



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	2
1.1	Objectives	2
2	Theory	2
2.1	Search in DNS cache	2
2.2	Request to ISP's DNS servers	3
2.3	Request to root nameservers	3
2.4	Request to Top Level Domain nameservers	3
2.5	Request to authoritative DNS servers	3
2.6	Retrieve the record	3
2.7	Receive the answer	3
3	Methodology	4
4	Setting up the DNS server	5
4.1	Experimental Result	5
5	Iterative DNS resolution	8
5.1	Experimental Result	8
5.2	Advantages	11
5.3	Disadvantages	11
5.4	Security Implications	11
6	Recursive DNS resolution	12
6.1	Experimental Result	12
6.2	Advantages	16
6.3	Disadvantages	16
6.4	Security Implications	17
7	Extending the system	18
7.1	Experimental Result	18
8	Time Requirement	23
9	Experience	23

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.

```
1 #this functions helps to fetch dns records from
  dns_records.txt
2 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.

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

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

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

```

File Edit Selection View Go Run Terminal Help
Task 1 > Clientpy > Clientpy > dns.records.txt
Server.py Clientpy
1 import socket
2 import struct
3
4 ADDR = ('127.0.0.1', 1234)
5 SIZE = 1024
6 FORMAT = 'utf-8'
7
8 def encode_msg(message):
9     data = message.split()
10    name = data[0]
11    type = data[1]
12
13    flag = 0
14    q = 0
15    a = 1
16    auth_rr = 0
17    add_rr = 0
18
19    ms = (name + ' ' + type).encode('utf-8')
20    packed_data = struct.pack(f"=6H{len(ms)}s", 50, flag, q, a, a,
21    add_rr, ms)
22
23 def main():
24     while True:
25         client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
26
27         message = input("Enter a domain name to send to the server: ")
28         if message == "EXIT":
29             break
30         message = encode_msg(message)
31
32         client.sendto(message, ADDR)
33
34         msg, addr = client.recvfrom(SIZE)
35         print('In bytes: ')
36         print(msg)
37
38         header = struct.unpack("=6H", msg[:12])

```

```

File Edit Selection View Go Run Terminal Help
Task 1 > Clientpy > Clientpy > dns.records.txt
Clientpy - Lab 4 [WSL: Ubuntu] - Visual Studio Code
Server.py Clientpy
1 import socket
2 import struct
3
4 ADDR = ('127.0.0.1', 1234)
5 SIZE = 1024
6 FORMAT = 'utf-8'
7
8 def encode_msg(message):
9     data = message.split()
10    name = data[0]
11    type = data[1]
12
13    flag = 0
14    q = 0
15    a = 1
16    auth_rr = 0
17    add_rr = 0
18
19    ms = (name + ' ' + type).encode('utf-8')
20    packed_data = struct.pack(f"=6H{len(ms)}s", 50, flag, q, a, a,
21    add_rr, ms)
22
23 def main():
24     while True:
25         client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
26
27         message = input("Enter a domain name to send to the server: ")
28         if message == "EXIT":
29             break
30         message = encode_msg(message)
31
32         client.sendto(message, ADDR)
33
34         msg, addr = client.recvfrom(SIZE)
35         print('In bytes: ')
36         print(msg)
37
38         header = struct.unpack("=6H", msg[:12])

```

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 Authoritative Server

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

Listing 2: For TLD Server

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

Listing 3: For Root Server

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

Listing 4: For Client Side

```
16 #this functions helps the clients to request for dns
17     records from Rootserver or TLD or Authoritative server
18 def encode_msg(message):
19
20 #this functions helps the clients to decode message sent
21     from the server
22 def decode_msg(msg):
```

client.py - Task 2 [WSL: Ubuntu] - Visual Studio Code

