



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 4 : Distributed Database System Management and Implementation of
Iterative and Recursive DNS

Submitted By:

Name: Muztoba Hasan Sinha

Roll No : 15

Name: Bholanath Das Niloy

Roll No : 22

Submitted On :

February 16th, 2023

Submitted To :

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Objectives | 2 |
| 2 | Theory | 2 |
| 2.1 | DNS IP Retrieval Process | 2 |
| 2.2 | Iterative DNS resolution | 2 |
| 2.3 | Recursive DNS resolution | 3 |
| 2.4 | Advantages of recursive DNS over iterative | 3 |
| 2.5 | Disadvantages of Recursive DNS over Iterative | 3 |
| 3 | Code | 4 |
| 3.1 | Part 1 - Setting up the DNS server | 4 |
| 3.2 | Part 2 - Iterative DNS Resolution | 5 |
| 3.3 | Part 3 - Recursive DNS Resolution | 7 |
| 3.3.1 | Part 4 - Improvements | 9 |
| 4 | Explaining the Code | 10 |
| 4.1 | Part 1 - Setting up the DNS server | 10 |
| 4.2 | Part 2 - Iterative DNS Resolution | 11 |
| 4.3 | Part 3 - Recursive DNS Resolution | 11 |
| 4.4 | Part 4 - Improvements | 12 |
| 5 | Screenshots of Working Code | 13 |
| 5.1 | Setting Up DNS | 13 |
| 5.2 | Iterative DNS Resolving | 14 |
| 5.3 | Recursive DNS Resolving | 15 |
| 5.4 | Improvements | 16 |
| 5.5 | Results of comparison | 17 |

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

The objective is to gain a deep understanding of DNS and it's hierarchy.

- Learn the basics of DNS query systems and it's structure.
- Learn about iterative DNS.
- Learn about recursive DNS.
- Learn about failures of the DNS system.

2 Theory

All domains are converted from a human readable web address to a machine readable IP address. To make this lookup efficient, secure and fast the mapping is not stored in a centralized server or authority, rather, parts of the domain is resolved by a distributed hierarchy of nameservers. It follows a few steps to resolve the query

2.1 DNS IP Retrieval Process

1. *DNS cache* The first step to resolving a hostname is to look in the machines local DNS cache. If the name is not found here only then a DNS query is performed.
2. *ISP DNS server request* If the hostname is not resolved locally the machine queries the ISP DNS servers or local nameservers if configured.
3. *Root nameservers* If ISP DNS servers fail to resolve the request then they query the root nameservers. The root nameservers extract the toplevel domain and query the appropriate TLD server. These are generally hard coded into machines as there are only a few of them worldwide. These are coordinates and managed by the Internet Assigned Numbers Authority.
4. *Top Level Domain nameservers* The TLDs review the next part of the request and direct the query to nameservers specific to domains. For each Top Level Domain such as com, org, net, edu and all country TLDs there exist a cluster of TLD servers managed by Verisign Global Registry Services.
5. *Authoritative DNS servers* These are services which can specifically resolve domains to names. These authoritative nameservers are responsible for knowing all the information about a specific domain, which are stored in DNS records. They send the Authoritative reply that is the specific mapping from domain name to IP address.
6. *Record Retrieval* the ISP DNS server retrieves the record from authoritative nameservers and stores the record in it's local cache. 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 isn't out of date.
7. *Receiving the answer* ISP returns the record back to our machine and the record is stored in the DNS cache. The browser then uses the IP information to open a connection to the webserver and receive the website.

2.2 Iterative DNS resolution

In simple terms the client asks the resolver for the IP address for the domain or the address of the next DNS server in the lookup process so that the client can probe the next server itself. It is generally slower but is more secure. Basically if all of the above step sends a response back to the ISP nameserver or local machine which then, armed with the new address makes another query is called iterative DNS resolution.

2.3 Recursive DNS resolution

Here the client delegates the duty of finding the mapping to the resolver. That is, the resolver itself looks into the next server in the process and so on until the mapping is found. It is faster and saves network resources but is insecure. The nameservers in the process described in the retrieval process section queries the next server aggregates the result and sends back the response to the client in the recursive resolution process.

2.4 Advantages of recursive DNS over iterative

Recursive DNS are generally faster than iterative queries. Because all the servers in the recursion stack can cache the mapping so that the next DNS resolution is faster.

2.5 Disadvantages of Recursive DNS over Iterative

Allowing DNS queries on open servers create the scope for security vulnerabilities such as DNS amplification attacks and DNS cache poisoning.

