Summer 2014

Privacy preservation using spherical chord

Doyal Tapan Mukherjee

Department:

PRIVACY PRESERVATION

USING SPHERICAL CHORD

by

DOYAL TAPAN MUKHERJEE

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2014

Approved by

Dr. Sriram Chellappan, Advisor
Dr. Bruce McMillin
Dr. Frank Liu

ABSTRACT

Structured overlay networks are primarily used in data storage and data lookup, but they are vulnerable against many kinds of attacks. Within the realm of security, overlay networks have demonstrated applicability in providing privacy, availability, integrity, along with scalability. The thesis first analyses the Chord and the SALSA protocols which are organized in structured overlays to provide data with a certain degree of privacy, and then defines a new protocol called Spherical Chord which provides data lookup with privacy, while also being scalable, and addresses critical existing weaknesses in Chord and SALSA protocols. Spherical Chord is a variant of the Chord, and utilizes the concept of distributed hash table (DHT). Chord sends packets uni-directionally over a virtual id space in the overlay. While this feature provides lower latencies, it can be used by attackers to misroute and drop packets. Spherical Chord protocol introduces additional connections in the structured overlay and increases the path length and the number of paths for sending messages, hence making it more resilient to routing attacks. A new protocol focusing for constructing the Spherical Chord, followed by a new lookup protocol is defined in this thesis. The protocols are analyzed and it is demonstrated using both theoretical analysis and simulations that improved path availability helps in maintaining privacy, while also limiting the impact of routing attacks.

ACKNOWLEDGMENTS

I have to thank my research advisor, Dr. Sriram Chellappan. Without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding over these past two years.

Secondly I would like to express my gratitude to Dr. Bruce McMillin and Dr. Frank Liu for serving in my thesis committee and for taking their time to review my work. I would also like to thank the Intelligent Systems Center (Missouri University of Science and Technology) and the National Science Foundation (NSF) for providing financial support for my research.

Lastly I would like to thank my parents Mr. T. D. Mukherjee and Mrs. Nupur Mukherjee and my sister Payal Mukherjee who have given me unconditional support and encouragement.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENTS iv

LIST OF ILLUSTRATIONS vii

LIST OF TABLES viii

SECTION

1. INTRODUCTION 1

2. RELATED CONCEPTS 3

2.1 OVERLAY NETWORK 3

2.2 DISTRIBUTED HASH TABLE (DHT) 3

2.3 CONSISTENT HASHING 4

3. THE CHORD LOOKUP PROTOCOL 5

3.1 CHORD IDENTIFIER CIRCLE 5

3.2 SCABALE NODE LOOKUP 6

3.2.1. Finger Table 6

3.2.2. Lookup Protocol 7

3.3 PRIVACY IN CHORD 9

4. THE SALSA PROTOCOL 10

4.1 LOOKUP PROCEDURE 10

5. SPHERICAL CHORD 12

5.1 ATTACK MODEL 12

5.2 CONSTRUCTION OF SPHERICAL CHORD 13

5.3 LOOKUP IN SPHERICAL CHORD 14

5.4 FINGER TABLE MODIFICATION IN SPHERICAL CHORD 17

5.5 STABILIZATION ALGORITHM 18

5.6 OVERHEAD IN SPHERICAL CHORD 20

6. ANALYSIS OF SPHERICAL CHORD 22

6.1 PATH LENGTH ANALYSIS OF SPHERICAL CHORD 22

6.2 PATH AVAILABILITY METRIC OF SPHERICAL CHORD 22

6.3 LOOKUP FAILURE ANALYSIS OF SPHERICAL CHORD 24
7. CONCLUSIONS 25
BIBLIOGRAPHY 26
VITA. 28

LIST OF ILLUSTRATIONS

Figure Page

3.1. An identifier circle with m=6, k=5 and n=10 5

3.2. Finger Table for node 8..... 7

3.3. Lookup operations for Key (54). 8

4.1. Binary tree structure of SALSA, where G0 to G7 are groups 11

5.1. Spherical Chord Structure. (*N=512*). Interlocking Nodes are from 0 to 9. 13

5.2. Spherical Chord Structure. (*N=512*). Interlocking Nodes are from 0 to 9. Lookup is for interlocking nodes 15

5.3. Spherical Chord Structure. (*N=512*). Interlocking Nodes are from 0 to 9. Lookup is for non-interlocking nodes 16

5.4. Overhead in Spherical Chord..... 21

6.1. Path Length Analysis 22

6.2. Path Availability Metric..... 23

6.3. Lookup Failure Analysis of Spherical Chord 24

LIST OF TABLES

Table Page

3.1. Definition of the finger table..... 6

5.1. Definition of the Spherical Chord finger table ..... .17

5.2. Interlocking Definition table..... 18

1. INTRODUCTION

Structured overlay peer to peer networks have found wide applicability today in distributed content management [8]. Numerous protocols have been designed for overlay networks in the realm of creation of nodes, organization of nodes and lookup of a particular node [4, 7, 11, 16]. The structure of the network and protocol should be designed such that there are no single points of failures, and there is good load balance among all nodes during network operation. Beyond just content lookup, other critical requirements of structured overlays today include scalability, privacy and integrity. In this thesis, we define a new type of structured overlay peer to peer network called Spherical Chord which is a look up service that provides a high degree of availability, scalability, privacy and integrity, while being resilient to a number of malicious attacks.

During communication over a network, unless security and privacy mechanisms are in place, the browser typically advertises the IP address, the domain, platform and the information which is requested. Such critical data could be out on the network and can be easily monitored and mishandled by attackers. Thus, privacy on the network can be compromised. Overlay peer to peer networks provide elegant solutions to protect privacy on the network. The nature of peer to peer communication means that typically, it is harder for an adversary to identify which nodes are sources and destinations of messages, and which nodes are mere forwarders.

Existing overlay networks like Tor [4] and Tarzan [3, 11] encrypt data across servers and create anonymous circuits. These systems rely on the communication of messages among a small set of randomly selected forwarders. However, when the number of nodes increases, the systems faces scalability issues from overhead and maintenance. Furthermore, these protocols are unstructured in nature, in the sense that there is a high degree of randomness in nodes choosing forwarders. This feature hence compromises availability. To circumvent this problem, structured overlays were introduced wherein the nodes in the overlay are arranged following a well-defined structure that is appropriately leveraged during routing. The presence of a well-defined structure improves availability,

while the presence of multiple intermediate forwarders to route messages improves privacy. Two of the well-known protocols in this realm are Chord [7] and SALSA (Structured Approach to Large Scale Anonymity) [16]. In Chord, nodes are arranged in a circular space with virtual identities, and routing proceeds in the circle uni-directionally during look-ups. SALSA is another protocol wherein nodes are arranged in a binary tree structure and messages are routed along the tree in a well-defined manner in reaching the correct destination. Both systems typically route a message in *log (N)* hops, where *N* is the number of nodes in the network. Unfortunately though, both systems suffer from poor performance in the presence of attacks that severely degrade privacy and availability, as we discuss subsequently.

In this thesis, we define a new kind of structured overlay network called Spherical Chord, which is a look up service. However, compared to the existing Chord and SALSA protocols, it provides multiple paths to route a message in the overlay, while still retaining the advantages of maintaining a structure during routing. As we demonstrate subsequently, Spherical Chord provides a high degree of availability, scalability, privacy and integrity, while being resilient to a number of malicious attacks.

2. RELATED CONCEPTS

2.1. OVERLAY NETWORK

An overlay network is defined as a separate “logical” network on top of the physical network. Overlay networks can be structured or unstructured. In structured overlay networks, the construction and maintenance of the network follows a well-defined structure, while in unstructured networks, there is a high degree of randomness during construction and maintenance. Overlay networks can also be centralized with some nodes performing more duties that other nodes or they can also be decentralized and distributed, wherein there is a very high degree of load balance along all nodes. Overlay networks like Napster [19] and Gnutella [18] are unstructured and centralized, while networks like are CAN [14], SALSA [16] and Chord [7] are structured and decentralized. All of these protocols provide lookup services, and each has its own strengths and weaknesses from lookup perspectives. More recently though, the issue of security and privacy are assuming critical significance in overlay networks. In overlay networks, since there are typically a large number of nodes, controlling access to only honest nodes become impossible. How to provide routing services in the presence of malicious nodes, while still providing a high degree of availability and scalability is hence a major challenge today. With the large number of nodes in the overlay, and in the presence of heavy network traffic, recent studies are also exploring the applicability of overlay networks for providing privacy preserved communications as well [7, 16].

2.2. DISTRIBUTED HASH TABLE (DHT)

Distributed Hash Tables are a generalized concept, wherein keys are distributed to nodes in a manner that attempts to distribute the assignments of keys to nodes in a uniform manner [9]. Any participating node in the DHT can now efficiently retrieve the value associated with any given key. DHTs also ensure load balancing in the sense that the responsibility for maintaining the mapping from keys to values is now purely distributed among the nodes. This also ensures that a change in the set of participants (when nodes join and leave) causes a minimal amount of disruption, hence enhancing

scalability. DHTs recently have been used in a number of applications including, cooperative Web caching, distributed file systems, domain name services, instant messaging, multicast, and also peer-to-peer file sharing and content distribution systems. While there are multiple techniques that can be used to generate DHTs, consistent hashing is one of the standard approaches, and is discussed next.

2.3. CONSISTENT HASHING

In consistent hashing both the node and the key are typically hashed using the same hash function, and is widely used for designing DHTs. In addition to balancing assignment of nodes to keys, consistent hashing provides significant benefits to maintenance of the network during the node join and node delete operations.

Consistent hashing maps the nodes and keys to a randomly distributed, "identifier points" on the circle. After hashing the system needs to find where a key should be assigned. Each node is responsible for the key which precedes itself in the identifier space. The result is that each node contains all the keys located between its identifier point and the previous node's identifier point.

If a node leaves the network then the requests for the key associated with it would be mapped to the next highest point in the identifier space. Since each node is associated with many pseudo-randomly distributed identifier points, the keys that were held by that node will now be map to different nodes.

A similar process occurs when a node is added. By adding a node, we assign keys to the new node which is succeeding the key in the identifier space. These keys will no longer be associated with the previous node, and any value previously stored there will not be found by the lookup method. The lookup request would now be directed to new node responsible for a particular key [9, 10].

With this design, if a node joins or leaves the network then only *O (K/N)* nodes are responsible for making changes to their routing tables where *K* is the number of keys and *N* is the number of nodes in the network. Consequently in a network with *N* number of nodes and *K* keys, each node is responsible for at most (*K/N*) keys [8].

3. THE CHORD LOOKUP PROTOCOL

In this section, we first illustrate the Chord protocol for scalable lookup services in large scale networks, and a discussion of privacy features in Chord.

3.1. CHORD IDENTIFIER CIRCLE

The Chord protocol uses the concept of DHT and consistent hashing for efficient lookup operation. In the Chord protocol, the keys are assigned to the node in the following way. An *identifier circle* is created and the identifiers are placed in a 2^m *modulo structure circle.* Key *k* is assigned to a node whose identifier is equal to or succeeds *k* in the identifier space. This is done using the function *successor (k),* also called as the successor node. In totality the nodes are arranged in a circular fashion from *0* to 2^m*-1*. The successor node of key *k* is the first node that is followed by *k* in a clockwise direction [7]. If a node *n* joins the network, then all the keys assigned to its successor would be now assigned to node *n.* If a node *n* leaves the network, then the successor of node *n* would be responsible for all the keys. Thus, using consistent hashing only a fraction of nodes would have the responsibility of updating the routing tables during subsequent node join and delete operations. Figure 3.1 shows an example of an identifier circle used in the Chord protocol.

