THE REPUTATION SYSTEM FOR ROBUST, STRUCTURED P2P SYSTEMS

by

APURV ASHOK DHADPHALE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

To my mother Lalita and my father late Dr.Ashok Dhadphale - the most important persons in my life, whose dreams inspired me to be here today

ACKNOWLEDGEMENTS

First of all, I would like to give my heartiest thanks to my supervising professor Dr. Matthew Wright for giving me the opportunity to work with him in the research project and pursue my thesis in Information Security. It was really an excellent learning experience to do research under his guidance. As a mentor, he is a wonderful person. I also wish to express my gratefulness for giving me an inspiration to overcome my failures and showing a great patience.

I would like to thank Dr.Donggang Liu and Dr. Mohan Kumar for their interest in my research and being part of my thesis committee.

I wish to express my gratitude for my late father who taught me to keep high ambitions and making me who I am today. Most importantly, I would like to thank my mother. It's because of her constant encouragement and sacrifices, I could achieve my dream. My special thanks to my dearest sisters Amita and Deepti for their continuous support. I am always grateful to these persons for being with me in all the ups and downs of my life.

Finally, I wish to give thanks to Qi Dong, Tara Mallesh and all lab-mates for their friendly accompany and cooperation. I also wish to thank my friends Aditya, Rushikesh, Sankalp and Akshay for relieving the stress during the tough times.

November 23, 2009

ABSTRACT

THE REPUTATION SYSTEM FOR ROBUST, STRUCTURED P2P SYSTEMS

Apurv Ashok Dhadphale, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Dr. Matthew Wright

Structured peer-to-peer systems are distributed communication systems that typically use Distributed Hash Table (DHT) indexing to efficiently locate the resources. These networks are highly scalable and can route the messages correctly even for the extremely dynamic environment. But these networks are vulnerable; even a small fraction of malicious nodes can bias the lookup results when they are present on a lookup path. In our thesis, we address this problem of reliably searching the insecure p2p networks. We propose a reputation system to reduce the number of failed lookups and make the networks more robust. For our study, the concept is applied to the Salsa peer-to-peer communication system. Since Salsa is a DHT-based structured p2p system, it is possible to approximate the actual overlay network and lookup path which is not possible with the other unstructured p2p networks like BitTorrent. Each node builds its own reputation tree using the look-up results. The closest results are considered to be good. The reputations are then used to select the nodes for lookup. Since we use the redundancy that is inherent in Salsa, there is no separate communication overhead to collect reputation. Also since the working of reputation system does not depend on peers, it is not subject to the attacks like bad-

mouthing, whitewashing. We first study the effectiveness of the reputation system for a static Salsa environment where the peers are fixed and then adapt the approach to dynamic environment where the peers join and leave randomly. We modified the existing continuous time Salsa simulator to include reputation system module. Using the simulation results, we show that the number of failed lookups reduces by up to 90%. The experimental results also demonstrate how the different system parameters can be changed to control reputation score and further decrease the number of failed lookups.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Peer-to-peer networks are distributed networks where the peers directly communicate with each other instead of using client-server architecture. This way they could harness huge amount of resources in terms of bandwidth, storage and processing power. They have become quite popular and are used for various applications such as sharing multimedia files like that in Gnutella [1] or for internet telephony using Skype [2]. The important consideration for p2p networks is the mapping of the resources (such as files) to their locations. The unstructured p2p networks lack a scalable, distributed mechanism for indexing and thus are inefficient in resource lookups.

Structured peer-to-peer networks, on the other hand, are based on DHT (Distributed Hash Table) - a distributed data structure. They thus provide an efficient lookup of resources, are highly scalable and stable under dynamic conditions. CAN [3], OpenDHT [4], CFS [5] represent few of the DHT based p2p applications. There has been a lot of research going on the various aspects of lookup functionality like efficiency, fault-tolerance, maintainability and load balancing to name a few.

1.1 Robust Lookup For Peer-to-peer networks

In most of the peer-to-peer networks, there is almost no control over the peers that join the system. In that case, security becomes a critical aspect of the lookup process. In this thesis, we investigate the problem of looking up the resources reliably against the attacks. We assume a threat model where the malicious peers are

distributed randomly across the networks and can be a part of distributed lookup process. In reality, colluding malicious peers could redirect the queries to unwanted destination or execute denial-of-service attack. For this study, we focus on the malicious behavior wherein the attackers subvert a lookup operation by reporting a false owner of the resource which could be another malicious node rather than the true owner. This way the attacker can subvert the fundamental process of lookup and can perform other types of attacks on the integrity of p2p networks. Thus the aim of the security mechanisms should be twofold: 1) to detect the attack 2) make lookup process robust so as to locate the correct owner of the resource in spite of the attack. Note that the resource replication is the solution implemented at higher level and is a complex solution in terms of maintaining the consistency, increased storage. But we are interested in the solution at low level that would help in avoiding the replication solution at higher level [6].

1.2 Contribution

The problem of reliable resource location has got significant attention among research community and several approaches have been proposed to mitigate the problem. The most common solution is to do redundant lookups to locate target. Halo [6] proposes a novel scheme of disjoint redundant searches in DHT by modifying the core search algorithm of DHT using the fact that target of lookup operation exists in routing tables of peers across DHT. Cyclone [7] is another system that proposes a new DHT (hierarchical version of Chord [8]) and through multi-path routing achieves the redundancy. In Tapestry [9], peers work in pairs for routing messages or searches for each other through their neighbors and by verifying the path taken. In this thesis, we propose a reputation system to reduce the number of failed lookups and make the networks more robust. For our study, the concept is applied to the Salsa peer-to-peer

communication system. Since Salsa is a DHT-based structured p2p system, it is possible to approximate the actual overlay network and lookup path. It also implements the redundant lookups through its local contacts and thus like above mentioned systems we use this redundancy to make the system robust against the attacks. We study the effectiveness of proposed system through simulations for static as well as the dynamic environment and show how the success rate of the lookups increases.

1.3 Organization

The rest of the paper is structured as follows. In Chapter 2, we discuss the background concepts and some related works. Since we design a reputation system for Salsa, we present the basic design of the Salsa system in Chapter 3. Then we explain in Chapter 4 the actual design of reputation system and related algorithms. In Chapter 5, we describe the simulation design and the setup of the systems that we tested. We also describe the objectives behind different experiments. We present and discuss the results of our simulation in Chapter 6. Chapter 7 concludes the thesis.

CHAPTER 2

BACKGROUND

In this chapter, we explain some basics of peer-to-peer networks, concept of DHT. We also present some related work on reputation systems. We introduce some nomenclature that will be used throughout the rest of the thesis. We address the peers in the network as nodes. The node attempting to do a lookup in peer-to-peer network, is called, the *source* and the intended resource or the peer that it is looking for as *target*. We define a node as *honest* if it does what it is supposed to do according to the system protocol, whereas we define it as *malicious* if it sometimes does things which it is not supposed to do. For example, a *malicious* node may provide misleading information.

2.1 Peer-to-peer Communication

A peer-to-peer network, also known as P2P, is basically a distributed network architecture in which the participants make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts). That is peers are both suppliers and consumers of resources. These networks have become popular because of their ability to share multimedia content. BitTorrent, Kazaa are few examples of the well-known file sharing networks on Internet. Peer-to-peer networks are broadly classified as Unstructured and Structured p2p networks

2.1.1 Unstructured p2p

In an unstructured P2P network, the nodes connect arbitrarily to each other to form a network of overlay links. Overlay links are the logical connections among nodes. In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network to find as many peers as possible that share the data. Broadly, there are three models of unstructured architecture. In pure peer-to-peer systems the entire network consists of peers that act as equals with roles of both client and server. There is no special network overlay and the resource lookups are done mostly through flooding. Hybrid peer-to-peer systems have infrastructure nodes which act as representatives of collection of other nodes, and often called as supernodes [8]. In centralized peer-to-peer systems, a central server is used to keep the indexing and initializes the entire system. This looks like a structured architecture; however the connections between peers are not determined by any algorithm. Napster, is a popular example of the centralized model. The examples of the decentralized model is Gnutella. Kazaa is an example of the hybrid model.

2.1.2 Structured p2p

In structured peer-to-peer networks, the overlay network for a peer is fixed. In other words, it has knowledge of fixed other nodes in the network so as to route the lookup query. Different systems define different algorithm to choose the nodes in the network. However, most of them use distributed hash table-based (DHT) indexing, such as in the Chord system. These types of network also define an algorithm or consistent protocol that can be used to route the messages efficiently through the network and perform resource lookups. We will see in the section what DHT is and how it is used for such effective searching.

2.2 Distributed Hash Table

A *hash table* is a data structure that stores keys with values and particularly efficient for lookup. For example, a hash table can associate person's name as keys with other information related to that person's name so that necessary information can be found efficiently. Such put-get operations form the fundamental functionalities in p2p networks. Thus a class of network structures and related search protocol known as Distributed hash table (DHT) is defined. Like hash table, DHT provides the efficient lookup service for a decentralized distributed system. In a P2P systems, a DHT partitions ownership of the keys among the nodes of the system in a distributed manner. Thus DHT based networks are becoming popular for efficient resource discovery. The main features of DHT include:

- Decentralization: The nodes collectively form the whole system without any central coordination. Instead of complete network representation, each node keeps minimum structural information about p2p networks by carefully selecting a small set of nodes.
- Scalability: Scalability is another important advantage as the algorithm is executed locally. When the node joins the network, it doesn't affect most of the other nodes. Thus such networks can add as many nodes as possible without smoothly.
- Fault Tolerance: There is no central control and thus no central point of failure. Also joining, leaving of nodes has little effect on the other nodes and system could perform reliably.

In DHTs, a graph defines the overlay network which specifies the machines that are linked and the algorithms to store and locate data of interest. Thus, the two important aspects for DHTS are:

1. An underlying graph or the network structure
2. The way the resources and their storage network nodes are associated.

We will see in the next section how this mapping and structure is defined for Chord. Salsa [14], Pastry [17], Coral Content Distribution Network [18] are the other few examples where DHT is used.

2.2.1 Chord

Chord [8] addresses the fundamental problem of efficiently locating a node that stores a particular data item in a P2P system. Each data item is associated with a key or identifier (ID) and this data-ID pair is stored at the corresponding node to which it maps. It uses an identifier circle where the IDs and nodes are arranged around a circle in sorted order. Each node owns all the IDs in between itself and its predecessor node in the circle. So whenever we look for any key or ID, we first find that ID and then find the successor node on the ring of that ID indeed where the data item corresponding to that ID is located. Chord uses consistent hashing which has several good features. Thus there is a high probability that the load gets balanced over all the nodes in the system. It allows nodes to join and leave the network with minimal disruption. It provides weak form of authentication. The most important use of consistent hashing has been to form the foundation of DHTs as described earlier. The underlying connectivity graph is defined such that each node in the network have knowledge of direct routing information about O(log n) other network nodes, called finger nodes. The routing information table which stores these contacts is called finger table. A node does not store information of all the nodes in the system but only a small amount of routing information about other nodes in the finger table. This way Chord improves the scalability of consistent hashing. Salsa [14] has some features similar to Chord. Finally, with this structure, the lookup

operation is implemented by iteratively, or recursively, using the routing information for locating the best possible owner of the searched key, i.e., the closest finger to the destination.

2.3 Threat Model

Since p2p networks have no central control mechanism or administration, there is no provision of verifying the system integrity and assuring a secure search. To ensure trustworthy system functionality, reliable resource searching is therefore, becomes an important requirement. Peer-to-peer networks are highly distributed and thus have a strong threat model: any participating network node or colluding nodes can behave maliciously and thus not follow the distributed protocol designed for implementing the lookup process. A node accessed during search process could be malicious and can either stop or maliciously redirect the lookup to other malicious node or any random node and thus can effectively cause the search process to fail. This thesis is about designing a reputation system to discourage such behavior and increase the successful lookups. We model the above threat, by assuming that a constant fraction f of the n network nodes in the p2p system are controlled by an attacker and can thus act maliciously. For structured p2p networks based on DHT, participating nodes cannot control their locations in the logical ID space, as these are usually determined by a cryptographic hash function. Also since IP addresses are generally difficult to be set to a special target value, there is very less chance for maliciously selecting the network-node IDs. In theory, it is possible for an attacking node to control a resource if have access to a large range of IP addresses. But practically for network with huge number of nodes, say million, malicious node would need access to approximately million IP addresses to become the owner of a particular ID space. We assume that a set of

malicious nodes cannot somehow own a large number of IP addresses. Therefore, we can assume that they are distributed uniformly over the logical ID space.

2.4 Reputation Systems

Reputation systems have become important since when the security of distributed computing and communication systems became the critical issue. Typical reputation systems are used in Internet application especially on E-commerce websites (like eBay) and P2P file sharing systems. But these systems are feedback based and encourage the users of the applications to rate the service or product or any resource. The reputation systems play significant roles in (1) decision-making, (2) encouraging trustworthy behavior, and (3) deterring participation by malicious or dishonest users [11]. In the proposed thesis, *reputation system* may sound a misnomer initially as it is not a typical feedback based protocol. But the system does play a similar role as mentioned above to make the lookup process robust.

But like other security mechanisms, reputation systems themselves can be vulnerable to the attacks. The Sybil attack is the most basic attack of them. In this the attacker creates multiple identities. For example, one attacker may have several IP addresses in a system, i.e. he may own several nodes. The existence of botnets, in which the attacker controls a large number of corrupted nodes widely distributed in the Internet, makes the Sybil attack [12] as especially dangerous threat. Once the attacker creates multiple identities, they can then execute different attacks with following different goals

- Self-promoting: Increase the reputation falsely.
- Bad-mouthing: Deliberately reducing the reputation of others through negative feedback.

- Denial-of-Service: Subvert the underlying detection algorithms to make honest users or objects look suspicious [13].
- Whitewashing: Rejoin the network as a new user to clear the old bad behavior or behave good for the nodes other than target

The proposed reputation system runs locally at each node and the behavior of each node itself is a feedback. Thus bad-mouthing is not possible. Also self-promoting attack is not possible since then the malicious node has to behave good in order to increase its score. This in a way discourages the attacks and reduces false lookup result.

CHAPTER 3

DESIGN OF SALSA

For our study, we develop the concept of reputation system for *Salsa*. Thus before going into the details of reputation system, we present an overview of Salsa. It is a structured overlay design proposed in [14], to overcome the drawbacks of current P2P anonymous systems and to provide more secure, distributed organization for P2P anonymity systems. Salsa is designed to select nodes to be used in anonymous circuits randomly from the complete set of nodes. However, each node has a limited knowledge of the network using which it can do lookup. In this lookup procedure, Salsa uses randomness, redundancy and bounds checking to detect a malicious behavior of the attacker nodes.

3.1 Salsa Architecture

In Chapter 2, we have described the Chord system which is based on DHT. Salsa also has a Chord-like ID-space based on the cryptographic hash of the node's IP address. This makes it a fully distributed system with advantages like scalability, fault tolerance. The nodes are sorted and are arranged around a circular ID-space. Each node N_j is said to own the fraction of the ID-space between itself and its preceding node N_{j-1}. If a node requests a specific ID, say γ, which is between N_{j-1} and N_j, will be routed to N_j as it owns that ID-space. We call γ as the *target* ID. N_j is said to be the owner of that target. In Salsa, this ID-space is further divided into groups of contiguous IDs. These groups are then conceptually organized as a virtual binary tree (Figure 3.1). A node belongs to a group if its ID is in that group's

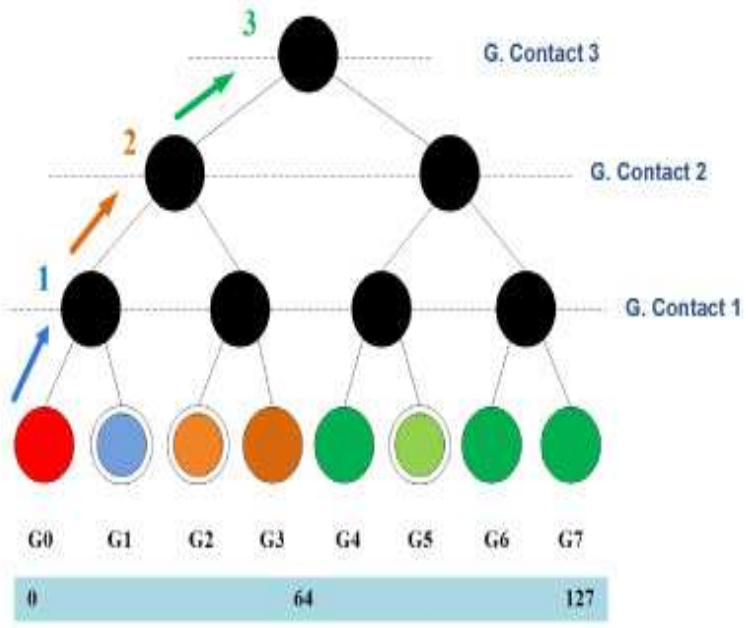

Figure 3.1. Logical organization of Salsa [19].

ID-space. The groups are numbered and then arranged on the *group ID-space* that is separate from the node ID-space and usually is smaller. In Figure 3.1, we present a tree which uses 7-bit ID-space (0-127) and 3-bit group ID-space (0-7).

Each node in Salsa has two routing tables namely, a *local contact table* and a *global contact table*. These contain the information that the node will use to route a lookup through network. The *local contact table* contains information of all the nodes in the same group. The information is nothing but the ID and their owner IDs. In its *global contact table*, a node stores information about a contact for each level of the virtual tree structure of Salsa. Let us explain it using Figure 3.1. Say a node S is in group G0. It goes one level up in the tree and selects a node from the other sub-tree, which is from G1. For selecting a contact, it uses the ID-space and selects a random ID from the space. This forms its global contact at that level. For

example, here it selects an ID randomly from the ID-space belonging to G1 and finds its owner. Then it goes up by one more level and selects its second global contact from the other sub-tree, i.e. from G2-G3 randomly. Let say S selects a node from G3. It again goes one level up and selects its root level global contact from the other half of the tree randomly, that is from G4-G7. Here assume that it selects a node from G6. