



UNIVERSITY OF DHAKA

CSE:3111 COMPUTER NETWORKING LAB

Lab Report 4

**Distributed Database Management,
Implementation of Iterative, and Recursive
Queries of DNS Records.**

Shariful Islam Rayhan
Roll: 41
Md Sakib Ur Rahman
Roll: 37

Submitted to:

Dr. Md. Abdur Razzaque

Dr. Muhammad Ibrahim

Dr. Md. Redwan Ahmed Rizvee

Dr. Md. Mamun Or Rashid

Submitted on: 2024-02-15

1 Introduction

In the vast landscape of the internet, every website or webpage possesses a unique identity address, known as an IP address. However, these numerical identities are cumbersome to remember. Instead, we rely on Uniform Resource Locators (URLs), which are more human-friendly and easier to recall. Behind the scenes, the Domain Name System (DNS) serves as the backbone, seamlessly translating these user-friendly URLs into corresponding IP addresses. This pivotal role of the DNS system enables our browsers to effortlessly connect to websites and webpages, enhancing the accessibility and usability of the internet.

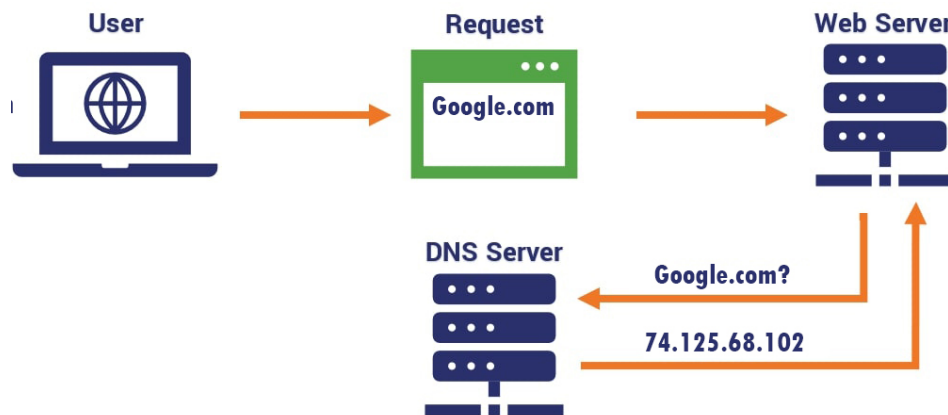


Figure 1: DNS system to convert a url to ip

1.1 Objectives

- Emulate the Domain Name Service (DNS) protocol within a controlled laboratory environment.
- Investigate the differences between iterative and recursive DNS resolution methods. clients to request the IP address of desired domains.
- Utilize the nameserver hierarchy to facilitate DNS resolution processes.
- Verify the validity of domain names and return corresponding IP addresses to clients.
- Gain practical insights into DNS operations, including resolution and translation mechanisms.
- Enhance understanding of distributed database management principles in networked environments through DNS emulation.

2 Theory

In the journey of accessing a domain such as google.com, our computer undergoes a meticulously orchestrated process to translate the user-friendly web address into a machine-readable IP address. Initially, our computer consults its local DNS cache, seeking a swift resolution of the hostname. Should the necessary information be absent from the cache, a DNS query is initiated, reaching out to our Internet Service Provider's (ISP) DNS servers for assistance. In the event of a cache miss at the ISP level, the query progresses to the root nameservers, which serve as the foundational layer of the DNS hierarchy. These nameservers meticulously direct the query to the appropriate Top-Level Domain (TLD) nameservers, such as .com, .org, or .bd, based on the domain extension. Subsequently, the TLD nameservers, acting as authoritative gatekeepers, route the query to the specific authoritative DNS servers responsible for the queried domain. These authoritative servers harbor comprehensive information about the domain, stored within DNS records. Upon retrieval of the requisite record from the authoritative nameservers, the ISP's DNS server caches this information locally, facilitating expedited resolutions for subsequent requests. Ultimately, armed with the IP address gleaned from the DNS resolution process, our computer establishes a connection with the webserver, enabling the retrieval of the desired website content through the browser.

3 Methodology

3.1 Task 1:Setting up the DNS server

In this step,we just make the authorivite name server that basically provide the ip for first time if there use cache and a client.

3.1.1 Server

```
1 import socket
2 import threading
3 import os
4 import struct
5
6
7 def load_file(filename):
8     dns_record = {}
9     with open(filename, "r") as file:
10         for line in file:
11             name, value, r_type, ttl = line.strip().split()
12             name = name.lower() # Convert domain name to
lowercase
13             print(name)
14             if name not in dns_record:
15                 dns_record[name] = [] # Initialize list if
domain name doesn't exist
16                 dns_record[name].append((value, r_type, int(ttl)))
17     return dns_record
18
19
20 def handle_dns_query_iterative(msg, c_addr, server):
21     flag="Auth server is working"
22     server.sendto(flag.encode(),c_addr)
23     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
24
25     domain_name = msg.decode().lower()
26     my_string =domain_name
27     modified_string = my_string[:-3]
28     print(modified_string)
29     domain_name=modified_string
30
31
32     # Decode the message from bytes to string
33     print("Lowercased domain:", domain_name)
34     if domain_name in dns_record:
35         records = dns_record[domain_name][-1] # Get the last
record for the domain
```

```

36         # Construct the response by joining the last record's
           components
37
38         result = domain_name+"com"+" " + " ".join(map(str,
           records))
39     else:
40         result = "Not found. You have to register this domain
           name."
41
42     server.sendto(result.encode(), c_addr) # Send the response
           to the client
43
44
45 def main():
46     server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
47
48     global dns_record
49     dns_record = load_file("dns_records.txt")
50     server.bind(('10.33.2.203', 9997))
51
52     print("dns server is running")
53
54     while True:
55         msg, c_addr = server.recvfrom(1024)
56         th = threading.Thread(target=handle_dns_query_iterative
           , args=(msg, c_addr, server))
57         th.start()
58
59
60 if __name__ == '__main__':
61     main()

```

3.1.2 Client

```
1 import socket
2 import time
3
4
5 def main():
6     client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7     server_addr = (("10.33.2.204", 9993))
8
9     for i in range(0,3):
10         t1 = time.time()
11         domain = "cse.du.ac.bd.com"
12         client.sendto(domain.encode(), server_addr)
13
14         msg, _ = client.recvfrom(1024)
15         print(msg.decode())
16
17         msg, _ = client.recvfrom(1024)
18         print(msg.decode())
19         t2 = time.time()
20         print(t2 - t1)
21
22
23 if __name__ == "__main__":
24     main()
```

3.1.3 Experimental Result for task 1

```
student@student-Veriton-N4660G:~/Downloads/Task3$ python3 authserver.py
cse.du.ac.bd.
cse.du.ac.bd.
ns1.cse.du.ac.bd.
ns2.cse.du.ac.bd.
ns1.cse.du.ac.bd.
ns2.cse.du.ac.bd.
cse.du.ac.bd.
cse.du.ac.bd.
www.cse.du.ac.bd.
cse.du.ac.bd.
mail.cse.du.ac.bd.
mail.cse.du.ac.bd.
dns server is running
Connected to ('10.33.2.204', 41303), and the client is querying for b'cse.du.ac.
bd.com'
cse.du.ac.bd.
Lowercased domain: cse.du.ac.bd.
Connected to ('10.33.2.204', 40237), and the client is querying for b'cse.du.ac.

student@student-Veriton-N4660G:~/Downloads/Task3$ python3 client.py
cse.du.ac.bd.com mail.cse.du.ac.bd. MX 86400
0.003762483596801758
student@student-Veriton-N4660G:~/Downloads/Task3$ python3 clientdns.py
cse.du.ac.bd.com mail.cse.du.ac.bd. MX 86400
0.0034596920013427734
cse.du.ac.bd.com mail.cse.du.ac.bd. MX 86400
0.002398252487182617
cse.du.ac.bd.com mail.cse.du.ac.bd. MX 86400
0.002315044403076172
student@student-Veriton-N4660G:~/Downloads/Task3$
```

Figure 2: Authoritive Dns and client is running

3.2 Task 2: Iterative DNS resolution

To implement this iterative approach, we make 5 files. They are client, local dns server, root server, tld server and authoritative server. Every time we use the same authoritative server and client server, so, we don't mention it in future.

3.2.1 Local DNS Server

```
1 import socket
2 import threading
3 import os
4 import struct
5
6 def load_file(filename):
7     dns_record = {}
8     with open(filename, "r") as file:
9         for line in file:
```



```

10         name, value, r_type, ttl = line.strip().split()
11         name = name.lower() # Convert domain name to
lowercase
12         # print(name)
13         if name not in dns_record:
14             dns_record[name] = [] # Initialize list if
domain name doesn't exist
15             dns_record[name].append((value, r_type, int(ttl)))
16         return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20
21
22
23
24     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
25
26     print("locan dns cannot find the ip..so it start the
request for root")
27
28
29     #connected with root
30     # kisu=input("press enter for access root")
31     client=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
32     server_addr=(("10.33.2.203",9195))
33
34     client.sendto(msg,server_addr)
35
36     # flag,_=client.recvfrom(1024)
37     # print(flag.decode())
38
39
40     msg1,_=client.recvfrom(1024)
41
42     address=msg1.decode()
43     address=address.split()
44     ip=address[0]
45     port=int(address[1])
46     #connected to tld
47
48     client_tld=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
49     server_addr_tld=((ip,port))
50
51     client_tld.sendto(msg,server_addr_tld)
52
53     msg2,_=client_tld.recvfrom(1024)
54

```

```

55
56
57
58     #connect to auth
59     msg2=msg2.decode()
60
61     print(f"ip of auth {msg2}")
62
63     msg2=msg2.split()
64     ip1=msg2[0]
65
66     port1=int(msg2[1])
67
68     client_auth=socket.socket(socket.AF_INET, socket.SOCK_DGRAM
69 )
70     server_addr_auth=((ip1,port1))
71
72     client_auth.sendto(msg,server_addr_auth)
73
74     msg3,_=client_auth.recvfrom(1024)
75
76
77
78
79
80
81     server.sendto(msg3, c_addr)
82
83
84
85
86
87
88
89
90     # domain_name = msg.decode().lower() # Decode the message
91     from bytes to string
92     # print("Lowercased domain:", domain_name)
93     # if domain_name in dns_record:
94     #     records = dns_record[domain_name][-1] # Get the last
95     record for the domain
96     # # Construct the response by joining the last record's
97     components
98
99     #     result = domain_name+" ".join(map(str, records))
100    # else:
101    #     result = "Not found. You have to register this domain
102    name."

```

```

99
100     # Send the response to the client
101
102
103 def main():
104     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
105
106     global dns_record
107     dns_record=load_file("dns_records.txt")
108     server.bind(('10.33.2.204',9993))
109
110     print("local_dns server is running")
111
112     while True:
113         msg,c_addr =server.recvfrom(1024)
114
115         th=threading.Thread(target=handle_dns_query_iterative,
116                             args=(msg,c_addr,server))
117         th.start()
118
119
120 if __name__=='__main__':
121     main()

```

3.2.2 Root DNS Server

```
1 import socket
2 import threading
3 import os
4 import struct
5 def handlelocaldns(msg, c_addr, server):
6     print(f'Connected to {c_addr}, and the client is querying
7     for {msg}')
8     msgs=msg.decode()
9     dom=msgs.split('.')
10    kisu=dom[-1]
11    print(kisu)
12    #aaa=input("please enter to open tld server")
13    #client=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
14    address="10.33.2.204 9996"
15    server.sendto(address.encode(),c_addr)
16
17
18    # msg,_=client.recvfrom(1024)
19    # print(msg.decode())
20    #
21    # msg,_=client.recvfrom(1024)
22    # server.sendto(msg,c_addr)
23 def main():
24     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25
26     server.bind(('10.33.2.203',9195))
27
28     print("root server is running")
29
30     while True:
31         msg,c_addr =server.recvfrom(1024)
32         flag="Root server working"
33         #server.sendto(flag.encode(),c_addr)
34         handlelocaldns(msg,c_addr,server)
35
36
37 if __name__=='__main__':
38     main()
```

3.2.3 TLD Name Server for .com

```
1 import socket
2 import threading
3 import os
4 import struct
5
```

```

6 def load_file(filename):
7     dns_record = {}
8     with open(filename, "r") as file:
9         for line in file:
10             name, value, r_type, ttl = line.strip().split()
11             name = name.lower() # Convert domain name to
lowercase
12             # print(name)
13             if name not in dns_record:
14                 dns_record[name] = [] # Initialize list if
domain name doesn't exist
15                 dns_record[name].append((value, r_type, int(ttl)))
16     return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
21
22
23
24     #print("locan dns cannot find the ip..so it start the
request for root")
25
26     #kisu=input("press enter for access autorivite")
27
28     add="10.33.2.203 9997"
29     server.sendto(add.encode(), c_addr)
30
31
32
33
34
35
36
37
38
39
40
41
42
43     # domain_name = msg.decode().lower() # Decode the message
from bytes to string
44     # print("Lowercased domain:", domain_name)
45     # if domain_name in dns_record:
46     #     records = dns_record[domain_name][-1] # Get the last
record for the domain
47     # # Construct the response by joining the last record's
components

```

```

48
49     #         result = domain_name+" ".join(map(str, records))
50     # else:
51     #         result = "Not found. You have to register this domain
        name."
52
53     # Send the response to the client
54
55
56 def main():
57     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
58
59     global dns_record
60     dns_record=load_file("dns_records.txt")
61     server.bind(('10.33.2.204',9996))
62
63     print("tld server is running")
64
65     while True:
66         msg,c_addr =server.recvfrom(1024)
67         th=threading.Thread(target=handle_dns_query_iterative,
        args=(msg,c_addr,server))
68         th.start()
69
70
71
72 if __name__=='__main__':
73     main()

```

3.2.4 TLD Name Server for .org

```
1 import socket
2 import threading
3 import os
4 import struct
5
6 def load_file(filename):
7     dns_record = {}
8     with open(filename, "r") as file:
9         for line in file:
10             name, value, r_type, ttl = line.strip().split()
11             name = name.lower() # Convert domain name to
lowercase
12             # print(name)
13             if name not in dns_record:
14                 dns_record[name] = [] # Initialize list if
domain name doesn't exist
15                 dns_record[name].append((value, r_type, int(ttl)))
16     return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
21
22     #print("locan dns cannot find the ip..so it start the
request for root")
23
24     kisu=input("press enter for access autorivite")
25     client=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
26     server_addr=(("10.33.3.38",9997))
27
28     client.sendto(msg,server_addr)
29
30
31     msg,_=client.recvfrom(1024)
32
33
34
35     server.sendto(msg, c_addr)
36
37
38
39
40
41
42
43
```

```

44     # domain_name = msg.decode().lower() # Decode the message
    from bytes to string
45     # print("Lowercased domain:", domain_name)
46     # if domain_name in dns_record:
47     #     records = dns_record[domain_name][-1] # Get the last
        record for the domain
48     # # Construct the response by joining the last record's
        components
49
50     #     result = domain_name+" ".join(map(str, records))
51     # else:
52     #     result = "Not found. You have to register this domain
        name."
53
54     # Send the response to the client
55
56
57 def main():
58     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
59
60     global dns_record
61     dns_record=load_file("dns_records.txt")
62     server.bind(('10.33.3.20',9999))
63
64     print("tld server is running")
65
66     while True:
67         msg,c_addr =server.recvfrom(1024)
68         th=threading.Thread(target=handle_dns_query_iterative,
        args=(msg,c_addr,server))
69         th.start()
70
71
72
73 if __name__=='__main__':
74     main()

```


3.2.5 Authoritative Name Server is same like before

3.2.6 Experimental Result for task 2

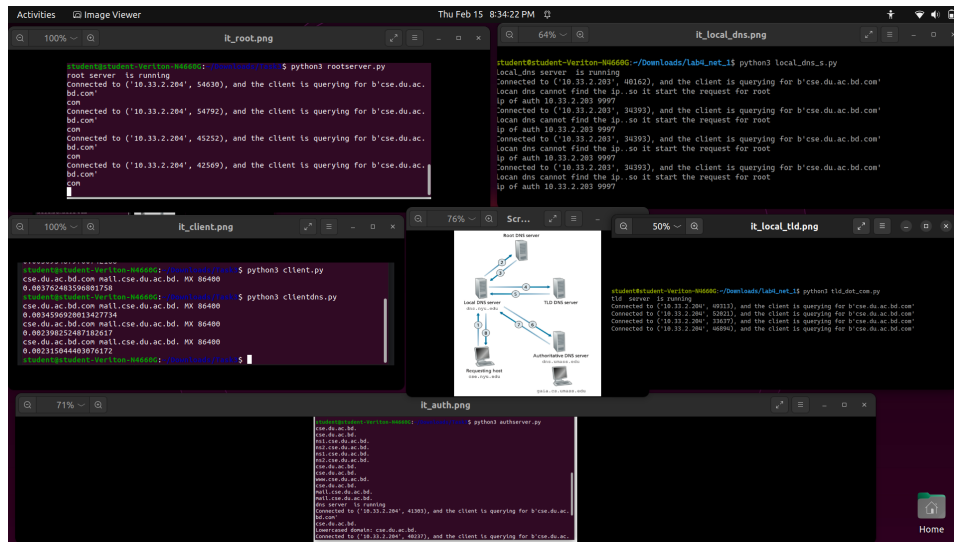


Figure 3: Iterative Dns resolution

3.3 Task 3: Recursive DNS resolution

To implement this approach, we make 5 files. They are client, local dns server, root server, tld server and authoritative server. Every time we use the same authoritative server and client server, so we don't mention it in future.

3.3.1 Local DNS Server

```
1 import socket
2 import threading
3 import os
4 import struct
5
6 def load_file(filename):
7     dns_record = {}
8     with open(filename, "r") as file:
9         for line in file:
10             name, value, r_type, ttl = line.strip().split()
11             name = name.lower() # Convert domain name to lowercase
```

```

12         # print(name)
13         if name not in dns_record:
14             dns_record[name] = [] # Initialize list if
domain name doesn't exist
15             dns_record[name].append((value, r_type, int(ttl)))
16         return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20
21
22     flag="true..local dns process to find your data from root"
23     server.sendto(flag.encode(), c_addr)
24
25     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
26
27     print("locan dns cannot find the ip..so it start the
request for root")
28
29     # kisu=input("press enter for access root")
30     client=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
31     server_addr=(("10.33.2.203",9195))
32
33     client.sendto(msg,server_addr)
34
35     flag,_=client.recvfrom(1024)
36     print(flag.decode())
37
38
39     msg,_=client.recvfrom(1024)
40
41
42
43     server.sendto(msg, c_addr)
44
45
46
47
48
49
50
51
52     # domain_name = msg.decode().lower() # Decode the message
from bytes to string
53     # print("Lowercased domain:", domain_name)
54     # if domain_name in dns_record:
55     #     records = dns_record[domain_name][-1] # Get the last
record for the domain

```

```

56     # # Construct the response by joining the last record's
    components
57
58     #     result = domain_name+" ".join(map(str, records))
59     # else:
60     #     result = "Not found. You have to register this domain
    name."
61
62     # Send the response to the client
63
64
65 def main():
66     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
67
68     global dns_record
69     dns_record=load_file("dns_records.txt")
70     server.bind(('10.33.2.204',9993))
71
72     print("local_dns server is running")
73
74     while True:
75         msg,c_addr =server.recvfrom(1024)
76
77         th=threading.Thread(target=handle_dns_query_iterative,
    args=(msg,c_addr,server))
78         th.start()
79
80
81
82 if __name__=='__main__':
83     main()

```

3.3.2 Root DNS Server

```
1 import socket
2 import threading
3 import os
4 import struct
5 def handlelocaldns(msg, c_addr, server):
6     print(f'Connected to {c_addr}, and the client is querying
7     for {msg}')
8     msg=msg.decode()
9     dom=msg.split('.')
10    kisu=dom[-1]
11    print(kisu)
12    #aaa=input("please enter to open tld server")
13    client=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
14    server_addr=(("10.33.2.204",9996))
15
16    client.sendto(msg,server_addr)
17    msg,_=client.recvfrom(1024)
18    print(msg.decode())
19
20    msg,_=client.recvfrom(1024)
21    server.sendto(msg,c_addr)
22 def main():
23     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
24
25     server.bind(('10.33.2.203',9195))
26
27     print("root server is running")
28
29     while True:
30         msg,c_addr =server.recvfrom(1024)
31         flag="Root server working"
32         server.sendto(flag.encode(),c_addr)
33         handlelocaldns(msg,c_addr,server)
34
35 if __name__=='__main__':
36     main()
```

3.3.3 TLD Name Server for .com

```
1 import socket
2 import threading
3 import os
4 import struct
5
6 def load_file(filename):
7     dns_record = {}
```

```

8     with open(filename, "r") as file:
9         for line in file:
10             name, value, r_type, ttl = line.strip().split()
11             name = name.lower() # Convert domain name to
lowercase
12             # print(name)
13             if name not in dns_record:
14                 dns_record[name] = [] # Initialize list if
domain name doesn't exist
15                 dns_record[name].append((value, r_type, int(ttl)))
16             return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
21
22
23     flag="true..tld is working"
24     server.sendto(flag.encode(), c_addr)
25     #print("locan dns cannot find the ip..so it start the
request for root")
26
27     #kisu=input("press enter for access autorivite")
28     client=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
29     server_addr=(("10.33.2.203",9997))
30
31     client.sendto(msg,server_addr)
32     msg,_=client.recvfrom(1024)
33     print(msg)
34
35     msg,_=client.recvfrom(1024)
36
37
38
39     server.sendto(msg, c_addr)
40
41
42
43
44
45
46
47
48     # domain_name = msg.decode().lower() # Decode the message
from bytes to string
49     # print("Lowercased domain:", domain_name)
50     # if domain_name in dns_record:
51     #     records = dns_record[domain_name][-1] # Get the last

```

```

52     record for the domain
53     # # Construct the response by joining the last record's
54     components
55
56     #     result = domain_name+" ".join(map(str, records))
57     # else:
58     #     result = "Not found. You have to register this domain
59     name."
60
61     # Send the response to the client
62
63 def main():
64     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
65
66     global dns_record
67     dns_record=load_file("dns_records.txt")
68     server.bind(('10.33.2.204',9996))
69
70     print("tld server is running")
71
72     while True:
73         msg,c_addr =server.recvfrom(1024)
74         th=threading.Thread(target=handle_dns_query_iterative,
75         args=(msg,c_addr,server))
76         th.start()
77
78 if __name__=='__main__':
79     main()

```

3.3.4 Authoritative Name Server for .Same like before

3.3.5 Experimental Result for task 3

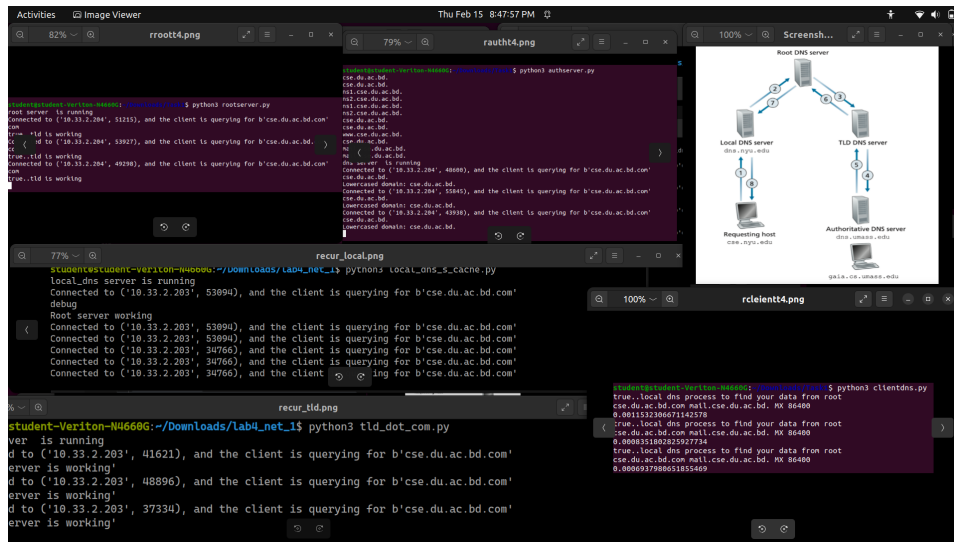


Figure 4: Recursive Dns resolution

3.4 Task 4: Extending the System

Here ,we just modify the authorivite server so that it can delete the same domain name that is big than its min ttl value.And we update the local and TLD server,actually we update the local server for dns caching.

3.4.1 Authoritive DNS Server

```
1 import socket
2 import threading
3 import os
4 import struct
5
6
7 def load_file(filename):
8     dns_record = {}
9     with open(filename, "r") as file:
10         for line in file:
11             name, value, r_type, ttl = line.strip().split()
```

```

12         name = name.lower() # Convert domain name to
lowercase
13         print(name)
14         if name not in dns_record:
15             dns_record[name] = [] # Initialize list if
domain name doesn't exist
16             dns_record[name].append((value, r_type, int(ttl)))
17         return dns_record
18
19
20 def handle_dns_query_iterative(msg, c_addr, server):
21     flag = "Auth server is working"
22     server.sendto(flag.encode(), c_addr)
23     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
24
25     domain_name = msg.decode().lower()
26     modified_domain = domain_name[:-3]
27     print(modified_domain)
28
29     # Decode the message from bytes to string
30     print("Lowercased domain:", modified_domain)
31
32     if modified_domain in dns_record:
33         # Fetch all records for the domain
34         records = dns_record[modified_domain]
35
36         # Find the record with the minimum TTL
37         min_ttl = float('inf') # Initialize with infinity
38         min_ttl_record = None
39
40         for record in records:
41             if record[2] < min_ttl:
42                 min_ttl = record[2]
43                 min_ttl_record = record
44
45         # Construct the response by joining the components of
the record with minimum TTL
46         result = modified_domain + ".com" + " " + " ".join(map(
str, min_ttl_record))
47
48         # Update text file by removing records with TTL greater
than min_ttl
49         dns_record[modified_domain] = [min_ttl_record] # Keep
only the record with minimum TTL
50         with open("ndns_records.txt", "r+") as file:
51             lines = file.readlines()
52             file.seek(0)
53             for line in lines:

```



```

54         parts = line.strip().split()
55         if parts[0].lower() == modified_domain and int(
parts[-1]) > min_ttl:
56             continue # Skip writing this line if TTL
is greater than min_ttl
57             file.write(line)
58             file.truncate() # Remove any extra lines at the
end of the file
59         else:
60             result = "Not found. You have to register this domain
name."
61
62             server.sendto(result.encode(), c_addr) # Send the response
to the client
63
64
65 def main():
66     server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
67
68     global dns_record
69     dns_record = load_file("ndns_records.txt")
70     server.bind(('10.33.2.203', 9997))
71
72     print("dns server is running")
73
74     while True:
75         msg, c_addr = server.recvfrom(1024)
76         th = threading.Thread(target=handle_dns_query_iterative
, args=(msg, c_addr, server))
77         th.start()
78
79
80 if __name__ == '__main__':
81     main()

```

3.4.2 Local DNS Cache Server

```
1 import socket
2 import threading
3 import os
4 import struct
5
6 cache = []
7
8 def handle_dns_query_iterative(msg, c_addr, server):
9     flag = "true..local dns process to find your data from root"
10
11     server.sendto(flag.encode(), c_addr)
12     print(f'Connected to {c_addr}, and the client is querying for {msg}')
13     #print("local dns cannot find the ip, so it starts the request for root")
14     debug="debug"
15     for item in cache:
16         parts = item.split() # Splitting the string by space
17         debug=parts[0]
18         if parts[0] == msg.decode():
19             server.sendto(item.encode(), c_addr)
20
21     return
22
23 print(debug)
24 client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25 server_addr = ("10.33.2.203", 9195)
26
27 client.sendto(msg, server_addr)
28
29 flag, _ = client.recvfrom(1024)
30 print(flag.decode())
31
32 msg, _ = client.recvfrom(1024)
33 server.sendto(msg, c_addr)
34
35 cache.append(msg.decode())
36
37 def main():
38     server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
39     global dns_record
40
41     server.bind(('10.33.2.204', 9993))
42     print("local_dns server is running")
43
44     while True:
45         msg, c_addr = server.recvfrom(1024)
46         th = threading.Thread(target=handle_dns_query_iterative
```

```

    , args=(msg, c_addr, server))
45     th.start()
46
47 if __name__ == '__main__':
48     main()

```

3.4.3 TLD Name Cache Server for .com

```

1 import socket
2 import threading
3 import os
4 import struct
5
6 def load_file(filename):
7     dns_record = {}
8     with open(filename, "r") as file:
9         for line in file:
10             name, value, r_type, ttl = line.strip().split()
11             name = name.lower() # Convert domain name to
lowercase
12             # print(name)
13             if name not in dns_record:
14                 dns_record[name] = [] # Initialize list if
domain name doesn't exist
15                 dns_record[name].append((value, r_type, int(ttl)))
16     return dns_record
17
18
19 def handle_dns_query_iterative(msg, c_addr, server):
20     print(f'Connected to {c_addr}, and the client is querying
for {msg}')
21
22
23     flag="true..tld is working"
24     server.sendto(flag.encode(), c_addr)
25     #print("locan dns cannot find the ip..so it start the
request for root")
26
27     #kisu=input("press enter for access autorivite")
28     client=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
29     server_addr=(("10.33.2.203",9997))
30
31     client.sendto(msg,server_addr)
32     msg,_=client.recvfrom(1024)
33     print(msg)
34
35     msg,_=client.recvfrom(1024)
36
37

```

```

38
39     server.sendto(msg, c_addr)
40
41
42
43
44
45
46
47
48     # domain_name = msg.decode().lower() # Decode the message
    from bytes to string
49     # print("Lowercased domain:", domain_name)
50     # if domain_name in dns_record:
51     #     records = dns_record[domain_name][-1] # Get the last
    record for the domain
52     # # Construct the response by joining the last record's
    components
53
54     #     result = domain_name+" ".join(map(str, records))
55     # else:
56     #     result = "Not found. You have to register this domain
    name."
57
58     # Send the response to the client
59
60
61 def main():
62     server=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
63
64     global dns_record
65     dns_record=load_file("dns_records.txt")
66     server.bind(('10.33.2.204',9996))
67
68     print("tld server is running")
69
70     while True:
71         msg,c_addr =server.recvfrom(1024)
72         th=threading.Thread(target=handle_dns_query_iterative,
    args=(msg,c_addr,server))
73         th.start()
74
75
76
77 if __name__=='__main__':
78     main()

```

3.4.4 Experimental Result for task 4

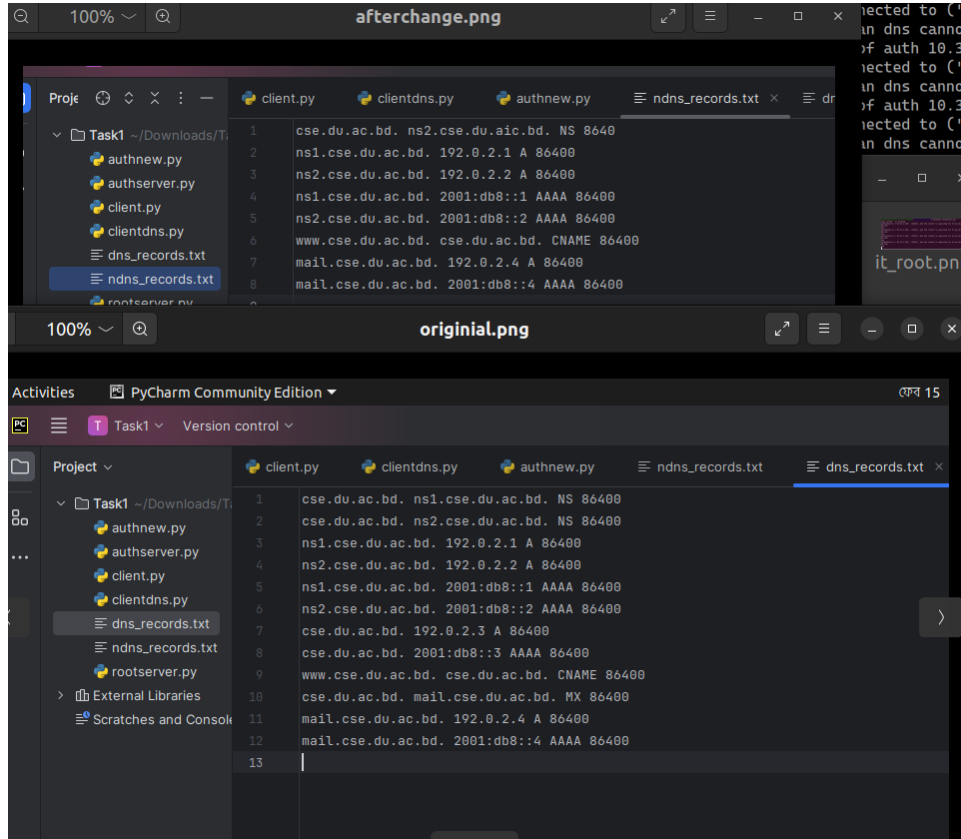


Figure 5: Update authoritative server using ttl value

4 Experience

1. The browser first checks its local DNS cache to determine if the IP address for google.com is already stored, aiming for swift resolution.
2. In case the ISP's DNS servers do not possess the required information, the query is escalated to the root nameservers.
3. TLD nameservers, acting as gatekeepers, direct the query to the authoritative DNS servers responsible for google.com.
4. It was nice to implement server and client and make sense

References

- [1] HTTP : <https://constellix.com/news/dns-record-types>
- [2] DNS Basic: <https://aws.amazon.com/route53/what-is-dns>
- [3] SendingGetandPost:<https://www.digitalocean.com/community/tutorials/java-httpurlconnection-example-java-http-request-get-post>