TLD Server

```
25    def getDnsRecords():  # to fetch NX or MS dns records
26    def encode_msg(message): #to sort out the request from
          the client
27    return packed_data #encode the request in dns format
28
29    def decode_msg(msg): #decode the bit request from the
          client
30    return ms[1],ms[4]
31
32    def handle_client(data, addr, server): #this function
          handles the threading for sending to the client the
          dns records
```

Listing 7: For Root Server

```
33      def getDnsRecords():  # to fetch any dns records except
            the authoratitive dns
34      def encode_msg(message): #to sort out the request from
            the client
35      return packed_data #encode the request in dns format
36
37      def decode_msg(msg): #decode the bit request from the
            client
38      return ms[1],ms[4]
39
40      def handle_client(data, addr, server): #this function
            handles the threading for sending to the client the
            dns records
```

Listing 8: For Client Side

```
41    #this functions helps the clients to request for dns
          records from Rootserver or TLD or Authorative server
42     def encode_msg(message):
```

```
43    #this functions helps the clients to decode message sent
          from the server
44      def decode_msg(msg):
```

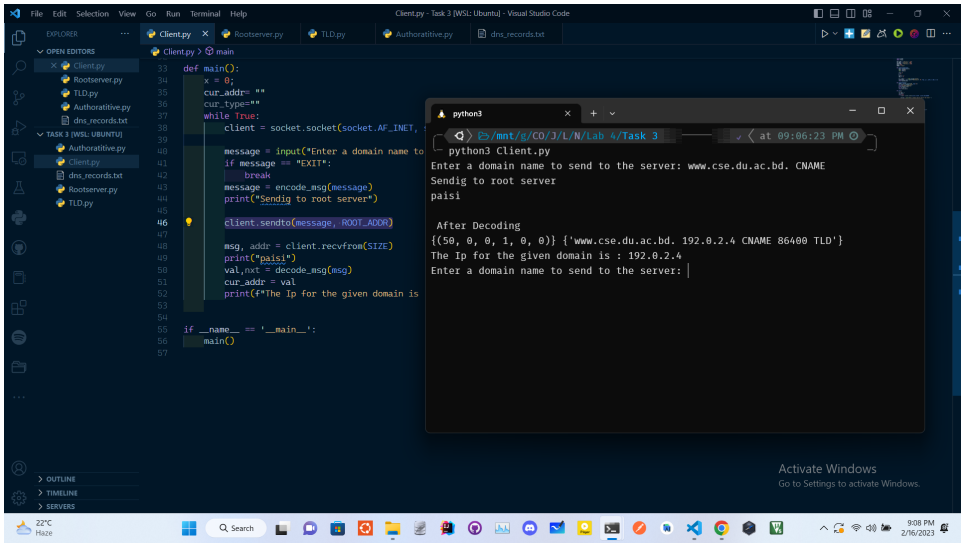

Figure 7: Client Side View

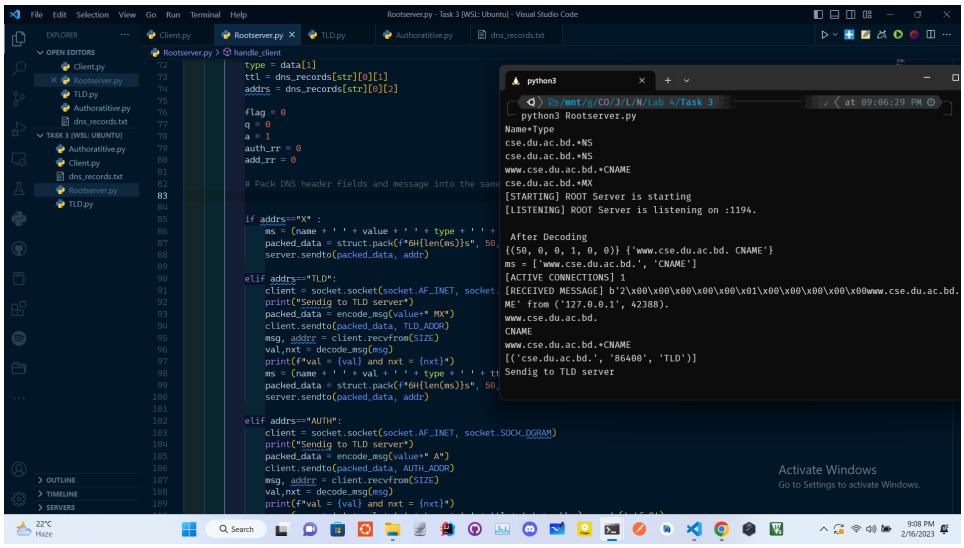

Figure 8: Root Server Side View

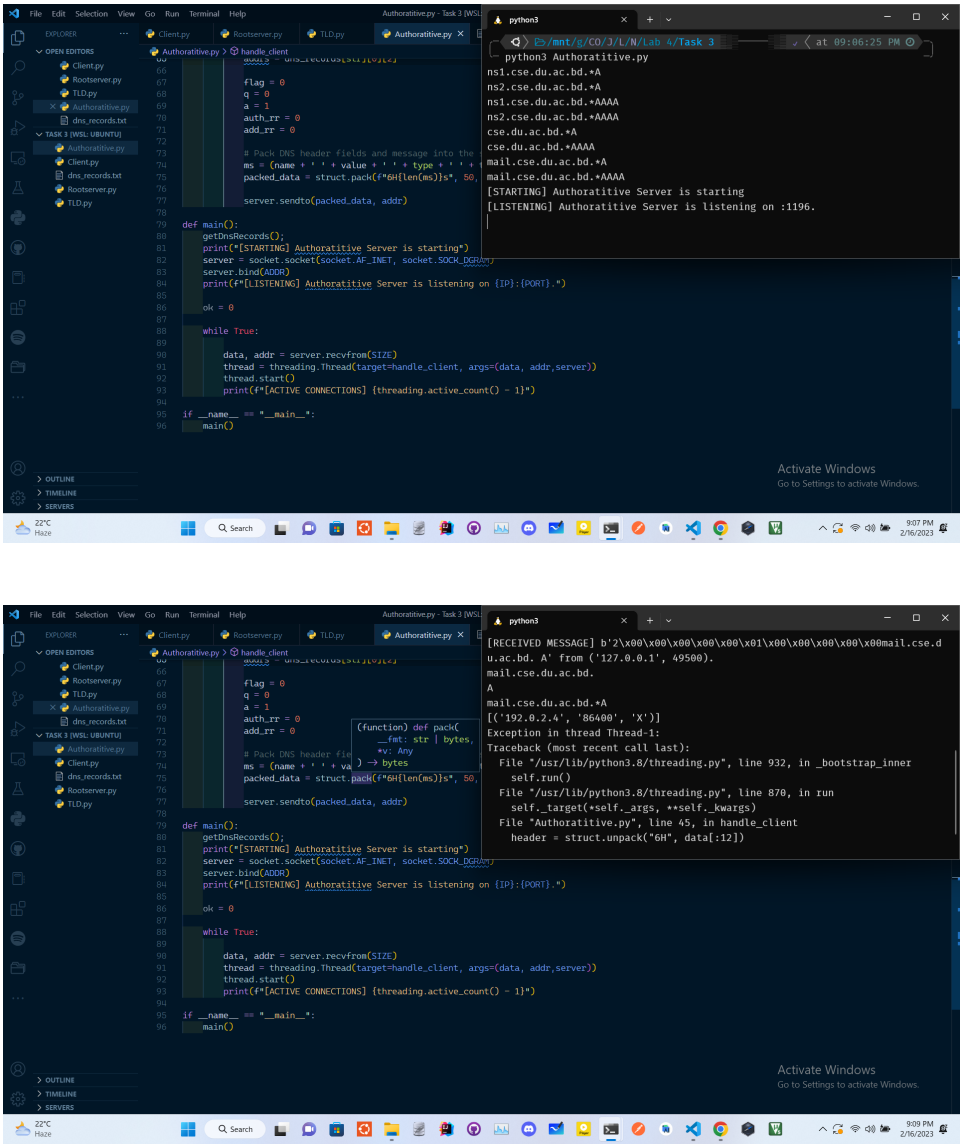

Figure 9: Authoratitive Server Side View

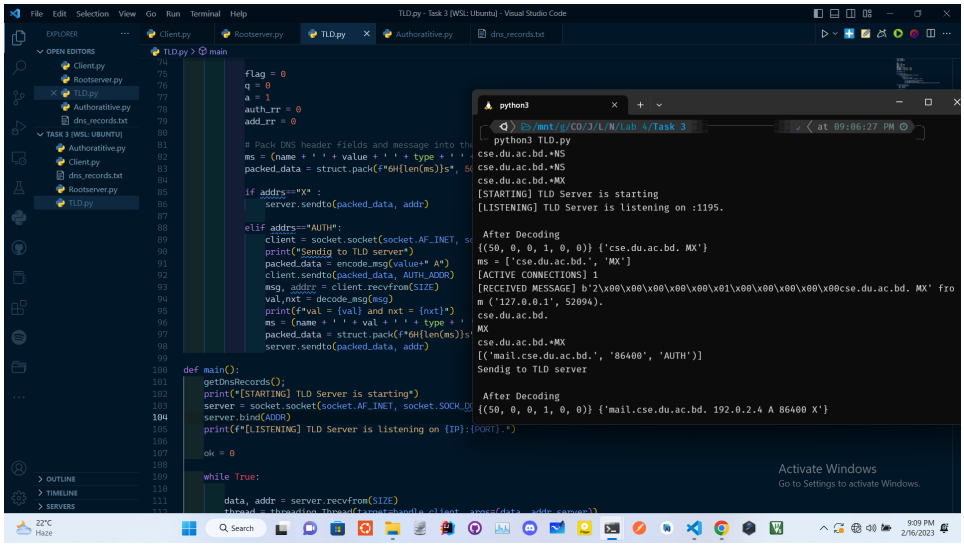

Figure 10: TLD Server Side View

6.2 Advantages

- Convenience: Recursive DNS queries are more convenient for clients because they provide a complete answer to the client's query without requiring the client to perform additional queries or have a deep understanding of DNS.
- Wide Range of Information: Recursive queries can provide a wide range of information, including IP addresses, domain names, and other related information.
- No Cache Poisoning Vulnerability: Recursive DNS queries are not vulnerable to cache poi- soning attacks, as the DNS server verifies the authenticity of each response it receives.

6.3 Disadvantages

- More Network Traffic: Recursive DNS queries generate more network traffic than iterative queries, as the DNS server has to query multiple name servers to get a complete answer. This can result in slower response times and increased network congestion.

- Higher Load on DNS Servers: Recursive queries can put more load on DNS servers, as they have to process and respond to multiple queries from different clients. This can result in slower response times or even server downtime during periods of high traffic.
- Security Risks: Recursive DNS queries can be vulnerable to security risks, such as DNS hijacking, where an attacker intercepts the client's DNS queries and responds with fake responses. This can allow the attacker to redirect the client to a malicious website or intercept sensitive information.

6.4 Security Implications

Recursive DNS queries can have security implications, including the potential for cache poisoning attacks, DNS hijacking, and other forms of DNS-based attacks. Recursive queries can be more vulnerable to these types of attacks than iterative queries, as the client relies on the DNS server to perform all the necessary queries. If the DNS server is not properly configured or is vulnerable to attacks, an attacker may be able to intercept the client's DNS queries and respond with fake responses, potentially redirecting the client to a malicious website or intercepting sensitive information. To mitigate these risks, it is important to use trusted DNS servers, implement secure channels for DNS queries, and use security software to detect and block suspicious DNS queries.