1. *DNS amplification attacks* This is a variant of DDoS attack, The attackers use a botnet to send a high volume of DNS queries using a spoofed IP address. Thus sending a large amount of requests from their own IP to send a large amount of responses to the victim. These lengthy and torrential responses disrupt or take down the clients servers. The amplified DNS packets are reponses to recursive DNS queries thus it is only possible for a recursive DNS queries.
2. *DNS cache poisoning attacks* This is when an attacker intercepts a DNS request and gives a fake response, often links to a malicious website. This is significantly more harmful since this malicious response is saved to cache and can affect a large number of people. This is minimized in an iterative query because even if the response is poisoned it will affect only one user or ISP nameserver.

3 Code

3.1 Part 1 - Setting up the DNS server

1. Server Side

```
1 import socket
2 import dns.message
3
4 mpA = {
5     "cse.du.ac.bd." : "192.0.2.3",
6     "ns1.cse.du.ac.bd." : "192.0.2.1",
7     "ns2.cse.du.ac.bd." : "192.0.2.2",
8     "mail.cse.du.ac.bd." : "192.0.2.4"
9 }
10
11 mpNS = {
12     "cse.du.ac.bd." : "ns1.cse.du.ac.bd.",
13 }
14
15 mpAAAA = {
16     "cse.du.ac.bd." : "2001:db8::3",
17     "ns1.cse.du.ac.bd." : "2001:db8::1",
18     "ns2.cse.du.ac.bd." : "2001:db8::2",
19     "mail.cse.du.ac.bd." : "2001:db8::4"
20 }
21
22 mpCNAME = {
23     "www.cse.du.ac.bd." : "cse.du.ac.bd."
24 }
25
26 mpMX = {
27     "cse.du.ac.bd." : "10 mail.cse.du.ac.bd."
28 }
29
30 def handle_request(data, client_address):
31     request = dns.message.from_wire(data)
32
33     response = dns.message.make_response(request)
34     for item in request.question:
35         domain_name = item.to_text().split(" IN ")
36         if qtype == "NS":
37             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
38 mpNS[domain_name]);
39             response.answer.append(rrset)
40         elif qtype == "AAAA":
41             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
42 mpAAAA[domain_name]);
43             response.answer.append(rrset)
44         elif qtype == "CNAME":
45             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
46 mpCNAME[domain_name]);
47             response.answer.append(rrset)
48         elif qtype == "MX":
49             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
50 mpMX[domain_name]);
51             response.answer.append(rrset)
52         else:
53             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
54 mpA[domain_name]);
55             response.answer.append(rrset)
56             print(f"DEBUG: Query for {domain_name} with type {qtype}")
57
58     return response.to_wire()
59
60 def run_server():
61     server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
62     server_socket.bind(("127.0.0.1", 5050))
63     print("Server starting on 127.0.0.1 on port 5050")
64     while True:
```

```

60     data, client_address = server_socket.recvfrom(4096)
61     response = handle_request(data, client_address)
62     server_socket.sendto(response, client_address)
63
64 run_server()

```

Listing 1: Server Side Code

2. Client Side

```

1  import dns.message
2  import dns.query
3  import dns.resolver
4  import dns.rdatatype
5
6  def dns_resolve(domain, qtype):
7      query = dns.message.make_query(domain, qtype)
8      response = None
9
10     server = "127.0.0.1"
11
12     print(f"DEBUG: Querying {server} for {domain} ({qtype})\n\n")
13     response = dns.query.udp(query, server, port=5050)
14     print(f"DEBUG: Got response: {response}\n\n")
15
16     return response
17
18 print(dns_resolve("cse.du.ac.bd.", dns.rdatatype.A))

```

Listing 2: Server Side Code

3.2 Part 2 - Iterative DNS Resolution

1. Server Side

```

1  import socket
2  import dns.message
3
4  mpA = {
5      "cse.du.ac.bd." : "192.0.2.3",
6      "ns1.cse.du.ac.bd." : "192.0.2.1",
7      "ns2.cse.du.ac.bd." : "192.0.2.2",
8      "mail.cse.du.ac.bd." : "192.0.2.4"
9  }
10
11  mpNS = {
12      "cse.du.ac.bd." : "ns1.cse.du.ac.bd.",
13  }
14
15  mpAAAA = {
16      "cse.du.ac.bd." : "2001:db8::3",
17      "ns1.cse.du.ac.bd." : "2001:db8::1",
18      "ns2.cse.du.ac.bd." : "2001:db8::2",
19      "mail.cse.du.ac.bd." : "2001:db8::4"
20  }
21
22  mpCNAME = {
23      "www.cse.du.ac.bd." : "cse.du.ac.bd."
24  }
25
26  mpMX = {
27      "cse.du.ac.bd." : "10 mail.cse.du.ac.bd."
28  }
29
30  def handle_request(data, client_address):
31      request = dns.message.from_wire(data)
32
33      response = dns.message.make_response(request)
34      for item in request.question:
35          domain_name, qtype = item.to_text().split(" IN ")