```

client.py > main
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1198
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1298
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1318
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1398
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1418
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1498
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  1597
  1598
  1599
  1600
  1601
  1602
  1603
  1604
  1605
  1606
  1607
  1608
  1609
  1610
  1611
  1612
  1613
  1614
  1615
  1616
  1617
  1618
  1619
  1620
  1621
  1622
  1623
  1624
  1625
  1626
  1627
  1628
  1629
  1630
  1631
  1632
  1633
  1634
  1635
  1636
  1637
  1638
  1639
  1640
  1641
  1642
  1643
  1644
  1645
  1646
  1647
  1648
  1649
  1650
  1651
  1652
  1653
  1654
  1655
  1656
  1657
  1658
  1659
  1660
  1661
  1662
  1663
  1664
  1665
  1666
  1667
  1668
  1669
  1670
  1671
  1672
  1673
  1674
  1675
  1676
  1677
  1678
  1679
  1680
  1681
  1682
  1683
  1684
  1685
  1686
  1687
  1688
  1689
  1690
  1691
  1692
  1693
  1694
  1695
  1696
  1697
  1698
  1699
  1700
  1701
  1702
  1703
  1704
  1705
  1706
  1707
  1708
  1709
  1710
  1711
  1712
  1713
  1714
  1715
  1716
  1717
  1718
  1719
  1720
  1721
  1722
  1723
  1724
  1725
  1726
  1727
  1728
  1729
  1730
  1731
  1732
  1733
  1734
  1735
  1736
  1737
  1738
  1739
  1740
  1741
  1742
  1743
  1744
  1745
  1746
  1747
  1748
  1749
  1750
  1751
  1752
  1753
  1754
  1755
  1756
  1757
  1758
  1759
  1760
  1761
  1762
  1763
  1764
  1765
  1766
  1767
  1768
  1769
  1770
  1771
  1772
  1773
  1774
  1775
  1776
  1777
  1778
  1779
  1780
  1781
  1782
  1783
  1784
  1785
  1786
  1787
  1788
  1789
  1790
  1791
  1792
  1793
  1794
  1795
  1796
  1797
  1798
  1799
  1800
  1801
  1802
  1803
  1804
  1805
  1806
  1807
  1808
  1809
  1810
  1811
  1812
  1813
  1814
  1815
 
```

The screenshot shows the Visual Studio Code interface with the following details:

- Code Editor:** The file `tld.py` is open, showing Python code for a TLD Server. It includes functions for sending DNS messages and handling connections.
- Terminal:** A terminal window titled "python3" is running the command `python3 -u tld.py`. The output shows the server starting and listening on port 1195. It then receives a message from an MX record and prints the response.
- System:** The taskbar at the bottom shows the date as 2/16/2023 and the time as 7:48 PM.

Figure 5: TLD Server Side View

The screenshot shows the Visual Studio Code interface with the following details:

- Code Editor:** The file `authoritative.py` is open, showing Python code for an Authoritative Server. It includes functions for sending DNS messages and handling connections.
- Terminal:** A terminal window titled "python3" is running the command `python3 -u authoritative.py`. The output shows the server starting and listening on port 1196. It then receives a message from an AAAA record and prints the response.
- System:** The taskbar at the bottom shows the date as 2/16/2023 and the time as 7:49 PM.

Figure 6: Authoritative 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 its 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 Authoritative Server

```
20  def getDnsRecords(): # to fetch "Authoritative type dns
21    records
22  def encode_msg(message): #to sort out the request from
23    the client
24  return packed_data #encode the request in dns format
25
26  def handle_client(data, addr, server): #this function
27    handles the threading for sending to the client the
28    dns records
```

Listing 6: For TLD Server

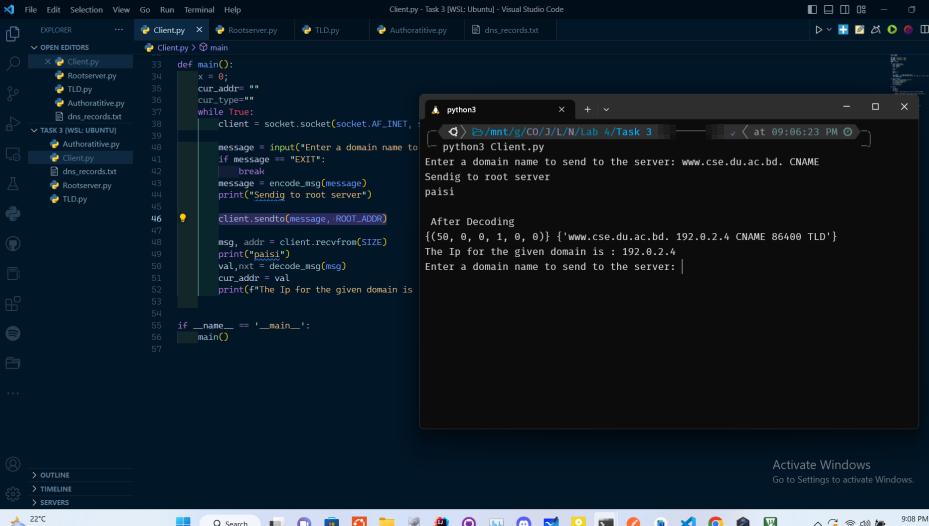
```
25  def getDnsRecords(): # to fetch NX or MS dns records
26  def encode_msg(message): #to sort out the request from
27    the client
28  return packed_data #encode the request in dns format
29
30  def decode_msg(msg): #decode the bit request from the
31    client
32  return ms[1],ms[4]
33
34  def handle_client(data, addr, server): #this function
35    handles the threading for sending to the client the
36    dns records
```

Listing 7: For Root Server

