DNS On Top Of Chord

Dinker Goel, Debmalya Kundu, Dheeraj Nuthalapati

Department of Computer Science, Stony Brook University

Github Repository: Evaluating DNS on Top Of Chord

Presentation: Evaluating DNS on Top Of Chord

1 Introduction

Domain Name System is the 'Beating Heart' of all IP networks that makes the core functions of the Internet work. As traffic escalates, communications service providers (CSPs) struggle to provide instant capacity for on-demand requests and other traffic, therefore, they need solutions that scale as needed to meet demand without 'missing a beat'. In its simplest terms the Domain Name System (DNS) is the network service that translates all the user requests for domain names or URLs to the Internet Protocol (IP) addresses where the desired resources can be found.
Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes. Peers are both suppliers and consumers of resources, in contrast to the traditional client–server model in which the consumption and supply of resources is divided. A number of recent systems are built on top of peer to peer distributed hash lookup systems. Lookups for keys are performed by routing queries through a series of nodes; each of these nodes uses a local routing table to forward the query towards the node that is ultimately responsible for the key. These systems can be used to store data, for example, as a distributed hash table or file system.

In this paper, we explore a storage system for the DNS, using a peer to peer distributed hash table, DistHash, on top of the Chord ring in order to enhance DNS lookups, balancing load, and fault tolerance of nodes. It serves as an added benefit in mitigating administrative issues existential with current DNS. One way to keep the DNS Hierarchy without redesigning the entire DNS hierarchy is through the use of DNSSEC, though this can prove to be inefficient as the distance between the original host and the final location of a hostname increases.
Here, we attempt to evaluate these methods on the basis of different metrics and will draw visual conclusions based on these results. While working with multiple record types, Chord DNS provides better load balancing in a distributed network as well as less dependency on single root servers. This report is organized in the following manner. First, we have stated the general problem statement and initial hypothesis, followed by the approach and implementation details. This section also explains the working implementation of Chord and integrating DNS to resolve hostnames. Next, we have performed experimental comparisons between tools such as dig, dnssec, and our Chord implementation by varying different parameters such as

number of nodes, value of m for computing hash results. Based on this, we used Python plotting libraries to generate visual representations for a better understanding of the various evaluation results. Finally, we conclude our work with a brief summary of our experiments, discussing the state-of-the art DNS systems, and future scope of this project.

2 Background Research and Related Work:

In this section, we will give an overview of the major components involved in the implementation of a P2P distributed network. These components include Chord and its implementation details, hashing methods, creation of finger tables, usage of distalgo, and last, the required resolution of DNS.

2.1 Overview of Chord:

Chord is a peer-to-peer protocol which presents a new approach to the problem of efficient location. Chord uses routed queries to locate a key with a small number of hops, which stays small even if the system contains a large number of nodes. The major distinguishable factor here is its simplicity, provable performance, and provable correctness.

2.2 Attributes of Chord:

- Decentralization: In the Chord system, there is no central server as compared to traditional DNS systems. Each node is of the same importance as any other node. Therefore, the system is very robust, since it does not have a single point of failure.
- Availability: Chord provides an efficient lookup method even if the system is in a continuous state of change. This property helps Chord to locate requested keys despite the joining of a large number of new nodes.
- Scalability: Scalability is one of the important features in network applications. Chord works well with heavy systems as the cost of a Chord lookup grows only logarithmically to the number of nodes in the system.
- Load balance: Chord uses a consistent hash function to assign keys to nodes. Therefore, the keys are spread evenly over the nodes and no single node is burdened with a large number of keys.

2.3 Formation of Chord Ring:

The Chord protocol uses SHA-1 (or SHA-512 or SHA-256) as a consistent hash function to assign a m-bit identifier to each node and each key. Consistent hash functions are hash functions with some additional advantageous properties, i.e. they let nodes join and leave the system with minimal disruption. The m is an integer which should be chosen big enough to make the probability that two nodes or two keys receive the same identifier negligible. The hash function calculates the key identifier by hashing the key, and the node identifier by hashing the IP address of the node. The key and the node identifiers are arranged on an identifier circle of size 2m called the Chord ring. The identifiers on the Chord ring are numbered from 0 to 2m-1. A key is assigned to a node whose identifier is equal to or greater than the identifier of the key. This node is called the successor node of k, denoted by successor(k), and is the first node clockwise from k on the circle.

