Course Title: Internet Protocols
Instructor: Dr. Muhammad Shahzad

DOMAIN NAME SYSTEM APPLICATION

December 3, 2018

Chakshu Singla
Chandan Sharma
Pavithra Iyer
Tushita Roychaudhury

BREAKDOWN OF INDIVIDUAL CONTRIBUTIONS

Component	**Component weightage**	**Chakshu Singla**	**Chandan Sharma**	**Pavithra Iyer**	**Tushita Roychaudhury**
High-level design	0.1	20	20	30	30
Algorithm development	0.20	25	25	25	25
Coding	0.35	25	25	25	25
Debugging	0.15	25	25	25	25
Setup	0.1	30	30	20	20
Report writing	0.1	25	25	25	25
Per student aggregate contribution		**25**	**25**	**25**	**25**

I. INTRODUCTION

The Domain Name System (DNS) is a distributed database to map between hostnames and IP address. The DNS serves as the phone book for the Internet by translating human-friendly computer hostnames into IP addresses (also vice-versa). The importance of the DNS can be understood from the fact that be any networked application it must do a name-to-IP conversion for communication. This process is called DNS Name resolution. The following types of servers are involved in this resolution:

- Root nameserver - The root server is the first step in translating humanly-readable hostnames into IP addresses. Typically, it serves as a reference to other more specific locations.
- TLD nameserver - The top-level domain server (TLD) is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In example.com, the TLD server is “com”).
- Authoritative nameserver - This is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the server that made the initial request.

A typical DNS lookup can have 2 types of DNS queries:

- Recursive query - In a recursive query, a DNS client requires that a DNS server will respond to the client with either the final resolved IP address of the requested resource/host or an error message if the server can't find the record.
- Iterative query - in this situation the DNS client will allow a DNS server to return the best answer it can. If the queried DNS server does not have a match for the query name, it will return a referral to a DNS server authoritative for a lower level of the domain namespace. The DNS client will then make a query to the referral address. This process continues with additional DNS servers down the query chain until either an error or timeout occurs.

Another important consideration for a DNS implementation is DNS caching. It involves storing data closer to the requesting client so that the DNS query can be resolved earlier and additional queries further down the DNS lookup chain can be avoided, thereby improving load times and reducing bandwidth/CPU consumption. DNS data can be cached in a variety of locations, each of which will store DNS records for a set amount of time determined by a time-to-live (TTL).

Response times achieved during DNS Name Resolution can vary depending on parameters like the type of query i.e. iterative or recursive, the size of the cache on the servers, or the type of connection i.e. TCP or UDP. The objective of our project is to implement a DNS application under these considerations and carry out an analysis of varying the mentioned parameters on the response times.

II. DESIGN

Figures 1 and 2 show the high-level architecture of our application.

1. **Requesting Host**

   This is the client or the initiator of the DNS query.

   - Generation of requests: In our implementation we generate requests with an exponential inter-arrival time and hence our DNS queries follow a Poisson Distribution.
   - Types of Connection: The client can either initiate a TCP request or a UDP request
   - Types of DNS query: The client can request a DNS query to be resolved either iteratively or recursively.

   The user is able to choose parameters mentioned above while running the DNS client as follows:
   python client.py tcp/udp itr/rec