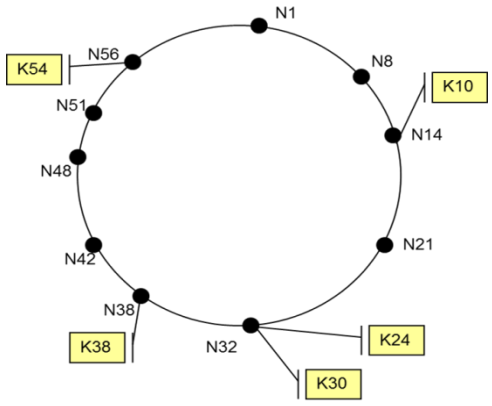

Figure 3.1. An identifier circle with m=6, k=5 and n=10

Figure 3.1 signifies an identifier circle with m=6. It therefore has total of 2^m identifiers. The identifiers are from 0 to a maximum of 63 (2^m-1). The number of nodes that are currently active in the overlay are shown in dots, and they are ten in number. The number of keys are 5.

3.2. SCALABLE NODE LOOKUP

3.2.1. Finger Table. To increase the efficiency of simple node lookup, the Chord protocol creates additional information for efficient lookup of a node. This information is stored in the *finger table.* If *m* is the number of bits in the identifier, then each node *n* maintains at least *m* entries in the finger table [7]. The i^{th} entry in the table at node *n* contains the identity of the first node, *s*, that succeeds *n* by at least 2^{i-1} on the identifier circle, i.e., *s=successor (n* $+2^{i-1}$*))*, where *1* $\leq i \leq m$ (and all arithmetic is modulo 2^m). The node *s* is called the i^{th} finger of node *n* and denotes it by *n.finger[i].node*. The definition of the finger table is given as follows,

Table 3.1. Definition of the finger table

Notation	Definition
finger[k].start	*(n + 2^{k-1}) mod 2^m*
.interval	*(finger[k].start, finger[k+1].start)*
.node	*First node >= n.finger[k].start*
Successor	*The next node on the identifier circle; finger[1].node*
Predecessor	*The previous node on the identifier circle*

3.2.2. Lookup Protocol. We now present the Chord protocol for looking up a key.

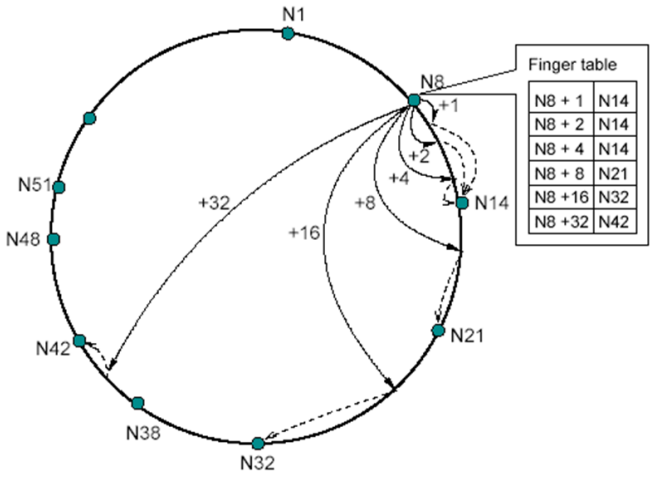

Figure 3.2. Finger Table for node 8.

The above figure shows the finger table for the propagation of a request originating from node N_8. The i^{th} entry of the finger table is given as *finger[i] = successor* $(n + 2^{i-1})$. In the scalable node localization, each node stores information of only a maximum for *log (N)* nodes. For a particular lookup operation the entry of the finger table is checked and propagated on basis of the pointers to the successor nodes. When a key is looked up, the Chord protocol will route the message to that node in its finger table, which is closest to the destination node in the virtual id space. Each node that receives the message will repeat the process until the destination node that stores the key is reached. The algorithm is as follows,

```
//ask node n to find the key (k)
n.find_successor(k)
 if (k in (n,successor])
  return successor;
 else
  n‘ = closest_preceding_node(k);
  return n‘.find_successor(k);
//search the local table for the highest predecessor of k
n.closest_preceding_node(k);
  for i=m downto 1 do
    if (finger[i] in (n,k))
      return finger[i];
   return n;
```

With this algorithm, a lookup in Chord will take *O (log N)* messages to find a destination, where *N* is the number of nodes in the system.

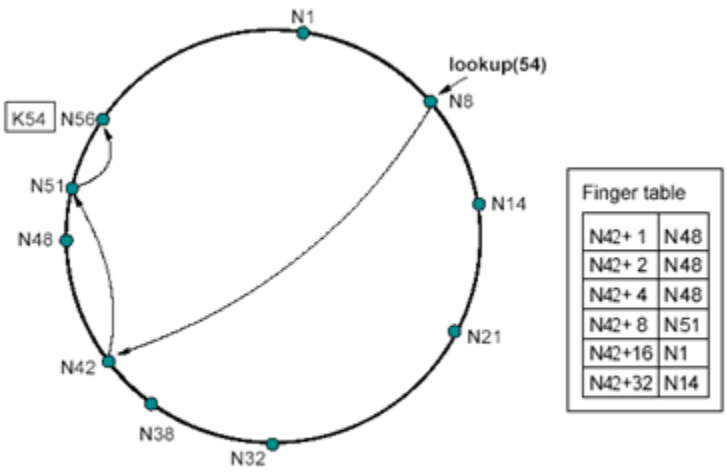

Figure 3.3. Lookup operations for Key (54).

Figure 3.3 shows an example of a lookup operation where K_{54} is looked up by node N_8. N_{56} stores the key K_{54} and thus is the target node. The figure demonstrates that using the finger table for node N_8, a long hop will be taken to node N_{42} as it is the node which is closest to the target node N_{56}. Consequently smaller hops are taken and the destination N_{56} is reached.

3.3. PRIVACY IN CHORD

Although not intended, Chord does provide a certain degree of privacy due to its inherent nature of using forwarder nodes to route messages. Privacy in Chord protocol is closely related to the method in which a particular request is routed. In the Chord protocol, the lookup requests are forwarded among the nodes. On account of this request propagation the identity of the requestor node or the target node is not revealed as they will appear to be no different from the initiator or target of a message in the virtual identifier space during routing [1].