2.4 Creation of Finger Table:

For accelerating the lookup time, additional routing information is stored. Each node maintains a table consisting of upto m entries (where m is the number of bits of the identifier) known as finger table. The ith entry in the finger table of the nth node contains the successor node s that is ahead by at least 2i-1. Figure[1] shows the finger table for N8th node where value till N8 + 4 has entry N14th which is the successor node till 2i-1.

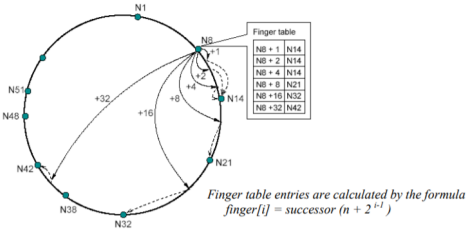

Figure 1: Creation of Finger Table

2.5 Key Location and Hashing:

For a key to be looked up, a random node is selected and if the key is present in the successor node then the next node is returned else the finger table is searched to get the nearest location of the node. In other words, When a node is asked to find a certain key, it will determine the highest predecessor of this key in its routing table and forward the key to that node. This procedure will recursively determine the node responsible for the key. The lookup time is O(log N), since the query is forwarded at least half the remaining distance around the circle in each step.

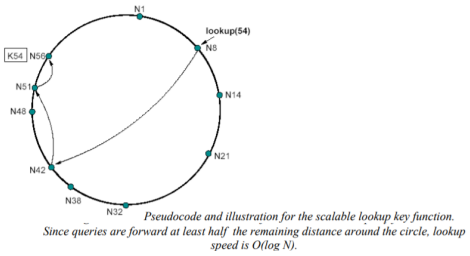

Figure 2: Explanation of locating key node 54

Overview of DistAlgo: DistAlgo is a language for high-level, precise, executable specifications of distributed algorithms, and study its use for specification, implementation, optimization, and simplification of such algorithms. For expressing distributed algorithms at a high level, DistAlgo supports four main concepts by building on an object-oriented programming language, Python: (1) distributed processes that send messages, (2) control flow for handling received messages, (3) high level queries for synchronization conditions, and (4) configuration for setting up and running. Moreover, configuring channels is somewhat straightforward. Three options for channel configuration currently available are fifo, reliable, and (reliable, fifo). If any option is specified, TCP is used for process communication, with UDP being the default.

3 Experimental Setup and Implementation:

3.1 Setup:

We have created the client-server setup using the DistAlgo library in python. First, we are creating the chord ring of n(=50) nodes and generating the hashed value of these n nodes with an m-bit identifier. Second, build the finger table for each node which has upto m entries and has the information for key lookups. Furthermore, the data having website names and IP addresses are fed into the program and corresponding hash values are generated using the same m-bit identifier. These values are distributed among the nodes in the chord ring thus making the chord server up and running. Now when the search query is invoked, the client program queries the server to get the result. If the server has the result present in the successor node then it replies to the client with a token message, then the client responds to the message by saying to get the node and finally the server sends the successor node to the client for obtaining the value. On the other hand, if the server does not have the result, then it consults the finger table and responds with the highest predecessor that can lead to the actual value in the cord ring. In the following sections, we have explained about the experiments done and interesting findings from them.

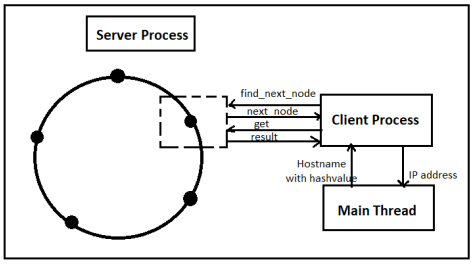

Figure 3: Communication of Client Server in Chord

3.2 Experiment 1: Comparison of Hop Count for finger table size=64,128,256

In this experiment, we are looking to compare the hop counts for different values of finger table size. We have taken three values for m: 64, 128, 256. As we can see from the figure[4], the more the value of the finger table size, the higher the number of hops required. This can be due to the fact that on increasing m, the ring size increases by an exponential factor but the finger table size increases by a linear factor only. Therefore, m=256 requires the highest number of hops.