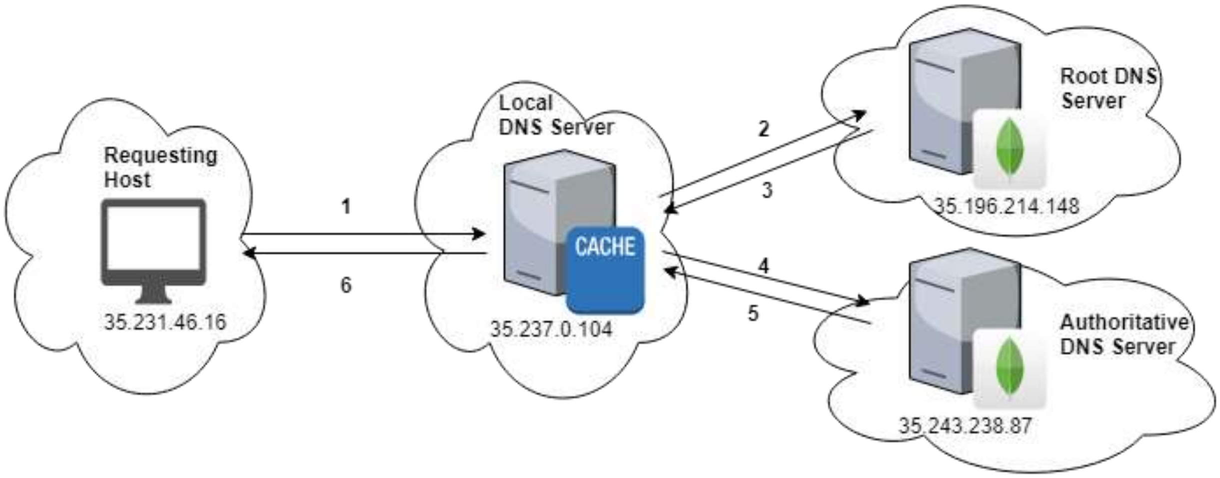

Fig.1 Iterative DNS Lookup

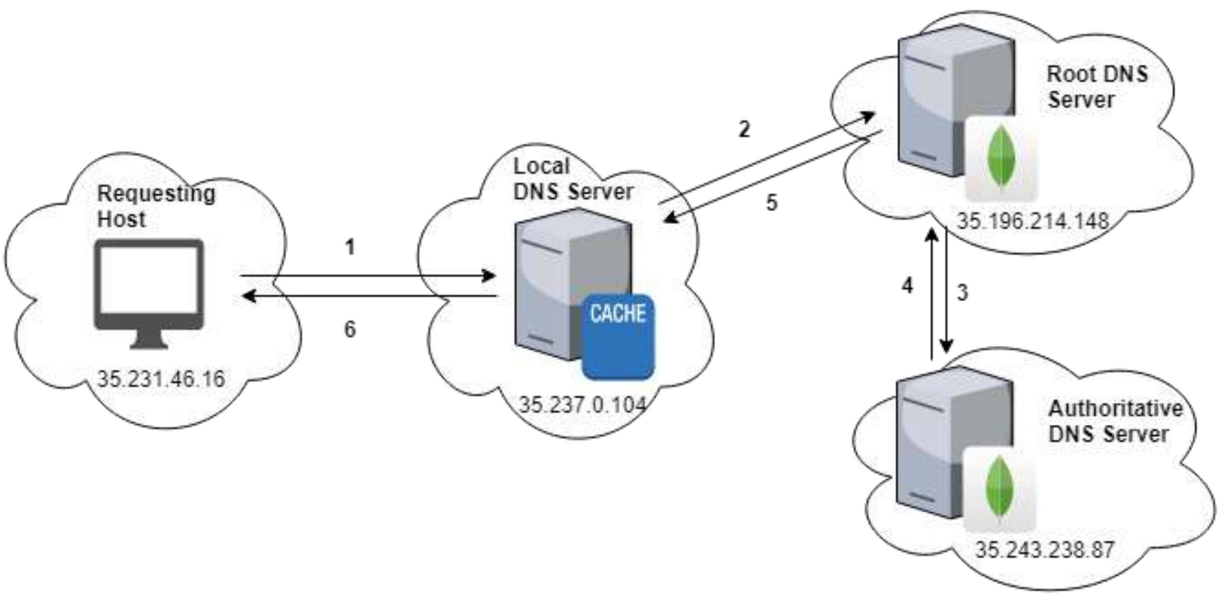

Fig.2 Recursive DNS Lookup

2. **Local DNS Server**

This server accepts the parameterized client query and forwards the TCP or UDP connection request in an iterative or recursive manner.

Before forwarding the request to the subsequent server, it first checks its cache for the host-to-IP resolution. The cache implementation in our Local DNS follows the Least Recently Used (LRU) policy.

The local DNS also maintains a log file for all the queries and their corresponding responses. This also gives us insights into cache hits or misses of the requests arriving at the server.

The Local DNS can be configured to accept TCP or UDP connections. To run the Local DNS, the following command is used with the desired connection type:

python3 dns_local_server.py tcp/udp

3. **Root DNS Server**

The root server maintains a local database containing the mapping of hostnames to their respective authoritative server addresses. Depending on type of DNS query – recursive or iterative – the root server either responds to the local server with the address of the next hop server i.e. the authoritative server or itself makes a call to the authoritative server to obtain the destination IP address and responds back to the local DNS with the fully resolved hostname.