The Chord structure however exposes a weakness when the presence of malicious nodes increases in the network. In the Chord protocol the requests from any particular initiator to a particular target go through the same set of nodes, which are then in a position to modify more than one request if they are malicious [12]. Furthermore requests from multiple initiators are likely to converge on nodes closer to the target in the virtual identifier space, and if some of these are malicious, then any request to that particular target will all be dropped, hence severely limiting Chord's ability to provide availability to the nodes [2, 13]. As mentioned earlier, the problem is exacerbated due to uni-directional routing in Chord.

4. THE SALSA PROTOCOL

The SALSA (Structured Approach to Large Scale Anonymity) network architecture also uses the distributed hash table (DHT) and consistent hashing The SALSA architecture organizes the node in a "Chord" like identifier space, but employs a binary tree like structure to organize nodes into groups. The node is said to belong in a group if it is in that group's identifier space [16]. On basis of this concept the SALSA protocol divides the network into local and global contacts. Each node has a local contact table where it stores the ID of itself and other nodes in the same group. Similarly each node has a global contact table where it stores a single contact for each level of the binary tree [24].

4.1. LOOKUP PROCEDURE

Let us consider an initiator node, *Q*, which looks for a target key. The query node *Q* selects *R* neighboring local contacts which fall under the same group. Node *Q* requests each of the nodes in that particular group to find the owner of the target key. Each node in the neighboring group of *R* nodes independently finds the owner of the key in the SALSA virtual binary tree. The *R* neighboring nodes send the location of the target node back to the query node *Q*. Now query node *Q* calculates the distance from itself to the target and chooses one with the minimum distance (in the event of malicious lookup returns). If the target node is within predefined distance bounds, it forms an anonymous circuit to the target node. Figure 4.1 shows a typical binary tree structure of SALSA.

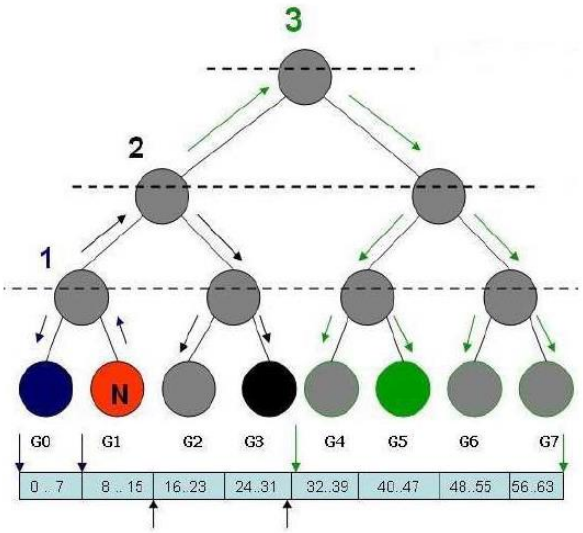

Figure 4.1. Binary tree structure of SALSA, where G0 to G7 are groups

SALSA lookups the message and performs better than Chord on account of sending redundant messages across the groups. Thus even if some nodes are malicious and drop requests, there are alternate good nodes to route the message successfully. The analytical model of SALSA shows that the probability to find the lookup initiator can be calculated as follows

$$1 - (1 - f)^{l}. \tag{1}$$

In the above expression, *f* is the fraction of malicious nodes in the network and *l* is given as the number of redundant lookups. This expression suggests when the number of nodes in the system increases, then the redundant messages also increases thereby increasing the probability of locating the initiator [2]. Furthermore the redundant lookup requests also travel through mostly same set of paths to reach the target node. If there are malicious nodes along that path then probability of locating the initiator also increases.

5. SPHERICAL CHORD

In this section, we present a new protocol that extends the Chord protocol, while providing a much higher degree of privacy and availability compared to Chord and SALSA. This protocol consists of many Chord rings to form a spherical structure and thus, denoted as *Spherical Chord.*

5.1. ATTACK MODEL

To evaluate the privacy and availability of the Spherical Chord protocol, we have defined an attack model to analyze the system and demonstrate its properties. The attacks that are considered are as follows,

- Dropping Lookup Requests: This is a denial of service attack. In this type of attack, when a malicious node receives a packet for a particular request, it will drop that particular packet [5].
- Randomly Misrouting Packets: In this type of attack the malicious node does not drop the packet but simply routes the lookup requests to a different node randomly. By doing this, the malicious node makes the lookup query to take a different path so that it never actually reaches the destination, or reaches it quite late after too many hops. This kind of attack is difficult to detect, since the requests per se are not dropped, but rather re-routed, which may happen under node dynamics like joins and leaves [6].
- Performing a Sub-ring Attack: This type of attack is salient to Spherical Chord, and itself is a novel contribution. In this attack, groups of malicious nodes collaborate so that the lookup request does not reach the destination. Here the attackers have two types of finger tables - one which is correct, and the other which contains the successor node entries of the malicious nodes. When the lookup request is received, the malicious node will use its finger table and make the lookup request propagate across the malicious nodes and it will finally reach a malicious destination. Here the malicious nodes collaborate and give an impression that they are propagating the request, but ultimately it will give us an incorrect target destination [6, 15].

5.2. CONSTRUCTION OF SPHERICAL CHORD

In Spherical Chord, multiple rings are constructed to place virtual identifiers in the overlay space, and rings are connected via carefully chosen interlocking nodes. Messages can traverse in the Spherical Chord within the ring and across rings by traversing through interlocking nodes. With this arrangement, nodes have multiple routes to send messages, hence improving availability, while the nature of using intermediate forwarders provides a high degree of privacy.

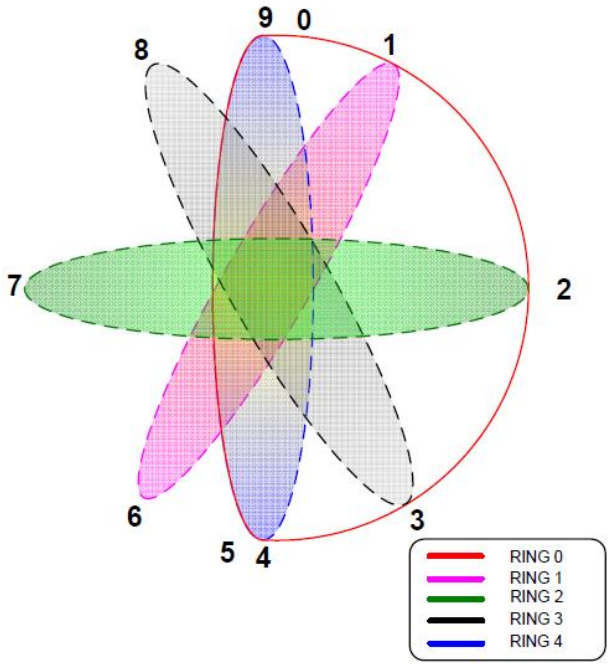

Figure 5.1. Spherical Chord Structure. (*N=512*). Interlocking nodes are from 0 to 9.