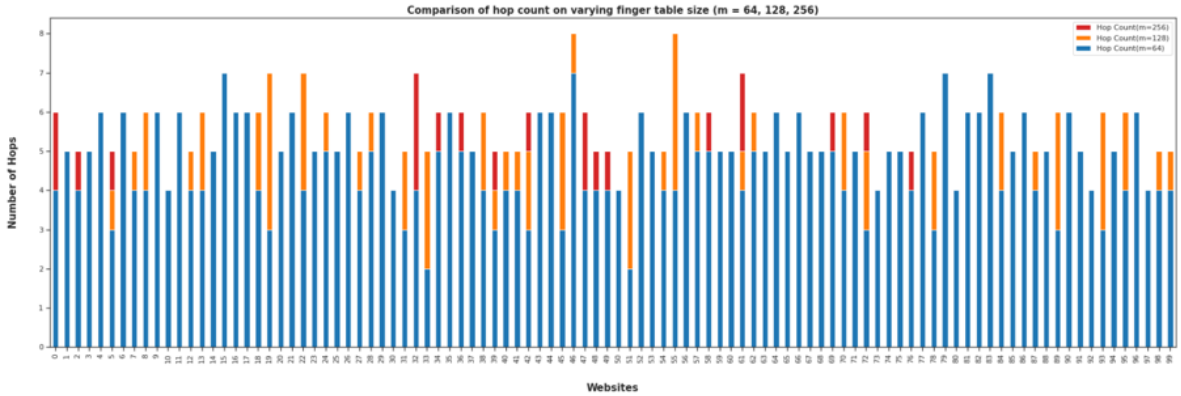

Figure 4: Comparison of Hop Count for 100 websites

3.3 Experiment 2: Comparison of Lookup times for node size = 20,50,100,200

Here, we have varied the number of nodes and compared the lookup times for each case. We found that on increasing the number of nodes, the lookup times also increased. The green line corresponds to node size = 200 and red line (best one) corresponds to node size=50. We observed an interesting thing i.e. when we decreased the node size to 20, then also the lookup times increased marginally. So, from this we can conclude that we require an optimal number of nodes to achieve the best lookup time results.

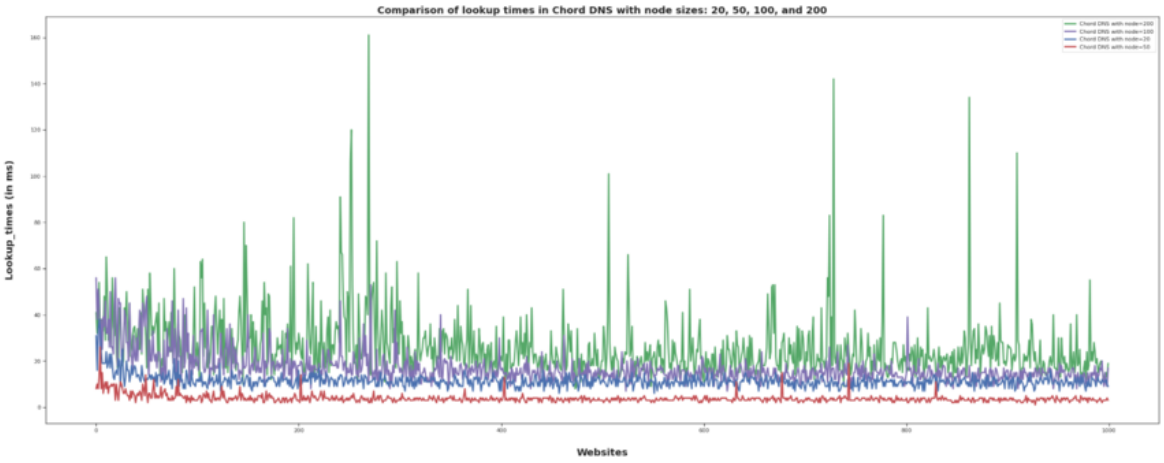

Figure 5: Comparison of Lookup Times for different node sizes(1K websites)