7 Extending the system

In this task we extend our system in two ways.We introduce DNS caching in our system.This will reduce the query time by storing the result of queries made previously.

7.1 Experimental Result

Listing 9: For Authoratitive Server

```
45      def getDnsRecords():  # to fetch "Authoratitve type dns
            records
46      def encode_msg(message): #to sort out the request from
            the client
47      return packed_data #encode the request in dns format
48
49      def handle_client(data, addr, server): #this function
            handles the threading for sending to the client the
            dns records
```

Listing 10: For TLD Server

```
50      def getDnsRecords():  # to fetch NX or MS dns records
51      def encode_msg(message): #to sort out the request from
            the client
52      return packed_data #encode the request in dns format
53
54      def decode_msg(msg): #decode the bit request from the
            client
55      return ms[1],ms[4]
56
57      def handle_client(data, addr, server): #this function
            handles the threading for sending to the client the
            dns records
```

Listing 11: For Root Server

```
58      def getDnsRecords():  # to fetch any dns records except
            the authoratitive dns
59      def encode_msg(message): #to sort out the request from
            the client
60      return packed_data #encode the request in dns format
61
62      def decode_msg(msg): #decode the bit request from the
            client
```

```
63      return ms[1],ms[4]
64
65      def handle_client(data, addr, server): #this function
            handles the threading for sending to the client the
            dns records
```

Listing 12: For Client Side

```
66      #this functions helps the clients to request for dns
            records from Rootserver or TLD or Authorative server
67      def encode_msg(message):
```

```
68      #this functions helps the clients to decode message sent
            from the server
69      def decode_msg(msg):
```