If there are *N* nodes in the network then the *r* rings which will be formed for communication in Spherical Chord, is defined as

$$r = ((\log N)/2) + 1 \quad (2)$$

The identities of the interlocking nodes in each ring are as follows,

$$0 \text{ and } ((\log N)/2) + 1 \quad \text{(Ring 0)}$$
$$1 \text{ and } ((\log N)/2) + 2 \quad \text{(Ring 1)}$$
$$\vdots \qquad\qquad \vdots$$
$$((\log N)/2) \text{ and } \log N \quad \text{(Ring } r\text{)}$$

Figure 5.1 shows the left hand side cross section of the spherical structure. Here each ring is highlighted with a different color.

The rings and the interlocking nodes in Figure 5.1 are as follows

- Ring 0 has interlocking nodes 0 and 5.
- Ring 1 has interlocking nodes 1 and 6.
- Ring 2 has interlocking nodes 2 and 7.
- Ring 3 has interlocking nodes 3 and 8.
- Ring 4 has interlocking nodes 4 and 9.

This configuration is based on equation 2. Since the number of rings increases with increase in the number of nodes, the path for a particular lookup also increases and privacy and availability will be maintained. This is further explained in the following sections.

5.3. LOOKUP IN SPHERICAL CHORD

When a lookup is initiated, the number of hops required to communicate with a node is related to the interlocking nodes and also the non-interlocking nodes. If a node in one ring wants to communicate with other ring's interlocking node, it takes a maximum of *2*O (log N)* hops. If node in one ring wants to communicate with another ring's node, the number of hops required would be *3*O (log N),* where *N* is the number of nodes in the network.

As demonstrated in Figure 5.2, if a node marked in red wants to communicate with an interlocking node, it has to communicate to the outer ring and there are two paths of communication available corresponding to the number of rings. Thus, for this particular lookup operation the path length is given as *2*O (log N).* Figure 5.2 illustrates that if a node in ring 1 wants to communicate with the interlocking node of ring 3, then it has to first communicate to the outer ring. There are two paths available for the lookup. The first being, the node can look up to node 1 for the outer ring and then communicate to node 3. The other path will be, the node looks up to node 6 and then to node 3. Thus the path length for a lookup to an interlocking node is *2*O (log N).*

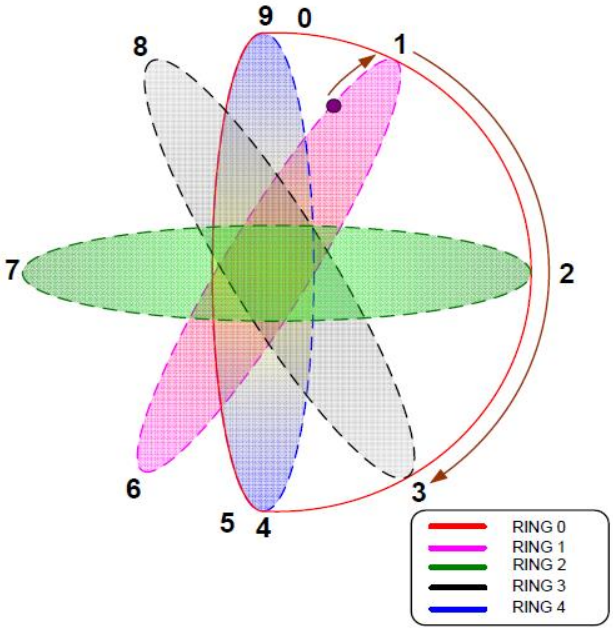

Figure 5.2. Spherical Chord Structure. (*N=512)*. Interlocking nodes are from 0 to 9. Lookup is for interlocking nodes

The most frequent case is when nodes in one ring would communicate with nodes in another ring. If the other node is not an interlocking node then the path length increases. As presented in Figure 5.3 If a node in ring 1 wants to communicate with a node in the ring 2 which is a non-interlocking node, then the number of paths required would be *3*O (log N)* hops. The node would first communicate to ring 0 through the interlocking node 1. Through this ring 0, it would find the interlocking node through which it can communicate to the other ring, which in our case is node 2. For that interlocking node the lookup request is initiated to the target node. Thus for this lookup operation the path availability increases, and the path length for communication along non - interlocking nodes is *3*O (log N).*

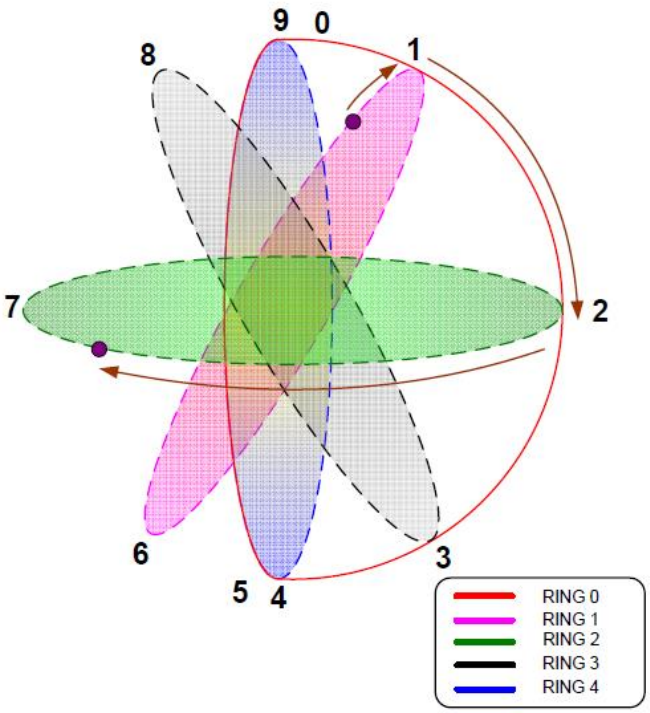

Figure 5.3. Spherical Chord Structure (*N=512*). Interlocking nodes are from 0 to 9. Lookup is for non-interlocking nodes