The Root DNS can be configured to accept TCP or UDP connections. To run the Root DNS, the following command is used with the desired connection type:

python3 dns_root_server.py tcp/udp

4. **Authoritative DNS Server**

The authoritative server maintains a local database containing the mapping of hostnames to IP address of the server that client eventually wants to connect to.

The server responds to every request with the corresponding IP address of the hostname.

The Authoritative DNS can be configured to accept TCP or UDP connections. To run the Authoritative DNS, the following command is used with the desired connection type:

python3 dns_authoritative_server.py tcp/udp

5. **Header Format**

In our implementation, we used the header format outlined in IETF as shown in Figure 3.

```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Fig.3 Header Format

Where:

ID	A 16-bit identifier assigned by the program that generates any kind of query. This identifier is copied the corresponding reply and can be used by the requester to match up replies to outstanding queries.
QR	A one-bit field that specifies whether this message is a query (0), or a response (1).
OPCODE	A four-bit field that specifies kind of query in this message.
AA	Authoritative Answer - this bit is valid in responses, and specifies that the responding name server is an authority for the domain name in question section.
TC	TrunCation - specifies that this message was truncated due to length greater than that permitted on the transmission channel.
RD	Recursion Desired - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively.
RA	Recursion Available - this be is set or cleared in a response and denotes whether recursive query support is available in the name server.
Z	Reserved for future use. Must be zero in all queries and responses.
RCODE	Response code - this 4-bit field is set as part of responses.

QDCOUNT	an unsigned 16-bit integer specifying the number of entries in the question section.
ANCOUNT	an unsigned 16-bit integer specifying the number of resource records in the answer section.
NSCOUNT	an unsigned 16-bit integer specifying the number of name server resource records in the authority records section.
ARCOUNT	an unsigned 16-bit integer specifying the number of resource records in the additional records section.

6. **Database Schema**
The root and authoritative DNS servers each contain a MongoDB instance that stores the mapping tables of their respective zones. Figure 4 shows an example of our database schema:

Fig.4 Database Schema at Root and Authoritative DNS

III. IMPLEMENTATION

We set up virtual machines, one each corresponding to Client, Local DNS, Root DNS, and Authoritative server on Google Cloud Platform.

MongoDB was used to store database mappings at the root and authoritative server level.

We used python3 for the implementation of our Domain Name System Application. Following are the most signification python libraries that aided our application:

- lru-dict
- pandas
- socket
- logging
- pymongo

Description of source code files:

- **dns_root_server.py**
The code in this file enables root server functionality to make TCP or UDP connections based on the RD flag present in the header of the message, retrieve host-to-IP mappings from its local MongoDB database, respond back to the local DNS in case of iterative queries or communicate with the authoritative DNS in case of recursive queries.