```

```

36         if qtype == "NS":
37             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpNS[domain_name])
38             response.authority.append(rrset)
39         elif qtype == "AAAA":
40             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpAAAA[domain_name])
41             response.answer.append(rrset)
42         elif qtype == "CNAME":
43             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpCNAME[domain_name])
44             response.authority.append(rrset)
45         elif qtype == "MX":
46             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpMX[domain_name])
47             response.answer.append(rrset)
48         else:
49             rrset = dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpA[domain_name])
50             response.answer.append(rrset)
51         print(f"DEBUG: Query for {domain_name} with type {qtype}")
52
53     return response.to_wire()
54
55 def run_server():
56     server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
57     server_socket.bind(("127.0.0.1", 5050))
58     print("Server starting on 127.0.0.1 on port 5050")
59     while True:
60         data, client_address = server_socket.recvfrom(4096)
61         response = handle_request(data, client_address)
62         server_socket.sendto(response, client_address)
63
64 run_server()

```

Listing 3: Server Side Code

2. Client Side

```

1 import dns.message
2 import dns.query
3 import dns.resolver
4 import dns.rdatatype
5 import re
6
7 def check_dns_record_type(record):
8     a_regex = r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"
9     aaaa_regex = r"[0-9a-fA-F]{1,4}(:[0-9a-fA-F]{1,4}){7}"
10    ns_regex = r"[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"
11    cname_regex = r"[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"
12    mx_regex = r"\d+ [a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"
13
14    if re.match(a_regex, record):
15        return "A"
16    elif re.match(aaaa_regex, record):
17        return "AAAA"
18    elif re.match(ns_regex, record):
19        return "NS"
20    elif re.match(cname_regex, record):
21        return "CNAME"
22    elif re.match(mx_regex, record):
23        return "MX"
24    else:
25        return None
26
27 def iterative_resolve(domain, qtype):
28     authoritative_servers = []
29     query = dns.message.make_query(domain, qtype)
30     response = None
31     server = "127.0.0.1"

```

```

32
33     while True:
34         if authoritative_servers:
35             domain = authoritative_servers.pop()
36             qtype = check_dns_record_type(domain)
37             if domain == "cse.du.ac.bd.":
38                 qtype = dns.rdatatype.NS
39             elif qtype == "NS":
40                 qtype = "A"
41             query = dns.message.make_query(domain, qtype)
42             print(f"DEBUG: Querying {server} for {domain} ({qtype})\n\n")
43         else:
44             print(f"DEBUG: Querying {server} for {domain} ({qtype})\n\n")
45             response = dns.query.udp(query, server, port=5050)
46             print(f"DEBUG: Got response: {response}\n\n")
47
48             if response.rcode() == dns.rcode.NOERROR:
49                 if len(response.authority) > 0:
50                     for rrset in response.authority:
51                         if rrset.rdtype == dns.rdatatype.NS or rrset.rdtype == dns.
rdatatype.CNAME:
52                             for rdata in rrset:
53                                 authoritative_servers.append(str(rdata))
54                             print(f"DEBUG: Adding authoritative servers: {authoritative_servers}\n\
n")
55                         else:
56                             break
57             else:
58                 print(f"Error: {response.rcode()}\n\n")
59
60             if not authoritative_servers:
61                 break
62
63         return response
64
65 print(iterative_resolve("cse.du.ac.bd.", dns.rdatatype.NS))

```

Listing 4: Server Side Code

3.3 Part 3 - Recursive DNS Resolution

1. Server Side

```

1 # import re
2 import socket
3 import dns.message
4
5 mpA = {
6     "cse.du.ac.bd." : "192.0.2.3",
7     "ns1.cse.du.ac.bd." : "192.0.2.1",
8     "ns2.cse.du.ac.bd." : "192.0.2.2",
9     "mail.cse.du.ac.bd." : "192.0.2.4"
10 }
11
12 mpNS = {
13     "cse.du.ac.bd." : "ns1.cse.du.ac.bd."
14 }
15
16 mpAAAA = {
17     "cse.du.ac.bd." : "2001:db8::3",
18     "ns1.cse.du.ac.bd." : "2001:db8::1",
19     "ns2.cse.du.ac.bd." : "2001:db8::2",
20     "mail.cse.du.ac.bd." : "2001:db8::4"
21 }
22
23 mpCNAME = {
24     "www.cse.du.ac.bd." : "cse.du.ac.bd."
25 }
26
27 mpMX = {