5.4. FINGER TABLE MODIFICATION IN SPHERICAL CHORD

The new protocol has significant modifications to the traditional Chord finger table. We have included the flags for the interlocking nodes, as they are the medium of communication with the other rings. Table 5.1 shows the definition of the new Spherical Chord finger table. In this table the *interlockingFlag* will be updated if the node acts as an interlocking node. The protocol also has a *ringNumber* associated with the nodes, which acts as a reference to the interlocking definition table.

Table 5.1. Definition of the Spherical Chord finger table

Notation	Definition
finger[k].start	$(n + 2^{k-1}) \bmod 2^m$
.interval	*(finger[k].start, finger[k+1].start)*
.node	*First node >= n.finger[k].start*
Successor	*The next node on the identifier circle; finger[1].node*
Predecessor	*The previous node on the identifier circle*
interlockingFlag	*true or false(Boolean)*
ringNumber	*Ring number of the node*

The table 5.2 is the interlocking definition table developed for Spherical Chord. It consists of *ringNumber*, which signifies the number of the ring to which the node belongs. The *nodesList* mentioned in the table is the list of all the nodes which act as the interlocking nodes associated with that particular *ringNumber*. This table will be primarily used when running the stabilization algorithm for node join and delete operations.

Table 5.2. Interlocking Definition table.

Notation	Definition
ringNumber	*The number of the ring to which the node belongs*
nodesList	*List containing the nodes as the interlocking nodes.*

5.5. STABILIZATION ALGORITHM

Nodes can join and leave the system. Our stabilization algorithm for Spherical Chord focuses on the interlocking nodes as they are basis of communication among Chord rings. For example, if an interlocking node joins the system, then the node sets its *interlockingFlag* as true in Spherical Chord finger table. This new node should also be added to the list containing the interlocking nodes in the interlocking definition table.

We develop the stabilization algorithm which keeps the pointers for the successor nodes and predecessor nodes updated, along with proper configuration of the interlocking nodes. The pointers to the nodes and the interlocking nodes should be kept updated in the Spherical Chord finger table as well as the interlocking definition table.

The stabilization algorithm which specifically works during node join and delete operations is as follows,

```
n.join (n')
    predecessor=nill;
   successor=n'.find_successor (n);
//obtain the node's successor
n.stabilize();
    x=successor.predecessor;
   if (x ε (n, successor))
      successor=x;
successor.notify(n);
//if n is the interlocking node
    x=successor.predecessor;
   if (x ε (n, nodeList))
      successor=x;
    successor.interlockingFlag=true;
successor.notify(n);
successor.fixInterlockingTable();
//calculate the ring number
n.calculateRingNumber();
//n' thinks it might be a predecessor
n.notify(n');
  if( predecessor is nill or n' ε(predecessor,n))
       predecessor=n';
//periodically fix fingers in finger table
//periodically fix the interlocking definition table
n.fix_fingers();
n.fixInterlockingTable();
```

When the stabilization algorithm successfully executes, a node will successfully resolve a lookup request even during node join and delete operations as the pointers to the nodes would be updated appropriately. Additionally, the interlocking definition table would contain the correct list of interlocking nodes, which form the basis of communication among different rings.

5.6. OVERHEAD IN SPHERICAL CHORD

The Spherical Chord runs a stabilization protocol and notifies both the finger table and interlocking definition table of any update during node join and delete operations. The additional computation required to notify the interlocking definition table increases the overhead of the Spherical Chord protocol. This overhead is further increased with an increase in the number of failed nodes in the network.

Let us consider a system of *N* nodes. The average path length for Chord is *logN* and for Spherical Chord it is a maximum of *3*logN*. Let *k* be the number of failed nodes in the system, then the probability of the failed nodes on the path length is given as follows,

$$\text{Failed Node on a Path} = \frac{\text{Avg. Path Length} * \text{Failed Nodes}}{\text{Number of Nodes}} \tag{3}$$

In our analysis, we have considered 1000 nodes in the system. Therefore, the number of failed nodes on the path length in Spherical Chord is greater than the number of failed nodes in the traditional Chord protocol. The graphical representation is shown in Figure 5.4.

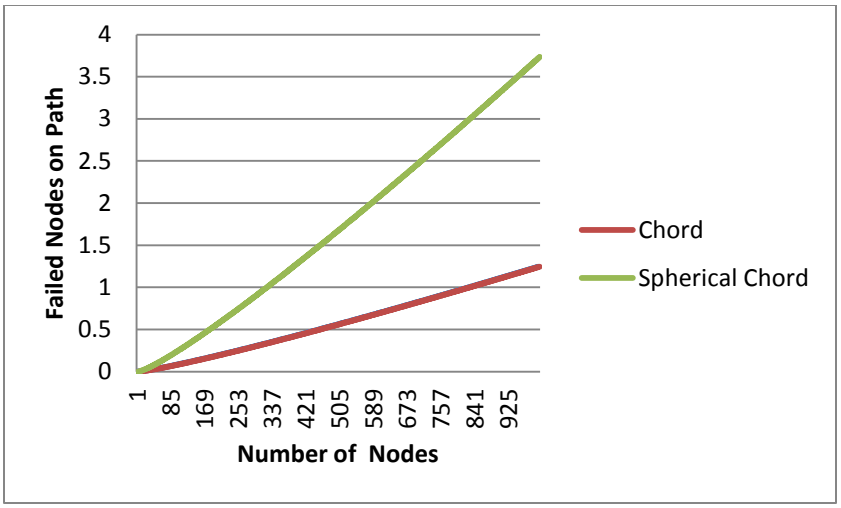

Figure 5.4. Overhead in Spherical Chord

The above analysis shows that the overhead for the computation of stabilization protocol will increase, as the probability of failed nodes on the lookup path increases. However, we can overcome this overhead by an increase in the number of lookup paths which subsequently increases the privacy of the network.