3.4 Experiment 3: Traditional DNS Vs Chord DNS Lookup times

For this experiment, we ran traditional DNS using python library dnspython and Chord on the local servers. We found that DNS takes quite a lot of time to return results whereas Chord is processing results quickly. This can be due to the fact that DNS suffers from different kinds of latency delays like system delay, client delay, server delay. To simulate a real network for Chord, we added time delays whenever a client node sends a request to the server and vice versa. According to the visualization[6,7], we can see that the Chord DNS actually has a better query time than traditional DNS. This can be attributed to the fact that Chord is a peer to peer distributed network whereas traditional DNS uses a hierarchical structure. It is also dependent on the number of hop counts. This is because, as the number of hops increases, the node latencies will occur and Chord DNS has a lesser number of hop counts as compared to traditional DNS. One factor that we may consider is the total number of websites that we are querying. As the number of websites increases, the load will increase on each node which will affect the performance of Chord as well. At that time, the query times for traditional DNS and Chord DNS will become comparable. Another observation is about the high spikes in the graph, this can be due to the fact that some of the websites are less commonly used than the others, which leads to higher query time for those websites.

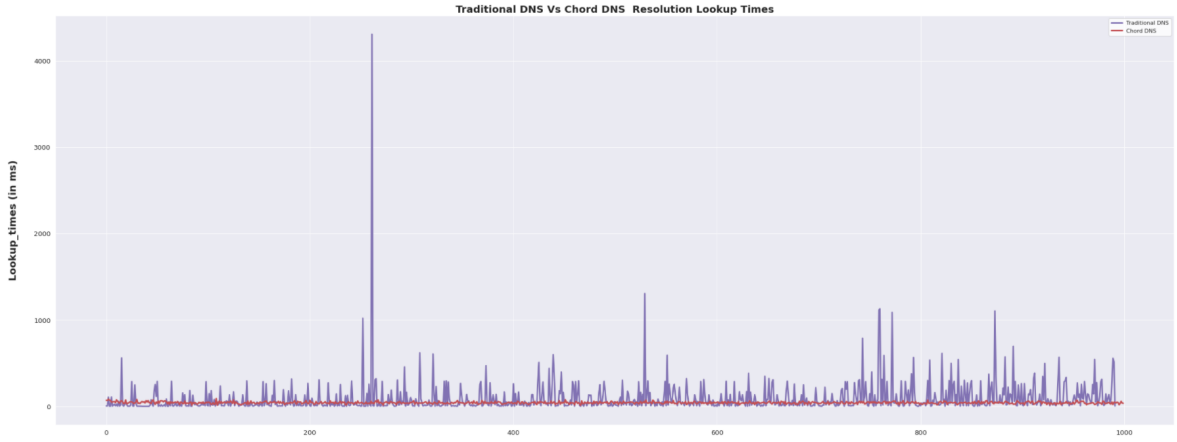

Figure 6: Traditional DNS Vs Chord DNS lookup times(1K websites)