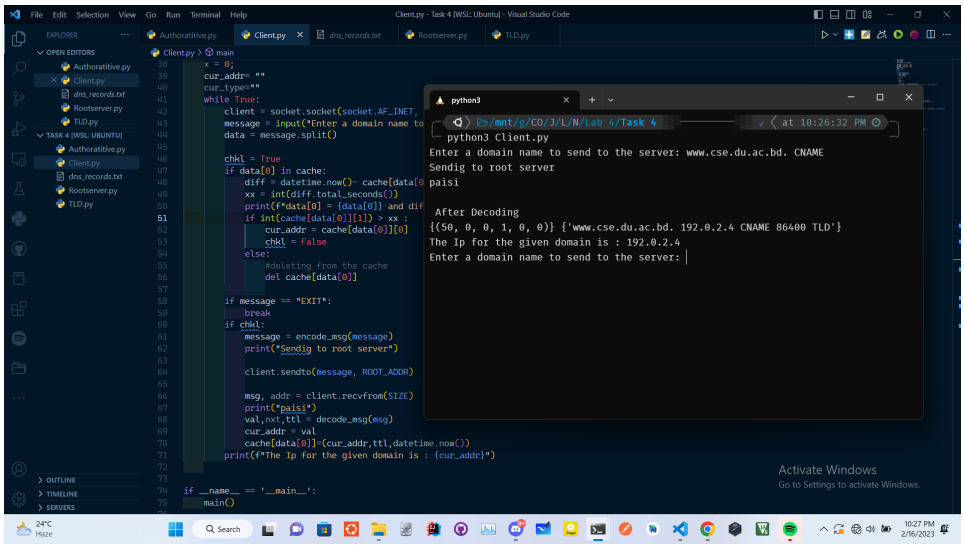

Figure 11: Client Side View

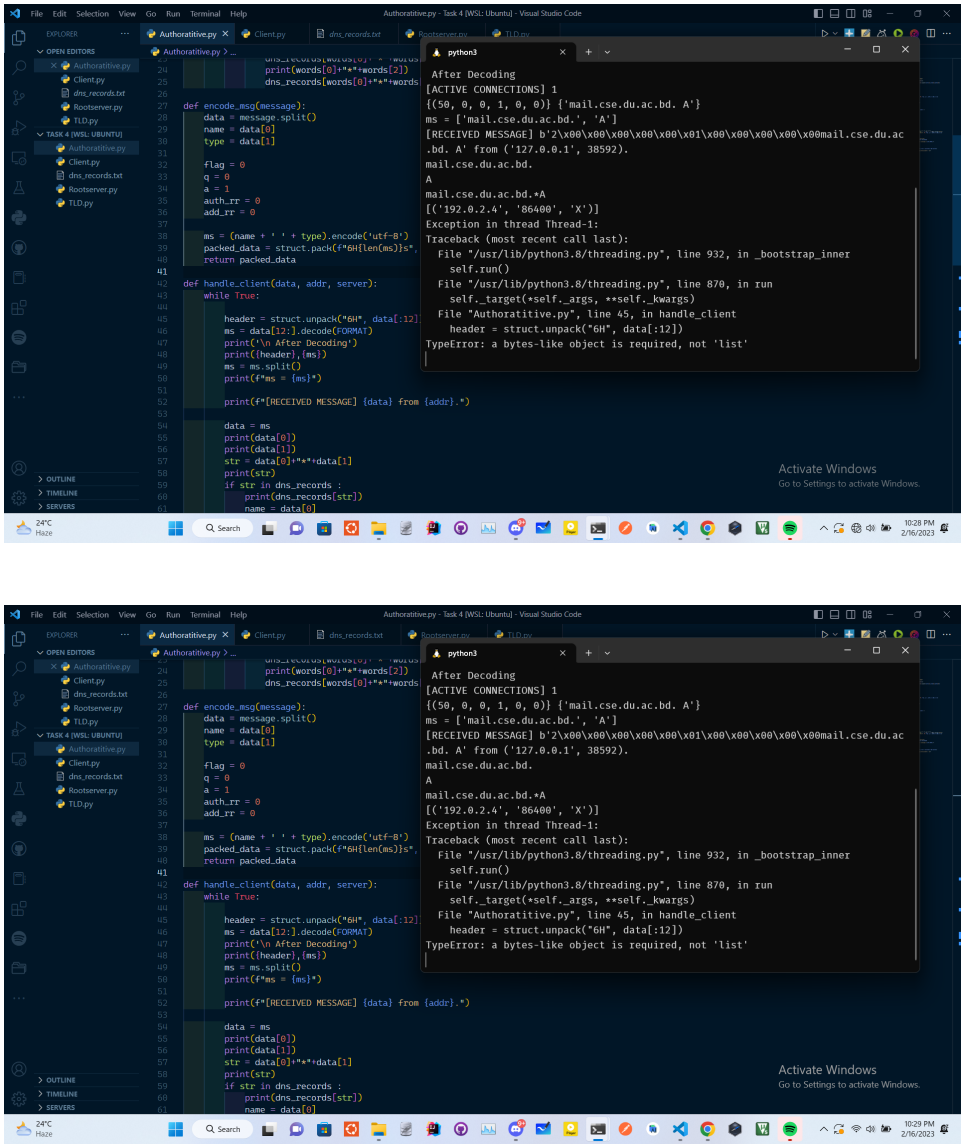

Figure 12: Authoratitive Server Side View

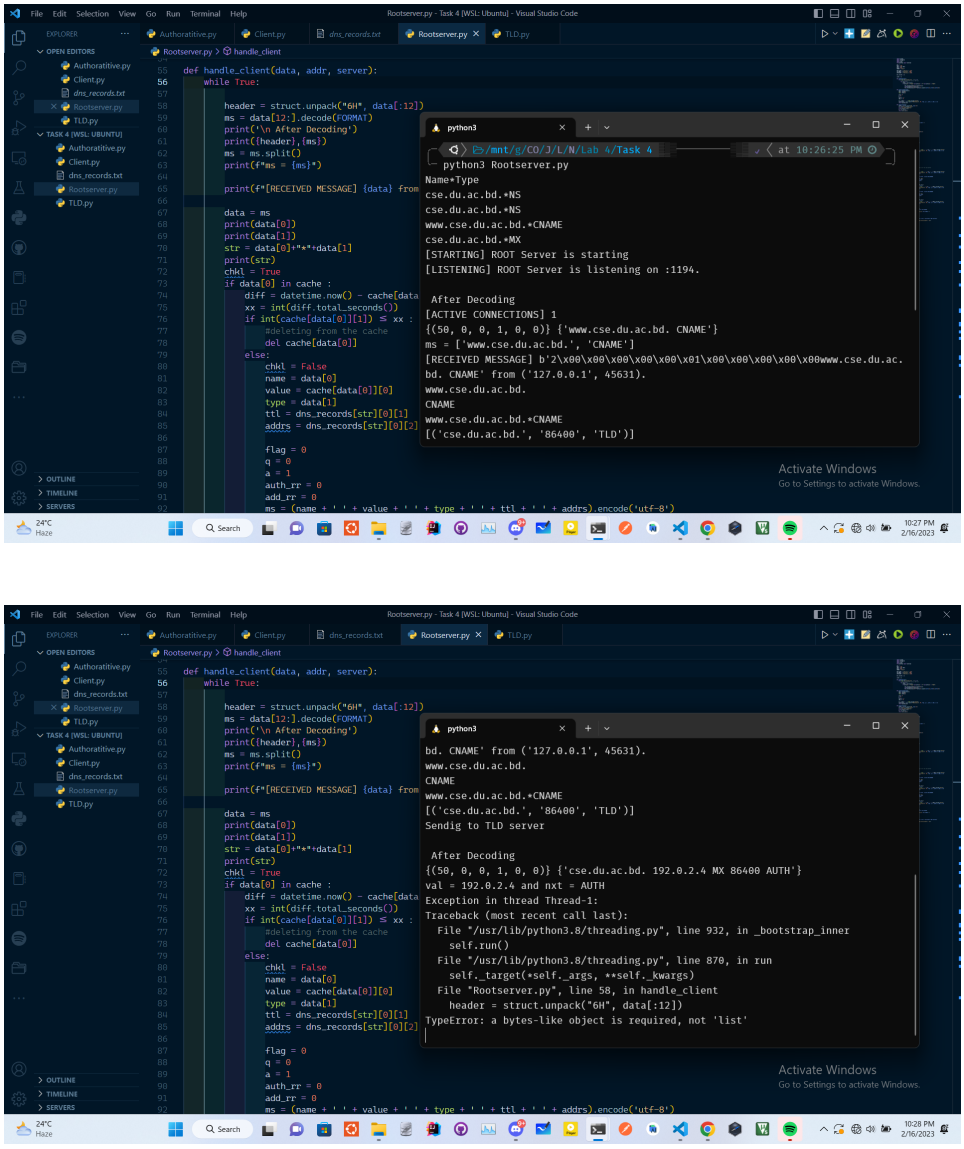

Figure 13: Root Server Side View

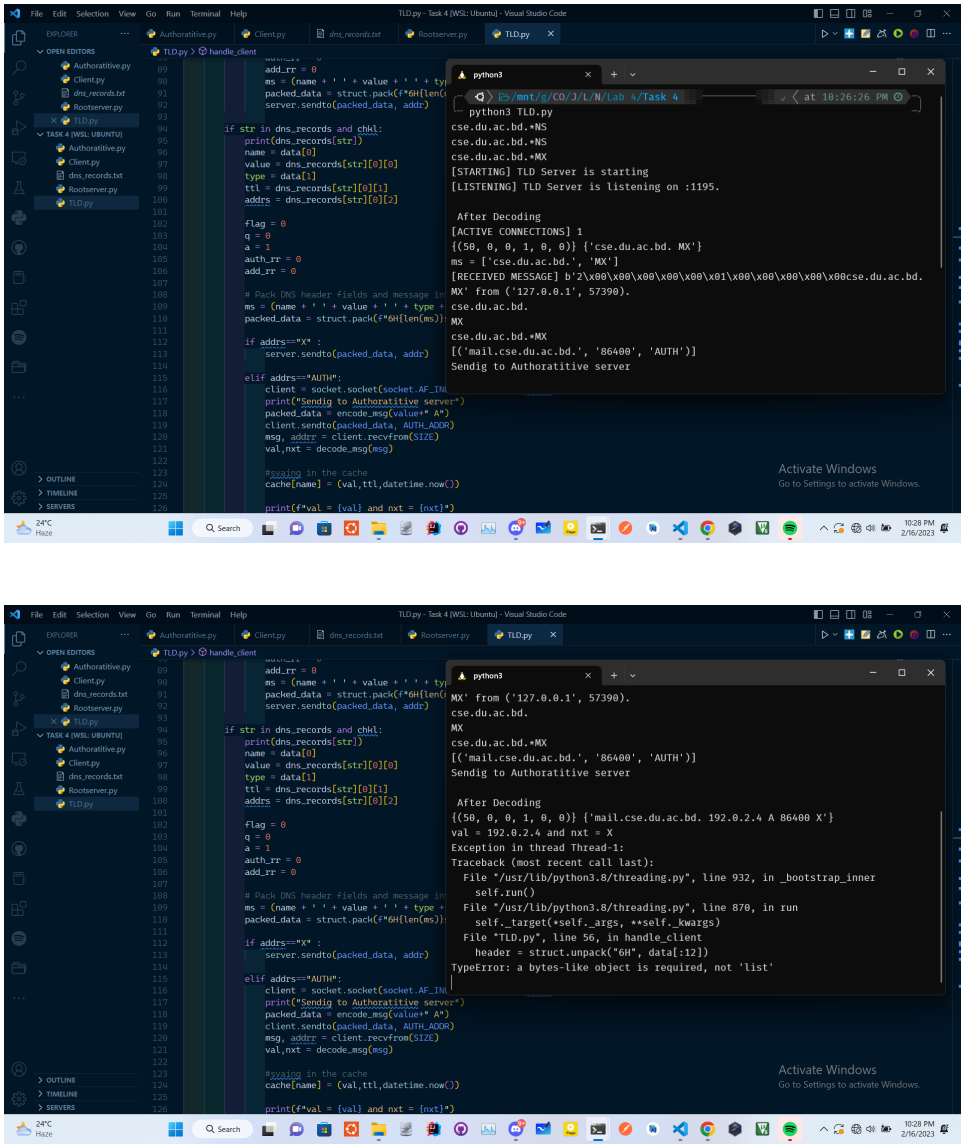

Figure 14: TLD Server Side View

8 Time Requirement

Recursive DNS acts faster in resolutions in average cases since it caches the final answer everytime for the TTL period. If the recursive DNS resolution is made to not memoize the answer, the time required for the both methode is almost same.

9 Experience

1. We had to evaluate the scalability of the distributed database by gradually increasing the number of nodes and measuring the impact on response time, accuracy, and resource usage.

2. We had to compare the performance of the distributed database with other types of database systems.

3. We had to evaluate the data consistency mechanism of the distributed database by intentionally introducing inconsistencies into the database.

4. We had performed security testing on the distributed database, including penetration testing, vulnerability scanning, and encryption testing, to evaluate the effectiveness of the security mechanisms implemented in the system.

References

[1] What is DNS? – Introduction to DNS - AWS. https://aws.amazon.com/route53/what-is-dns/. [Online; accessed 2023-02-13].

[2] Dns message format. *GeeksforGeeks*, may 18 2022. [Online; accessed 2023-02-16].

[3] Staff Writer. What's the difference between ANAME, AAAA, A, and CNAME records? https://constellix.com/news/dns-record-types, sep 27 2021. [Online; accessed 2023-02-13].