6. ANALYSIS OF SPHERICAL CHORD

6.1. PATH LENGTH ANALYSIS OF SPHERICAL CHORD

The Spherical Chord protocol is analyzed with the traditional Chord and SALSA protocols. The concluding results related to path length are shown. The initial conclusion is that the path length of the Spherical Chord is greater than the traditional system. Our system has a greater path length when a lookup is initiated. However, latency resulting out of greater path length is balanced out by the increase in the number of paths. In Chord protocol there is only a single direction in which lookup requests are routed and the requests converge towards the target node. Malicious nodes take advantage of this situation and drop the packets. However, in Spherical Chord when a lookup is initiated multiple paths can be chosen and that would make our system more resilient against routing attacks. Figure 6.1 shows the path length analysis of Spherical Chord.

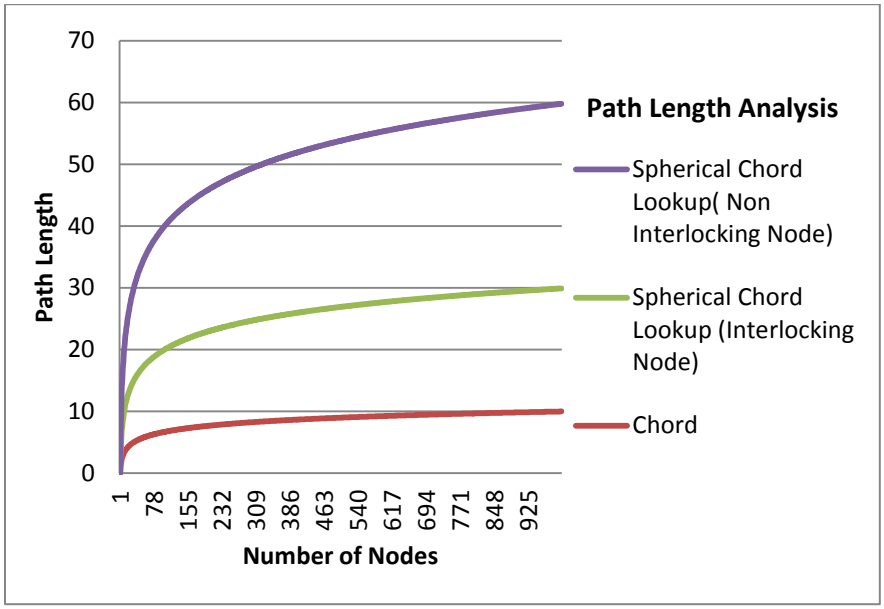

Figure 6.1. Path Length Analysis

6.2. PATH AVAILABILITY METRIC OF SPHERICAL CHORD

Path availability is defined as the number of paths available by Spherical Chord at any time during a lookup operation. The following steps are taken in order to calculate

the path availability metric of the Spherical Chord. Let the probability of a node failure during a lookup operation be *q*. Timeout is defined as the total failure in the lookup operation which is a function of the number of available paths [17]. Let *c* be the number of nodes contacted during the lookup operation. Thus, the number of timeouts for one lookup operation is *c*q*.

$$\text{Path Availability metric} = \frac{No.of\ timeouts}{Total\ Paths\ Available} \tag{4}$$

The failure to find a node is thus reduced because of the increase in the number of available paths in Spherical Chord. The graphical representation is given in figure 6.2.

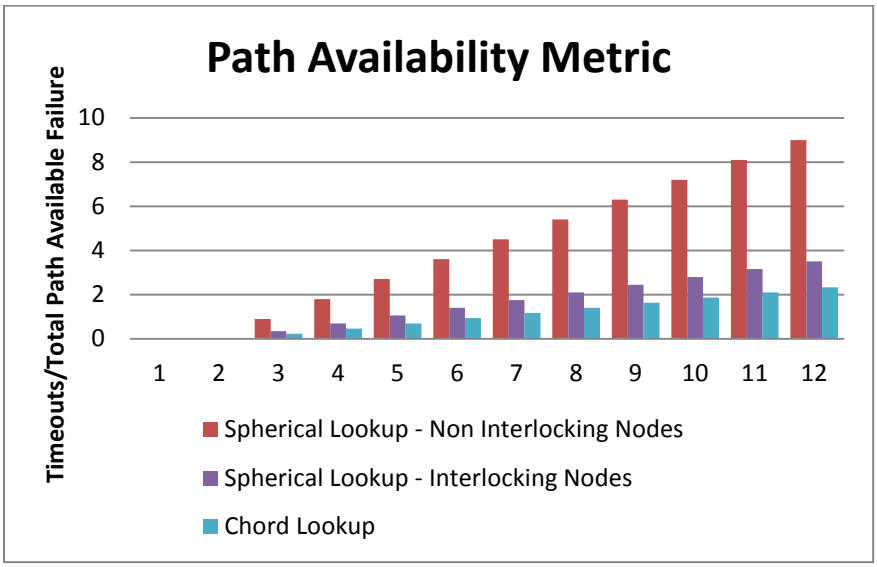

Figure 6.2. Path Availability Metric

The result shows the failure recovery of Spherical Chord in comparison with Chord. The reason for such an improvement is because at every lookup the protocol has extra paths available. Thus, when a node fails in the lookup operation it has another path available. The probability of failure for a particular lookup operation reduces as a result of the extra paths that can be taken in the protocol.

