Nishanth Muruganandam
110276247

Flattening the DNS using DHT

1. Introduction:

- *Problem*: DNS – Domain Name Server provides the exact IP address for an human readable string. Name servers are difficult and time-consuming to administer; ordinary people typically rely on ISPs to serve their name data. In 1992, Danzig et al. [4] found that most DNS traffic was caused by misconfiguration and faulty implementation of the name servers. In 2000,Jung et al. [1] found that approximately 35% of DNS queries never receive an answer or receive a negative answer, and attributed many of these failures to improperly configured name servers or incorrect name server records.

- *Plan:* Serving DNS data over Chord eliminates the need to have every system administrator be an expert in running name servers. It provides better load balance, since the concept of root server is eliminated completely. Finally,it provides robustness against denial-of-service attacks since disabling even a sizable number of hosts in the Chord network will not prevent data from being served.

-*Statement:* Develop an Application that on receiving an input String from any user in any country and return the correct IP address associated with that string.

-*Limitation:* This application does not provide security by incorporating the DNSSEC developed in 1990s.

2.Design:

- The design is based on Country's topology.
- An administrator sets up an DHT Network using some protocol(CHORD in this case)
- A User from each country joins this network setup by the administrator(use that node as a Bootstrap node).
- Every country insert their DNS – IP mappings in the network.
- Now, any user(it can be a browser too) will use this DHT network's API to get the value for the user entered URL.

The APIs the application extends to the outside world are: insert() & retrieve(). Only administrator can createNetwork() and user from a country who is already not in the network can connect to network using joinNetwork().

3. Implementation:

The implementation of CHORD protocol is taken from openchord[4]. It is written in Java. The application is also written in Java which can scale better and is also portable.

```
Workflow:
      1. Administrator login
      2. Set Admin Node as a Bootstrap node
      3. createNetwork()
      4. For user in setOfUsers:
            user.joinNetwork()
            user.insertData()
        5. Fork a process P that interacts with the external world
    Process P:
        1. Wait for command from user.
        2. case insert:
            chord.insertData()
           case retrieve:
            print chord.retrieve(DNS)
```

4. Performance Measurement:

Because of the time constraint and scope of the project, hierarchical DNS couldn't be implemented and hence comparison could not established. However, the time measurement for setting up of chord network and retrieval queries from users in random countries is present in:

Time to Setup Network: ~7mins[Includes inserting data into the network]

Query From Country	**Time (in ms)**
JM	1
PA	1
KW	5
SN	1
BN	0
KE	0
TT	0
CA	0
KE	0
AM	1

The data has been collected from http://public-dns.tk/. There were 98076 DNS – IP mappings. Based on the country of the origin of IP, it is simulated as if the User from that country is entering the data into the Chord network.

(An elaborate comparison of Hierarchal DNS and Flattened DNS is provided in [2])

5. Correctness Verification:

The correctness of OpenChord is verified by my teammate. There was a NULL POINTER EXCEPTION arising because of an improper condition check which was uncovered during the implementation. For correctness check of application, various error test cases were written as a part of the application which can be found in the JAR itself. Test cases that were tested are:

1. Querying for the KEY not yet inserted in the network.
2. Querying when there is only one node in the network.
3. User who is not administrator errs when he tries to setup the network.

Future Work:

- *Incorporate security by following the DNSSEC extensions.*
- *Provide an GUI to the application.*
- *Insert the data in an Asynchronous fashion.*

Lines of Code written:

1. No. of Packages: 11
2. No. of Classes : 20
3. Approximate LOC: 750

Steps to repeat tasks performed:

1. Make sure Java 1.5+ is installed.
2. In EclipseIDE import the JAR that has been shipped.
3. In the package edu.stonybrook.asynchronous.project.entrypoint, run the PrepareNetwork.java

(In case of use in real-time, this structure has to be modified. It is designed this way for easy test purposes)

References:

[1] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. Dns performance and the effectiveness of caching. In Proceedings of the ACM SIGCOMM Internet Measurement Workshop ’01, San Francisco, California, November 2001

[2]Russ Cox, Athicha Muthitacharoen, Robert T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. MIT Laboratory for Computer Science

[3] Krishna Gummadi, DHTs and their Application to the Design of PeertoPeer Systems, Presentation in University of Washington

[4] http://open-chord.sourceforge.net/