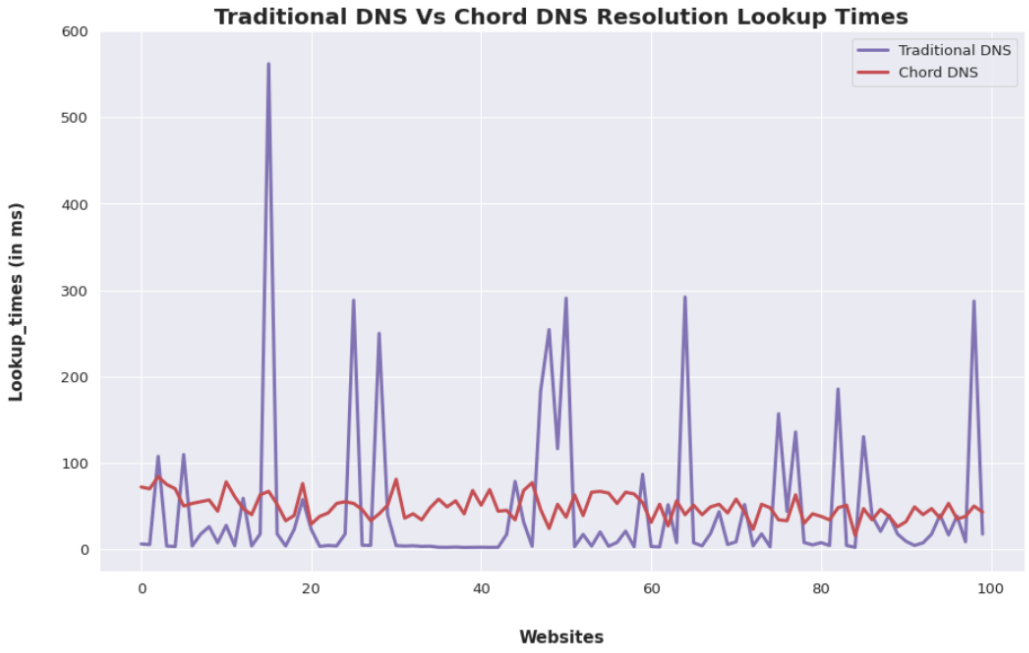

Figure 7: Traditional DNS Vs Chord DNS lookup times(100 websites)

3.5 Experiment 4: Traditional DNS Vs Chord DNS hop count comparison

For this experiment, we ran the traceroute command on our local system to fetch the entire resolution routes for each website. The last row will give us the total number of hops required for a particular website. Similarly, we ran Chord DNS and extracted the hop count values. On plotting the count for both the methods, we can see that the hop count is more in traditional DNS as compared to Chord DNS. The significant increase in hop counts for traditional DNS can be due to the fact that it depends on hierarchical structure rather than peer to peer distributed network and it will have to connect to the root server for each website again and again, thus increasing the count of overall hops.

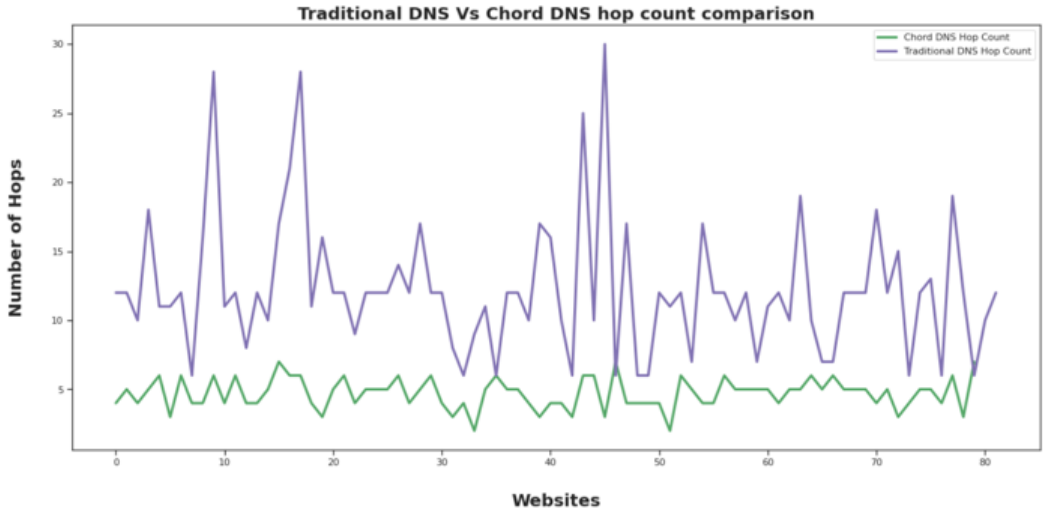

Figure 8: Traditional DNS Vs Chord DNS hop count(100 websites)

3.6 Experiment 5: Comparison of Lookup times for finger table size: 64, 128

Here, we have increased the finger table size from 64 to 128 to check the impact on lookup times. As can be seen from this graph, lookup time was more for m=64(blue line) initially, but as the number of websites increased, the lookup times for both values starts to flatten out and are comparable with each other. Similarly, we did this experiment for 1000 websites and we observed same kind of graphs for that as well.