The root server also maintains a log file root_server.log which has been implemented using the python library logging.

- **dns_authoritative_server.py**
The code in this file enables authoritative server functionality to make TCP or UDP connections based on the RD flag present in the header of the message, retrieve host-to-IP mappings from its local MongoDB database and respond to the root DNS with the retrieved mappings.
This server also maintains a log file auth_server.log which has been implemented using the python library logging.

- **dns_local_server.py**
This file implements functionality of the local DNS wherein the server can make TCP or UDP connections based on the RD flag present in the header of the message, and then depending on the client request, launch an iterative or a recursive query to the root DNS.
The code also implements a local cache using the python library lru-dict which follows an LRU policy and maintains a log file local_server.log which has been

implemented using the python library logging.

- **client.py**
  We follow the Poisson distribution to generate DNS queries. We thus sample inter-request time from the exponential distribution to implement this. The client is able to make TCP or UDP connections with the local DNS and is also able to specify the type of query – iterative or recursive – that it wants to follow for its host-to-IP resolution request.
  Logging has also been implemented in the client code and can be checked in the file client.log.

- **dns_utility.py**
  Common methods for operations like building a DNS query, sending and receiving responses, and parsing responses, that are used across the code base are included in this file.

IV. RESULTS AND DISCUSSION

To validate our implementation, we compare the responses of our queries to the actual host-to-IP mappings at the client and are able to report 100% accurate results.

The distribution of inter-request arrival time is shown in figure 5. As we can see, the query requests have been simulated to follow a Poisson distribution.

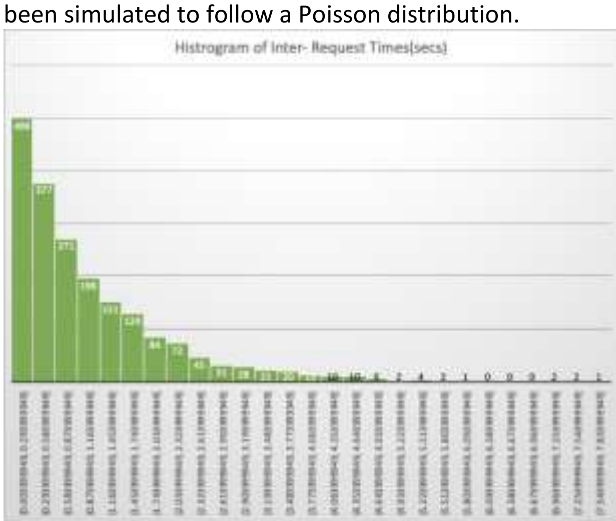

Fig.5 Poisson distribution followed by requests

We also analyzed the results of our implementation and had the following observations about the Response Times subject to different parameters:

1. TCP vs UDP

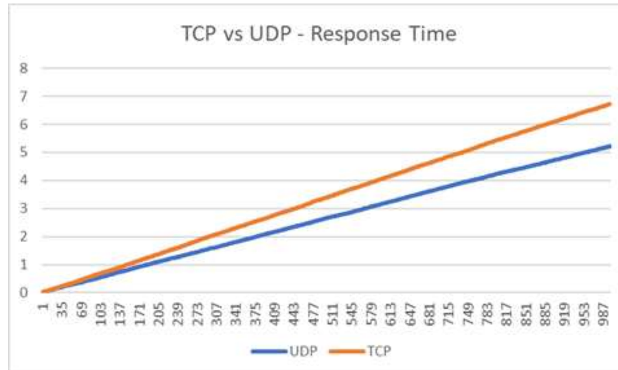

Fig.6 Effect of TCP vs UDP on Response Time

As seen in figure 6 and is expected, the response times for TCP requests are higher than those utilizing UDP due to the added handshake overhead of TCP.

2. Increasing Cache Sizes

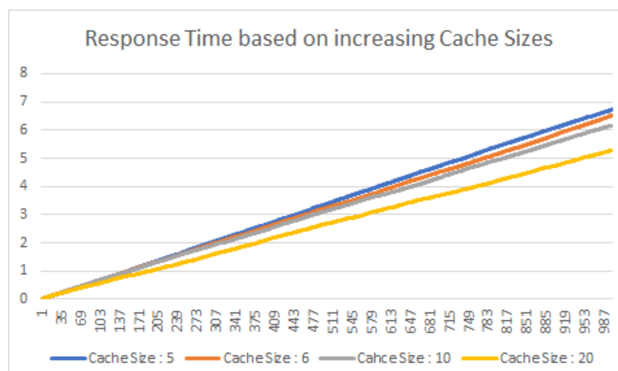

Fig.7 Effect of increasing cache size on Response Time

We see from the above plot in figure 7 that on increasing the cache size, the response time reduces. This is expected since, as cache size increases, more and more requests can be served from the cache of the local DNS thus eliminating the need to forward resolution requests to the root DNS server.

3. Iterative vs Recursive

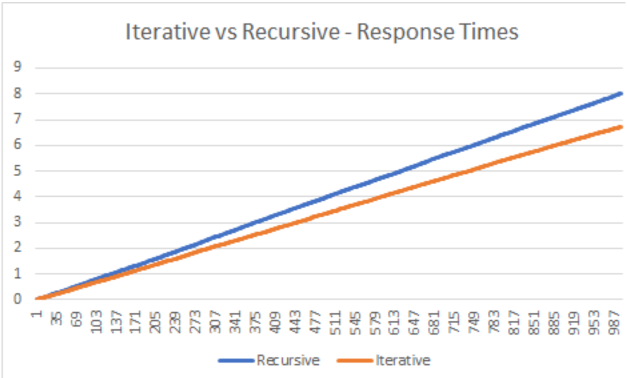

Fig. 8 Effect of Iterative vs Recursive queries on Response Time

Our analysis shows that recursive queries entail higher response times than iterative queries. This can be seen in figure 8.

V. RELATED WORK AND REFERENCES

[1] Message Format (DNS RFC): https://www.ietf.org/rfc/rfc1035.txt
[2] Learning Mongo DB: https://www.tutorialspoint.com/mongodb/mongodb_create_database.htm
[3] DNS Fundamentals: https://www.cloudflare.com/learning/dns/what-is-dns/