```
33     def getDnsRecords(): # to fetch any dns records except
34         the authoritative dns
35     def encode_msg(message): #to sort out the request from
36         the client
37     return packed_data #encode the request in dns format
38
39
40     def decode_msg(msg): #decode the bit request from the
41         client
42     return ms[1],ms[4]
43
44     def handle_client(data, addr, server): #this function
45         handles the threading for sending to the client the
46         dns records
```

Listing 8: For Client Side

```
41     #this functions helps the clients to request for dns
42         records from Rootserver or TLD or Authoritative server
43     def encode_msg(message):
44
45         #this functions helps the clients to decode message sent
46         from the server
47     def decode_msg(msg):
```



File Edit Selection View Go Run Terminal Help Client.py - Task 3 [WSL:Ubuntu] - Visual Studio Code

EXPLORER OPEN EDITORS Client.py Rootserver.py TLD.py Authoritative.py dns_records.txt

Client.py @ main

```
33     def main():
34         x = 0;
35         cur_addr= ""
36         type="""
37         while True:
38             client = socket.socket(socket.AF_INET,
39             message = input("Enter a domain name to
40             if message == "EXIT":
41                 break
42             message = encode_msg(message)
43             print("Sendig to root server")
44
45             client.sendto(message, ROOT_ADDR)
46
47             msg, addr = client.recvfrom(SIZE)
48             print("Recived")
49             print(msg)
50             cur_addr=decode_msg(msg)
51             cur_addr[0]
52             print(f"The Ip for the given domain is
53
54
55             if __name__ == '__main__':
56                 main()
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```

python3

python3 Client.py

Enter a domain name to send to the server: www.cse.du.ac.bd. CNAME

Sendig to root server

paisi

After Decoding

{50, 0, 0, 1, 0, 0} {'www.cse.du.ac.bd. 192.0.2.4 CNAME 86400 TLD'}

The Ip for the given domain is : 192.0.2.4

Enter a domain name to send to the server: |

Activate Windows
Go to Settings to activate Windows.

22°C Haze

Search

File Edit Selection View Go Run Terminal Help Client.py - Task 3 [WSL:Ubuntu] - Visual Studio Code

EXPLORER OPEN EDITORS Client.py Rootserver.py TLD.py Authoritative.py dns_records.txt

Client.py @ main

```
33     def main():
34         x = 0;
35         cur_addr= ""
36         type="""
37         while True:
38             client = socket.socket(socket.AF_INET,
39             message = input("Enter a domain name to
40             if message == "EXIT":
41                 break
42             message = encode_msg(message)
43             print("Sendig to root server")
44
45             client.sendto(message, ROOT_ADDR)
46
47             msg, addr = client.recvfrom(SIZE)
48             print("Recived")
49             print(msg)
50             cur_addr=decode_msg(msg)
51             cur_addr[0]
52             print(f"The Ip for the given domain is
53
54
55             if __name__ == '__main__':
56                 main()
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```

python3

python3 Client.py

Enter a domain name to send to the server: www.cse.du.ac.bd. CNAME

Sendig to root server

paisi

After Decoding

{50, 0, 0, 1, 0, 0} {'www.cse.du.ac.bd. 192.0.2.4 CNAME 86400 TLD'}

The Ip for the given domain is : 192.0.2.4

Enter a domain name to send to the server: |

Activate Windows
Go to Settings to activate Windows.

22°C Haze

File Edit Selection View Go Run Terminal Help Client.py - Task 3 [WSL:Ubuntu] - Visual Studio Code

EXPLORER OPEN EDITORS Client.py Rootserver.py TLD.py Authoritative.py dns_records.txt

Client.py @ main