```



```

28     "cse.du.ac.bd." : "10 mail.cse.du.ac.bd."
29 }
30
31 def recurse(domain_name, qtype):
32     print(f"DEBUG: Query for {domain_name} with type {qtype}")
33     ret = []
34     if qtype == "A":
35         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
36 mpA[domain_name]));
37         return ret
38     if qtype == "AAAA":
39         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
40 mpAAAA[domain_name]));
41         return ret
42     if qtype == "MX":
43         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
44 mpMX[domain_name]));
45         return ret
46     if qtype == "NS":
47         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
48 mpNS[domain_name]));
49         retrec = recurse(mpNS[domain_name], "A")
50         for item in retrec:
51             ret.append(item)
52         return ret
53     if qtype == "CNAME":
54         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
55 mpCNAME[domain_name]));
56         retrec = recurse(mpCNAME[domain_name], "NS")
57         for item in retrec:
58             ret.append(item)
59         return ret
60     return None
61
62 def handle_request(data, client_address):
63     request = dns.message.from_wire(data)
64
65     response = dns.message.make_response(request)
66     for item in request.question:
67         domain_name, qtype = item.to_text().split(" IN ")
68         print(domain_name + " " + qtype)
69         ret = recurse(domain_name, qtype)
70         for item in ret:
71             print(item)
72             response.answer.append(item)
73
74     return response.to_wire()
75
76 def run_server():
77     server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
78     server_socket.bind(("127.0.0.1", 5050))
79     print("Server starting on 127.0.0.1 on port 5050")
80     while True:
81         data, client_address = server_socket.recvfrom(4096)
82         response = handle_request(data, client_address)
83         server_socket.sendto(response, client_address)
84
85 run_server()

```

Listing 5: Server Side Code

2. Client Side

```

1 import dns.message
2 import dns.query
3 import dns.resolver
4 import dns.rdatatype
5
6 def recursive_resolve(domain, qtype):
7     query = dns.message.make_query(domain, qtype)

```

```

8     response = None
9
10    server = "127.0.0.1"
11
12    print(f"DEBUG: Querying {server} for {domain} ({qtype})\n\n")
13    response = dns.query.udp(query, server, port=5050)
14    print(f"DEBUG: Got response: {response}\n\n")
15
16    return response
17
18 print(recursive_resolve("www.cse.du.ac.bd.", dns.rdatatype.CNAME))

```

Listing 6: Server Side Code

3.3.1 Part 4 - Improvements

1. Server Side for Recursive

```

1 import socket
2 import dns.message
3
4 mpA = {
5     "cse.du.ac.bd." : "192.0.2.3",
6     "ns1.cse.du.ac.bd." : "192.0.2.1",
7     "ns2.cse.du.ac.bd." : "192.0.2.2",
8     "mail.cse.du.ac.bd." : "192.0.2.4"
9 }
10
11 mpNS = {
12     "cse.du.ac.bd." : "ns1.cse.du.ac.bd."
13 }
14
15 mpAAAA = {
16     "cse.du.ac.bd." : "2001:db8::3",
17     "ns1.cse.du.ac.bd." : "2001:db8::1",
18     "ns2.cse.du.ac.bd." : "2001:db8::2",
19     "mail.cse.du.ac.bd." : "2001:db8::4"
20 }
21
22 mpCNAME = {
23     "www.cse.du.ac.bd." : "cse.du.ac.bd."
24 }
25
26 mpMX = {
27     "cse.du.ac.bd." : "10 mail.cse.du.ac.bd."
28 }
29
30 mapCache = {}
31
32 def recurse(domain_name, qtype):
33     print(f"DEBUG: Query for {domain_name} with type {qtype}")
34     if [domain_name, qtype] in mapCache.keys():
35         return mapCache[[domain_name, qtype]]
36     ret = []
37     if qtype == "A":
38         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
39 mpA[domain_name]))
40         mapCache[[domain_name, qtype]] = ret
41         return ret
42     if qtype == "AAAA":
43         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
44 mpAAAA[domain_name]))
45         mapCache[[domain_name, qtype]] = ret
46         return ret
47     if qtype == "MX":
48         ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
49 mpMX[domain_name]))
50         return ret
51     if qtype == "NS":