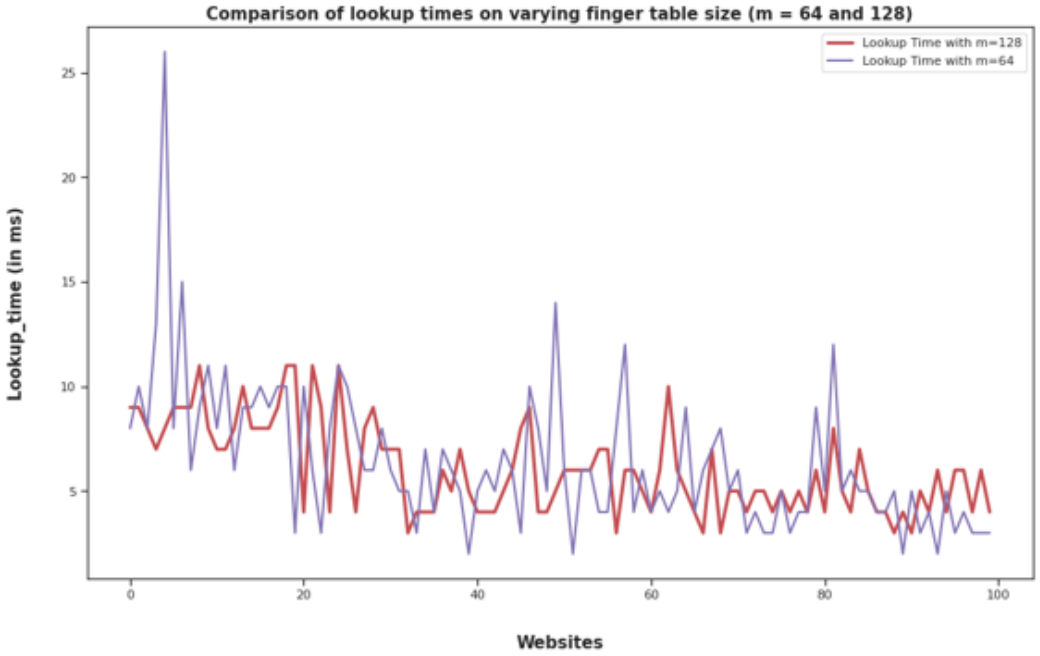

Figure 9: Comparison of Lookup Times for different finger table sizes(100 websites)

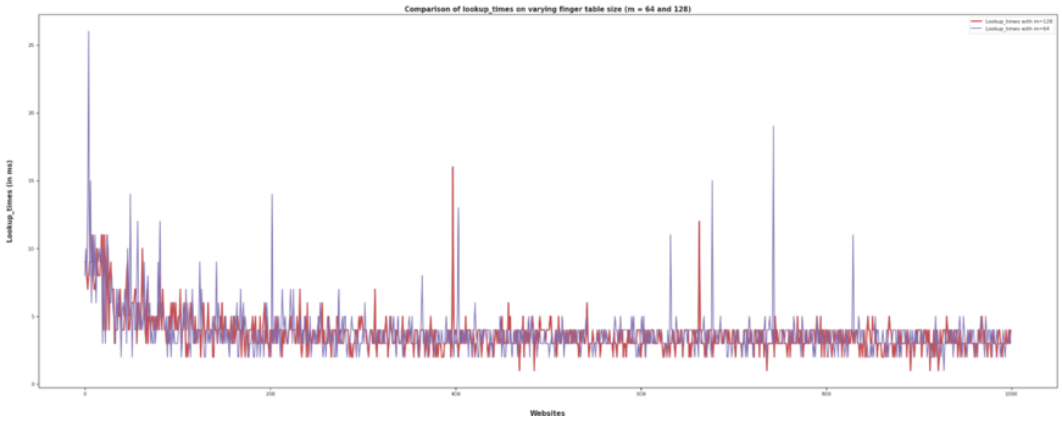

Figure 10: Comparison of Lookup Times for different finger table sizes(1K websites)