6.3. LOOKUP FAILURE ANALYSIS OF SPHERICAL CHORD

In Chord the requests follow only a single uni – directional route which makes it susceptible to routing attacks. In SALSA redundant lookups are initiated to fight against these attacks, however, these lookups follow the same set of predefined paths. In Spherical Chord, multiple paths are present which reduces the effect of such attacks. Thus, the probability of a lookup being captured by a malicious node is given as,

$$1 - (1 - f)^p \tag{5}$$

In the formula, *f* is fraction of malicious nodes in the system. Here *p* is said to be path length for the lookup operation. For the Chord protocol it is *log N*, and for the Spherical Chord the path length is a maximum of *3*log N*. The analysis is given as follows,

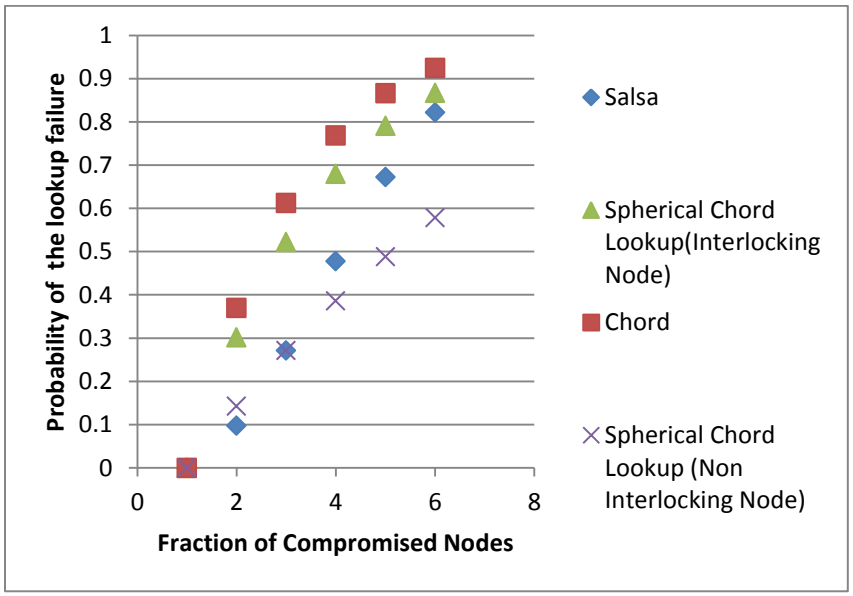

Figure 6.3. Lookup Failure Analysis of Spherical Chord

From Figure 6.3, it can be concluded that SALSA performs better than Chord. However, Spherical Chord performs better when the malicious nodes in the system increases because more paths are available the system becomes resilient against routing attacks.

7. CONCLUSIONS

Today privacy is becoming increasingly important in peer to peer based systems. This thesis discusses the Chord and SALSA protocols which are used to lookup a data item in a structured overlay network, while also providing a certain degree of privacy. A new protocol is subsequently presented called the Spherical Chord which incorporates more nodes and provides more paths to communicate, thereby enhancing privacy in data lookup requests. Furthermore, the protocol is analyzed against numerous other routing attacks and it is shown that an increase in the number of lookup paths also significantly improves the availability of the system as well.

BIBLIOGRAPHY

[1] Borisov, Nikita, "Anonymous routing in structured peer-to-peer overlays," *PhD dissertation*, University Of California, 2005.

[2] Borisov, Nikita, George Danezis, Prateek Mittal, and Parisa Tabriz, "Denial of service or denial of security," In *Proceedings of the 14th ACM conference on Computer and Communications Security*, 2007.

[3] Freedman, Michael J., and Robert Morris, "Tarzan: A peer-to-peer anonymizing network layer," In *Proceedings of the 9th ACM conference on Computer and Communications Security*, 2002.

[4] Goodin, Dan, "Tor at heart of embassy passwords leak," *The Register*, 2007.

[5] Levine, Brian N., Michael K. Reiter, Chenxi Wang, and Matthew Wright, "Timing attacks in low-latency mix systems," In *Financial Cryptography*, Springer Berlin Heidelberg, 2004.

[6] Wright, Matthew, Micah Adler, Brian Neil Levine, and Clay Shields, "An Analysis of the Degradation of Anonymous Protocols," In *Network and Distributed System Security Symposium*, 2002.

[7] 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, 2003.

[8] Karger, David, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997.

[9] Tran, Andrew, Nicholas Hopper, and Yongdae Kim, "Hashing it out in public: common failure modes of DHT-based anonymity schemes," In *Proceedings of the 8th ACM workshop on Privacy in the Electronic Society*, 2009.

[10] Sit, Emil, and Robert Morris, "Security considerations for peer-to-peer distributed hash tables," In *Peer-to-Peer Systems*, Springer Berlin Heidelberg, 2002.

[11] Wallach, Dan S, "A survey of peer-to-peer security issues," In *Software Security—Theories and Systems*, Springer Berlin Heidelberg, 2003.

[12] O'donnell, Charles W. and Vinod Vaikuntanathan, "Information leak in the Chord lookup protocol," In *Fourth IEEE International Conference on Peer-to-Peer Computing*, 2004.

[13] Needels, Keith, "Detecting and recovering from overlay routing attacks in peer-to-peer distributed hash tables," *PhD dissertaiton*, Rochester Institute of Technology, 2008.

[14] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S, "A scalable content-addressable network," In *Proceedings of ACM SIGCOMM*, 2001.

[15] Hazel, Steven, and Brandon Wiley,"Achord: A variant of the Chord lookup service for use in censorship resistant peer-to-peer publishing systems," In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[16] Nambiar, Arjun and Matthew Wright, "Salsa: a structured approach to large-scale anonymity," In *Proceedings of the 13th ACM conference on Computer and Communications Security*, 2006.

[17] Ray, Souvik, "Design and analysis of anonymous communications for emerging applications," *ProQuest*, 2008.

[18] Ripeanu, Matei, "Peer-to-peer architecture case study: Gnutella network," In *Proceedings for First International Conference on Peer-to-Peer Computing*, 2001.

[19] Saroiu, Stefan, Krishna P. Gummadi, and Steven D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," *Multimedia Systems*, 2003.

VITA

Doyal Tapan Mukherjee was born in India. In June 2008, she received her B.S. in Information Technology from D.Y.Patil College of Engineering, Pune. She received her M.S. degree in Computer Science from Missouri University of Science and Technology, Rolla in August 2014. She has also has relevant work experience as a Senior Software Engineer in Capgemini India for two years. She has also worked as a Software Development Research intern during her Master's program.

Doyal Tapan Mukherjee has been a member of the Institute of Electrical and Electronics Engineers (IEEE) since 2013.