```

```

49     ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpNS[domain_name]))
50     retrec = recurse(mpNS[domain_name], "A")
51     for item in retrec:
52         ret.append(item)
53     mapCache[[domain_name, qtype]] = ret
54     return ret
55 if qtype == "CNAME":
56     ret.append(dns.rrset.from_text(domain_name, 86400, dns.rdataclass.IN, qtype,
mpCNAME[domain_name]))
57     retrec = recurse(mpCNAME[domain_name], "NS")
58     for item in retrec:
59         ret.append(item)
60     mapCache[[domain_name, qtype]] = ret
61     return ret
62 return None
63
64 def handle_request(data, client_address):
65     request = dns.message.from_wire(data)
66
67     response = dns.message.make_response(request)
68     for item in request.question:
69         domain_name, qtype = item.to_text().split(" IN ")
70         print(domain_name + " " + qtype)
71         ret = recurse(domain_name, qtype)
72         for item in ret:
73             print(item)
74         response.answer.append(item)
75
76     return response.to_wire()
77
78 def run_server():
79     server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
80     server_socket.bind(("127.0.0.1", 5050))
81     print("Server starting on 127.0.0.1 on port 5050")
82     while True:
83         data, client_address = server_socket.recvfrom(4096)
84         response = handle_request(data, client_address)
85         server_socket.sendto(response, client_address)
86
87 run_server()

```

Listing 7: Server Side Code

4 Explaining the Code

4.1 Part 1 - Setting up the DNS server

1. Server Side

This code is a DNS server implementation that provides DNS resolution for a few domain names with the help of resource records stored in memory as dictionaries. When a DNS query is received, it constructs a DNS response using the resource records and sends the response back to the client. It listens on UDP port 5050 for incoming requests, and it can handle requests for different types of DNS records such as **A**, **AAAA**, **NS**, **MX**, and **CNAME**.

The **handle_request** function is responsible for processing the DNS request received from the client, and generating a DNS response.

It takes two parameters: **data** which represents the raw DNS request received over the socket and **client_address** which represents the address of the client that sent the request.

The function begins by decoding the DNS request from the binary data using the **dns.message.from_wire** method. It then creates a new DNS response using the **dns.message.make_response** method, which includes the header and question sections from the request.

For each question in the request, the function extracts the domain name and query type, and checks which type of record is being requested. If the query type is **NS**, **AAAA**, **CNAME** or **MX**, it constructs an answer containing the requested resource record(s) by looking up the values in the corresponding

map (**mpNS**, **mpAAAA**, **mpCNAME**, and **mpMX** respectively), and adds it to the response. If the query type is anything else, it constructs an answer using the corresponding value in the **mpA** map.

Finally, the function returns the response in binary format using the **response.to_wire()** method.

2. Client Side

The purpose of this code is to resolve a DNS query for a given domain name and record type (e.g. A record, MX record, etc.) using the DNS server running at IP address "127.0.0.1" and port number 5050.

The function **dns.resolve** constructs a DNS query message using the **make_query** function from the **dns.message** module, sends the query using the **dns.query.udp** function, and returns the response as a **dns.message.Message** object. The print statements are used for debugging purposes to display the query being sent and the response received from the DNS server.

Finally, the script calls the **dns.resolve** function with the domain name cse.du.ac.bd and record type A (IPv4 address), and prints the response.

4.2 Part 2 - Iterative DNS Resolution

1. Server Side

This is essentially the same thing as the code in normal DNS resolving, but with a key difference which is it adds the nameservers to the authoritative servers list instead of the answers list. What this enables us to do is on the client side iterate on this new nameserver and requery the server.

2. Client Side

The **iterative_resolve** function is used to resolve a DNS query iteratively by querying the authoritative DNS servers for the domain, starting with the root server and continuing down the hierarchy of authoritative servers until a final response is obtained. The function takes a domain name and query type as input and returns a **dns.message.Message** object containing the response from the authoritative server(s) for that domain and query type.

This function can be useful for applications that need to resolve DNS queries programmatically and want to implement a custom resolution strategy, such as load balancing requests across multiple authoritative servers or handling errors or timeouts more gracefully.

The function first creates an empty list called **authoritative_servers**. Then, it creates a DNS query using the **make_query()** function from the **dns.message** module.

It then enters a while loop which runs until it breaks out explicitly. It sets the server variable to the IP address of the DNS resolver it wants to use. If there are any items in the **authoritative_servers** list, it takes the first item and constructs a new query using that server for the domain A. The query variable is then updated with the new query.

The **dns.query.udp()** function is then called with the constructed query and server to get a response. The response variable is updated with the response from the query.

If the response code is **NOERROR**, the function checks if there are any authority records in the response. If there are authority records, it extracts the name server names from the response and appends them to the **authoritative_servers** list.

If there are no authority records, the function breaks out of the loop. If there are no more name servers left to query, the function breaks out of the loop.

The function returns the response, which is the result of iteratively resolving.

4.3 Part 3 - Recursive DNS Resolution

1. Server Side

This implements a DNS resolver server. The resolver server listens on a specific IP address and port ("127.0.0.1", 5050) for incoming DNS queries, processes them, and sends back a response.