```
33     def main():
34         x = 0;
35         cur_addr= ""
36         type="""
37         while True:
38             client = socket.socket(socket.AF_INET,
39             message = input("Enter a domain name to
40             if message == "EXIT":
41                 break
42             message = encode_msg(message)
43             print("Sendig to root server")
44
45             client.sendto(message, ROOT_ADDR)
46
47             msg, addr = client.recvfrom(SIZE)
48             print("Recived")
49             print(msg)
50             cur_addr=decode_msg(msg)
51             cur_addr[0]
52             print(f"The Ip for the given domain is
53
54
55             if __name__ == '__main__':
56                 main()
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```

python3

python3 Client.py

Enter a domain name to send to the server: www.cse.du.ac.bd. CNAME

Sendig to root server

paisi

After Decoding

{50, 0, 0, 1, 0, 0} {'www.cse.du.ac.bd. 192.0.2.4 CNAME 86400 TLD'}

The Ip for the given domain is : 192.0.2.4

Enter a domain name to send to the server: |

Activate Windows
Go to Settings to activate Windows.

22°C Haze

File Edit Selection View Go Run Terminal Help Client.py - Task 3 [WSL:Ubuntu] - Visual Studio Code

EXPLORER OPEN EDITORS Client.py Rootserver.py TLD.py Authoritative.py dns_records.txt

Client.py @ main

```
33     def main():
34         x = 0;
35         cur_addr= ""
36         type="""
37         while True:
38             client = socket.socket(socket.AF_INET,
39             message = input("Enter a domain name to
40             if message == "EXIT":
41                 break
42             message = encode_msg(message)
43             print("Sendig to root server")
44
45             client.sendto(message, ROOT_ADDR)
46
47             msg, addr = client.recvfrom(SIZE)
48             print("Recived")
49             print(msg)
50             cur_addr=decode_msg(msg)
51             cur_addr[0]
52             print(f"The Ip for the given domain is
53
54
55             if __name__ == '__main__':
56                 main()
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
```

python3

python3 Client.py

Enter a domain name to send to the server: www.cse.du.ac.bd. CNAME

Sendig to root server

paisi

After Decoding

{50, 0, 0, 1, 0, 0} {'www.cse.du.ac.bd. 192.0.2.4 CNAME 86400 TLD'}

The Ip for the given domain is : 192.0.2.4

Enter a domain name to send to the server: |

Activate Windows
Go to Settings to activate Windows.

22°C Haze

Figure 7: Client Side View

Figure 8: Root Server Side View

Figure 9: Authoritative Server Side View

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer:** Shows files in the workspace: Client.py, Rootserver.py, TLD.py (the active file), Authoritative.py, dns.records, dns.records.txt, and dns.records.tst.
- Code Editor:** The TLD.py file is open, showing Python code for a TLD server. It includes functions for packing DNS header fields and message into a byte string, sending data to a client or server, and handling AUTH and MX records.
- Terminal:** A terminal window titled "python3 TLD.py" is running. It shows the server starting on port 1195 and receiving a query from a client. The terminal output includes the received message and the server's response.
- Taskbar:** Shows various application icons and the date/time (9:09 PM, 2/6/2023).

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 poisoning 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 Authoritative Server

```
45  def getDnsRecords(): # to fetch "Authoritative type dns
46      records
47  def encode_msg(message): #to sort out the request from
48      the client
49  return packed_data #encode the request in dns format
50
51  def handle_client(data, addr, server): #this function
52      handles the threading for sending to the client the
53      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
52      the client
53  return packed_data #encode the request in dns format
54
55  def decode_msg(msg): #decode the bit request from the
56      client
57  return ms[1],ms[4]
58
59  def handle_client(data, addr, server): #this function
60      handles the threading for sending to the client the
61      dns records
```

Listing 11: For Root Server

```
58  def getDnsRecords(): # to fetch any dns records except
59      the authoritative dns
60  def encode_msg(message): #to sort out the request from
61      the client
62  return packed_data #encode the request in dns format
63
64  def decode_msg(msg): #decode the bit request from the
65      client
```

```

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

```

Listing 12: For Client Side

```

66     #this functions helps the clients to request for dns
67         records from Rootserver or TLD or Authorative server
68
69     def encode_msg(message):

```



```

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

```

The screenshot shows the Visual Studio Code interface with several files open in the Explorer sidebar: Authoritative.py, Client.py, dns_records.txt, Rootserver.py, and TLD.py. The Client.py file is the active editor, showing Python code for a client socket. A terminal window titled 'Python3' is open, showing the execution of 'python3 Client.py' and the interaction with the server. The terminal output includes the message 'Enter a domain name to send to the server: www.cse.du.ac.bd. CNAME' and 'Sendig to root server'. Below this, it shows the output 'After Decoding' with the message '[(50, 0, 0, 1, 0, 0)] {www.cse.du.ac.bd. 192.0.2.4 CNAME 86400 TLD}' and 'The Ip for the given domain is : 192.0.2.4'. The terminal also prompts 'Enter a domain name to send to the server:'. The status bar at the bottom right shows the date and time as '2/16/2023 10:27 PM'.

Figure 11: Client Side View

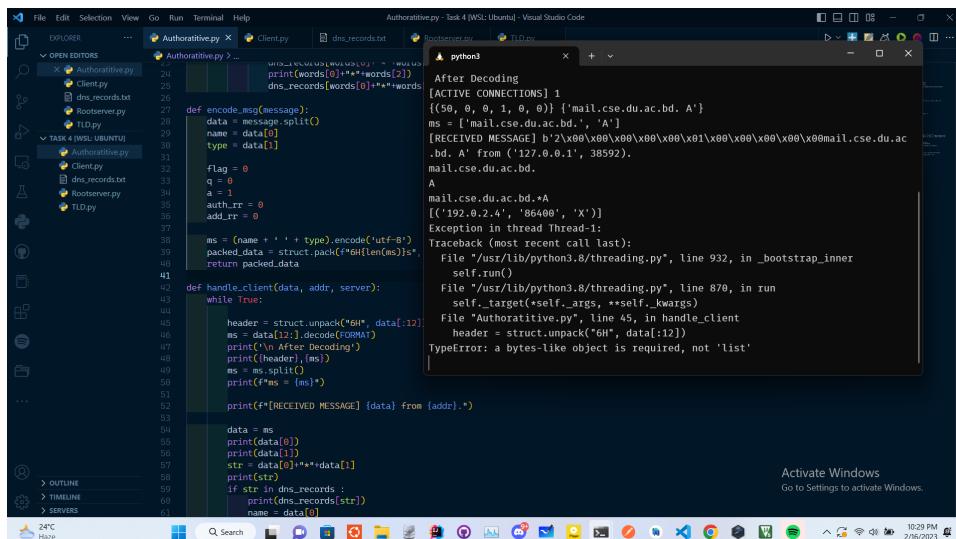
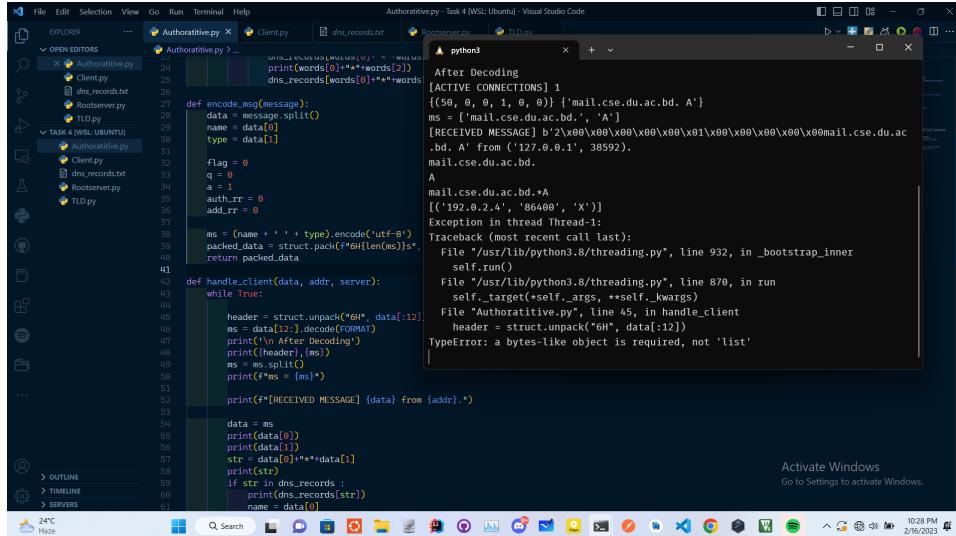
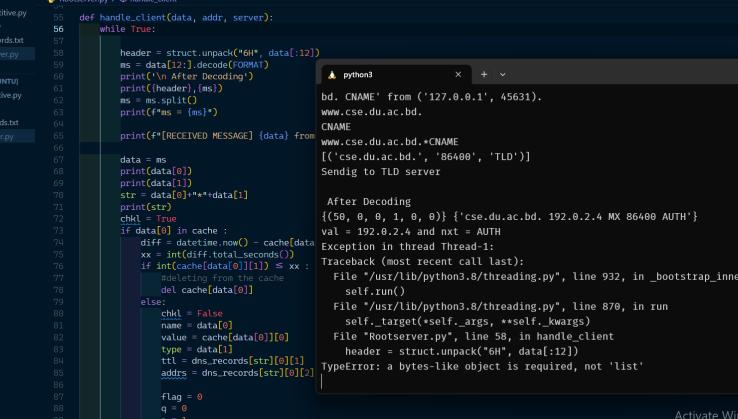


Figure 12: Authoritative Server Side View



```
python3
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.8/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "Rootserver.py", line 58, in handle_client
    header = struct.unpack('!H', data[12])
  TypeError: a bytes-like object is required, not 'list'

  After Decoding
  { (50, 0, 0, 1, 0, 0) } { 'cse.du.ac.bd', 192.0.2.4 MX 86400 AUTH }
  val = 192.0.2.4 and next = AUTH
  Exception in thread Thread-1:
  Traceback (most recent call last):
    File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
      self.run()
    File "/usr/lib/python3.8/threading.py", line 870, in run
      self._target(*self._args, **self._kwargs)
    File "Rootserver.py", line 58, in handle_client
      header = struct.unpack('!H', data[12])
  TypeError: a bytes-like object is required, not 'list'

  55     def handle_client(data, addr, server):
  56         while True:
  57
  58             header = struct.unpack('!H', data[12])
  59             ms = data[12:1].decode('FORMAT')
  60             print('\n After Decoding')
  61             print(header, [ms])
  62             ms = ms.split()
  63             print(ms[0] == ms[1])
  64
  65             print(F" [RECEIVED MESSAGE] {data} from {addr}")
  66
  67             data = ms
  68             print(data[0])
  69             data[0] = str(data[1])
  70             str = data[0] + '*' + data[1]
  71             print(str)
  72             chb = True
  73             if data[0] in cache:
  74                 diff = int(cache[data[0]].cache_time)
  75                 if diff <= 0:
  76                     del cache[data[0]]
  77                     # deleting from the cache
  78                     del cache[data[0]]
  79             else:
  80                 chb = False
  81                 name = data[0]
  82                 value = cache[data[0]][0]
  83                 type = data[1]
  84                 ttl = dns_records[str][0][1]
  85                 addrs = dns_records[str][0][2]
  86
  87                 flag = 0
  88                 q = 0
  89                 a = 1
  90                 auth_rr = 0
  91                 add_rr = 0
  92                 ms = (name + ' ' + value + ' ' + type + ' ' + ttl + ' ' + addrs).encode('utf-8')
```

Figure 13: Root Server Side View

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the current workspace, including `Authoritative.py`, `Client.py`, `dns.records.txt`, `Rootserver.py`, and `TLD.py`.
- Code Editor:** The `TLD.py` file is open, showing Python code for a DNS server. The code handles DNS requests and responses, including the `handle_client` function which handles queries for the 'cse.du.ac.bd' domain.
- Terminal:** A terminal window titled "python3" is running the command `python3 TLD.py`. The output shows the server starting and listening on port 1195. It then receives a message from the client and prints the received message.
- Status Bar:** Shows the date and time (2/26/2023, 10:26:26 PM), the number of lines (126), and the file name (TLD.py).

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files in the current workspace, including `Authoritative.py`, `Client.py`, `dns.records.txt`, `Rootserver.py`, and `TLD.py`.
- Code Editor:** The `TLD.py` file is open, showing the same Python code for the DNS server. The code handles DNS requests and responses, including the `handle_client` function.
- Terminal:** A terminal window titled "python3" is running the command `python3 TLD.py`. The output shows the server starting and listening on port 1195. It then receives a message from the client and prints the received message. However, the terminal shows an error message at the end:

```

After Decoding
{('50, 0, 0, 1, 0, 0) { 'mail.cse.du.ac.bd. 192.0.2.4 A 86400 X'}
val = 192.0.2.4 and nxt = X
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python3.8/threading.py", line 932, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.8/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "TLD.py", line 56, in handle_client
    header = struct.unpack("6H", data[12:])
TypeError: a bytes-like object is required, not 'list'

```

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].