4 Conclusion:

In this project, we have witnessed an interesting comparison study between traditional DNS and Chord DNS. First, we learnt about Chord architecture and how it creates its core components to form a distributed network of nodes. Then we did several experiments to compare and understand if Chord performs better or not and under what conditions. This also includes the performance of Chord DNS on varying different measures such as finger table size, number of nodes. Based on these experiments, we found that Chord DNS does a good job with respect to query lookup times when the number of hop counts are low. As we increase the number of websites, it increases the hop count as well as query lookup times. In such cases, the performance of traditional DNS seems to be comparable (and even better in some cases) as compared to Chord DNS. With respect to the performance of Chord for different internal measures such as number of nodes, size of finger table, we can conclude with two observations. First, if we increase the number of nodes, then the query lookup times increases and second, if we increase the finger table size, then the number of hop counts increases too. As noted in the previous section, we need to find an optimal combination of finger table size and number of nodes to achieve the maximal efficiency for Chord DNS. Additionally, Chord definitely lowers the dependency on root serves and distributes the load equally among nodes which leads to better load balancing. So, we can say that our initial hypothesis is partially correct i.e. Chord DNS works better than traditional DNS, given the corresponding measures are optimal.

5 Future Scope:

With respect to the future scope, this project can be extended to create a Chord system that caches the results of a particular website based on a threshold value which tells us how many websites can be cached at a particular node. This can make search more efficient and will further decrease lookup time. Another enhancement can be to include nodes dynamically in the Chord ring and perform stabilization of the finger table after a fixed time interval. This will reduce node lookup failures and would enhance fault tolerance capabilities of Chord. Finally, we can try to integrate the best features of other state of the art implementations of distributed networks used for resolving DNS such as Pastry, Globe, Ohaha to form a generic

resolver capable of resolving domain names dynamically at a fast moving pace with an added layer of security.

References

[1] Stoica, Ion, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. "Chord: a scalable peer-to-peer lookup protocol for internet applications." IEEE/ACM Transactions on networking 11, no. 1 (2003): 17-32.

[2] Sit, Emil, and Robert Morris. "Security considerations for peer-to-peer distributed hash tables." In International Workshop on Peer-to-Peer Systems, pp. 261-269. Springer, Berlin, Heidelberg, 2002.

[3] Stoller, Scott D., and Yanhong A. Liu. "Algorithm Diversity for Resilient Systems." In IFIP Annual Conference on Data and Applications Security and Privacy, pp. 359-378. Springer, Cham, 2019.

[4] Morris, Robert, M. Frans Kaashoek, David Karger, Hari Balakrishnan, Ion Stoica, David Liben-Nowell, and Frank Dabek. "Chord: A scalable peer-to-peer look-up protocol for internet applications." IEEE/ACM Transactions On Networking 11, no. 1 (2003): 17-32.

[5] Doi, Yusuke. "DNS meets DHT: Treating massive ID resolution using DNS over DHT." In The 2005 Symposium on Applications and the Internet, pp. 9-15. IEEE, 2005.

[6] Stoica, Ion, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. "Chord." In Proceedings 2001 Conf. Applications, Technol., Architectures, Protocols Comput. Commun.-SIGCOMM'01, pp. 149-160. 2003.

[7] Song, Yiting, and Keiichi Koyanagi. "Study on a hybrid P2P based DNS." In 2011 IEEE International Conference on Computer Science and Automation Engineering, vol. 4, pp. 152-155. IEEE, 2011.

[8] Doyen, Guillaume, Emmanuel Nataf, and Olivier Festor. "A performance-oriented management information model for the Chord peer-to-peer framework." In IFIP/IEEE International Conference on Management of Multimedia Networks and Services, pp. 200-212. Springer, Berlin, Heidelberg, 2004.