The server uses the DNS Python library to parse incoming DNS queries and create DNS responses. It also includes a number of pre-defined DNS records for specific domain names and record types, stored in dictionaries (**mpA**, **mpNS**, **mpAAAA**, **mpCNAME**, and **mpMX**).

The **handle_request** function is responsible for processing an incoming DNS query message. It uses the **dns.message.from_wire** function to convert the raw message bytes into a DNS message object. It

then extracts the domain name and record type from the question section of the message, and calls the **recurse** function to perform the actual DNS resolution.

The **recurse** function is responsible for looking up a given domain name and record type in the pre-defined record dictionaries. If the record is found, it creates an **dns.rrset.RRset** object representing the DNS response, which is then added to the response message. If the record is not found, the function may recursively query for additional records to resolve the name.

Finally, the **run_server** function sets up a UDP socket to listen for incoming DNS queries, and calls **handle_request** to process each query and send back a response.

The purpose of this code is to demonstrate how to implement a basic DNS resolver server in Python using the DNS Python library. It is not intended to be used as a production-grade DNS resolver, as it only supports a limited set of record types and does not perform any caching or error handling.

2. Client Side

This is the same as the normal DNS Client Code.

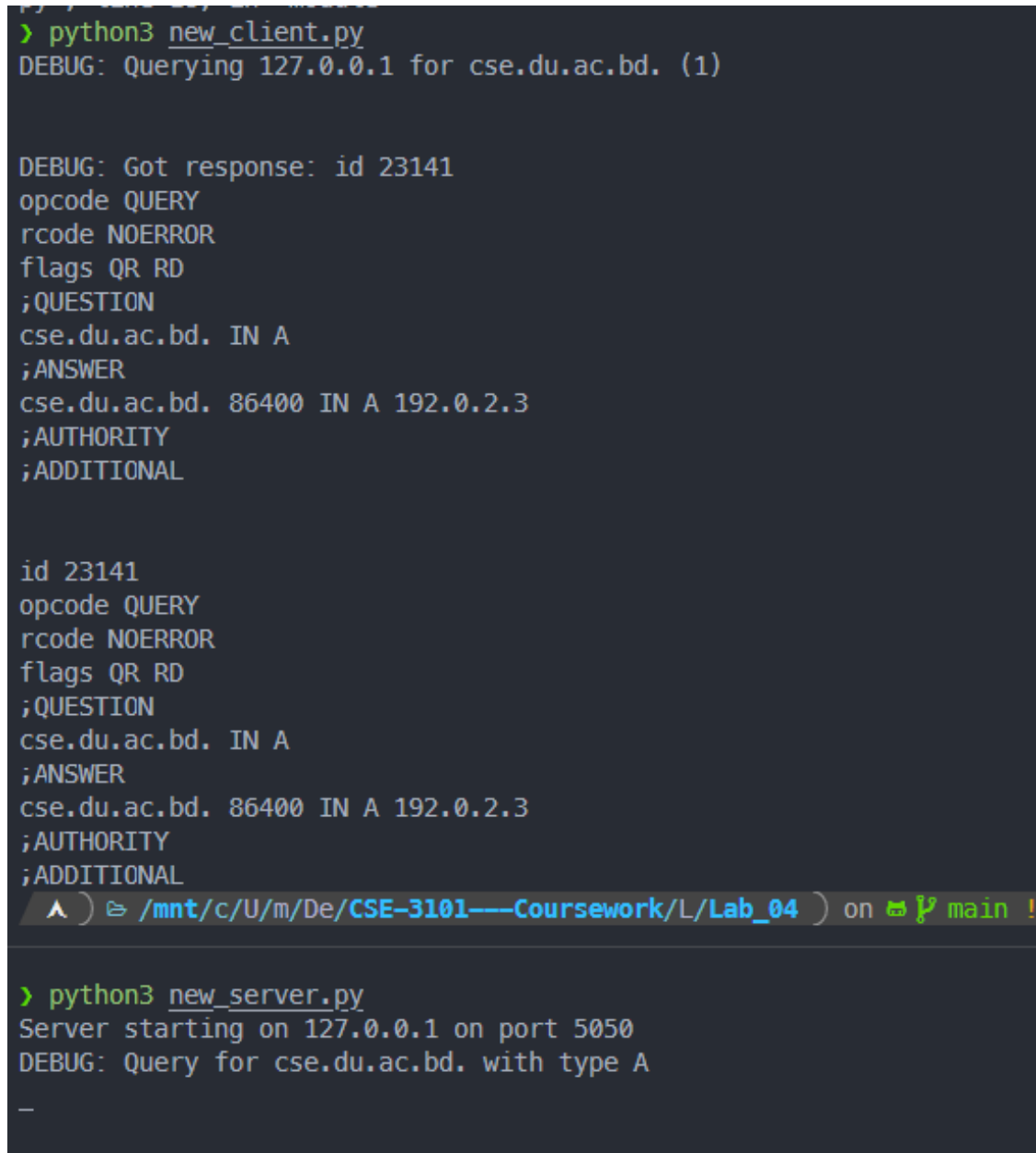
4.4 Part 4 - Improvements

What we have done is, on the server side, for recursive dns resolving we have cached the results and stored them in a map so that when once something is called once it is never called again.

5 Screenshots of Working Code

5.1 Setting Up DNS

Here we are querying for **cse.du.ac.bd** with a **A** type query and are getting back the result as desired:



```
> python3 new_client.py
DEBUG: Querying 127.0.0.1 for cse.du.ac.bd. (1)

DEBUG: Got response: id 23141
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
cse.du.ac.bd. IN A
;ANSWER
cse.du.ac.bd. 86400 IN A 192.0.2.3
;AUTHORITY
;ADDITIONAL

id 23141
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
cse.du.ac.bd. IN A
;ANSWER
cse.du.ac.bd. 86400 IN A 192.0.2.3
;AUTHORITY
;ADDITIONAL
^ ) /mnt/c/U/m/De/CSE-3101---Coursework/L/Lab_04 ) on 🐍 main !

> python3 new_server.py
Server starting on 127.0.0.1 on port 5050
DEBUG: Query for cse.du.ac.bd. with type A
—
```

Figure 1: DNS Query

5.2 Iterative DNS Resolving

Here the client is making multiple queries to servers to get from the Root nameserver to where we need to be:

```
> python3 iter_server.py
Server starting on 127.0.0.1 on port 5050
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
█

ns1.cse.du.ac.bd. IN A
;ANSWER
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
;AUTHORITY
;ADDITIONAL
time: 0.0032541751861572266
> python3 iter_client.py
DEBUG: Querying 127.0.0.1 for www.cse.du.ac.bd. (5)

DEBUG: Got response: id 39544
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
www.cse.du.ac.bd. IN CNAME
;ANSWER
;AUTHORITY
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
;ADDITIONAL

DEBUG: Adding authoritative servers: ['cse.du.ac.bd.']
```

Figure 2: Iterative DNS resolving

5.3 Recursive DNS Resolving

Here the server does the recursive queries to figure out what the client wanted and returns it:

```
> python3 recur_server.py
Server starting on 127.0.0.1 on port 5050
www.cse.du.ac.bd. CNAME
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
www.cse.du.ac.bd. CNAME
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.

;ADDITIONAL
time: 0.0019724369049072266
> python3 recur_client.py
id 12499
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
www.cse.du.ac.bd. IN CNAME
;ANSWER
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
;AUTHORITY
;ADDITIONAL
time: 0.0020377635955810547
> python3 recur_client.py
id 51791
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
www.cse.du.ac.bd. IN CNAME
;ANSWER
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
;AUTHORITY
;ADDITIONAL
time: 0.0023345947265625
```

Figure 3: Recursive DNS Resolving

5.4 Improvements

Here the recursive server is caching the results, i.e. if something has been queried once there is no point in recalculating whatever we needed just looking up the values is enough:

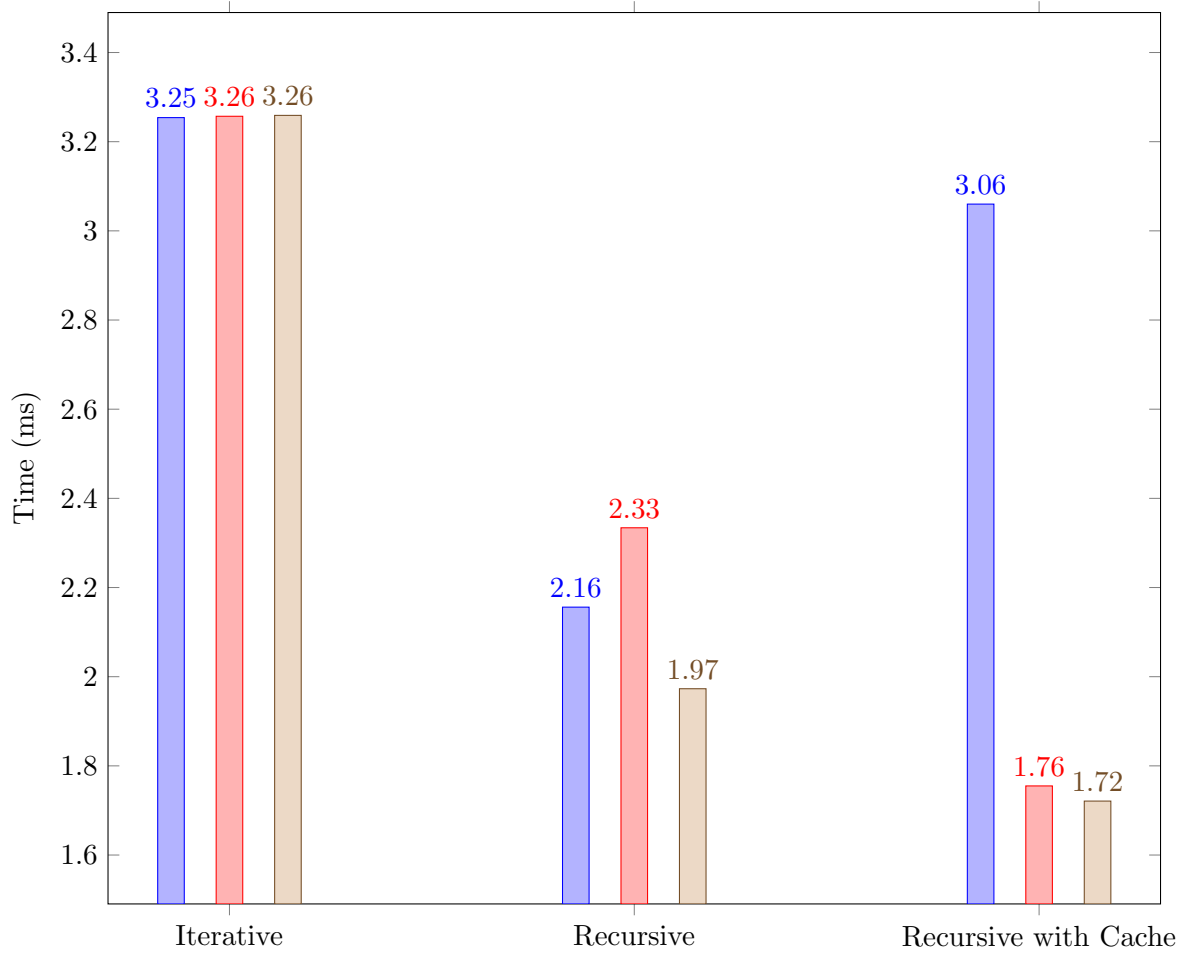
```
> python3 recur_server.py
Server starting on 127.0.0.1 on port 5050
www.cse.du.ac.bd. CNAME
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
DEBUG: Query for cse.du.ac.bd. with type NS
DEBUG: Query for ns1.cse.du.ac.bd. with type A
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
www.cse.du.ac.bd. CNAME
DEBUG: Query for www.cse.du.ac.bd. with type CNAME
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
█
```

```
> python3 recur_client.py
id 55977
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
www.cse.du.ac.bd. IN CNAME
;ANSWER
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
;AUTHORITY
;ADDITIONAL
time: 0.0030603408813476562
> python3 recur_client.py
id 1428
opcode QUERY
rcode NOERROR
flags QR RD
;QUESTION
www.cse.du.ac.bd. IN CNAME
;ANSWER
www.cse.du.ac.bd. 86400 IN CNAME cse.du.ac.bd.
cse.du.ac.bd. 86400 IN NS ns1.cse.du.ac.bd.
ns1.cse.du.ac.bd. 86400 IN A 192.0.2.1
;AUTHORITY
;ADDITIONAL
time: 0.0017287731170654297
```

```
█ █ ~/l/pyth python3 recur_client.py █
```

Figure 4: Recursive DNS Resolving with caching

5.5 Results of comparison



As can be seen iterative here is the slowest of all the methods we have implemented, and recursive is faster. This may be different for different workloads but for ours this is what we have got. In the recursive with caching we can see that initially the time taken is very high after which when the result has been cached the time required decreases significantly.

References

- [1] dnspython. <https://pypi.org/project/dnspython/>. [Online; accessed 2023-02-16].
- [2] dnspython. <https://www.dnspython.org/>. [Online; accessed 2023-02-16].
- [3] Recursive and iterative DNS queries. <https://www.omnisecu.com/tcpip/recursive-and-iterative-dns-queries.php>. [Online; accessed 2023-02-16].
- [4] Recursive DNS queries. <https://www.cloudflare.com/learning/dns/what-is-recursive-dns/:text=A> [Online; accessed 2023-02-16].
- [5] P. Mockapetris. Rfc 1035: Domain names. <https://www.rfc-editor.org/rfc/rfc1035>. [Online; accessed 2023-02-16].
- [6] J. Postel. Rfc 768: user datagram protocol. <https://www.rfc-editor.org/rfc/rfc768>. [Online; accessed 2023-02-16].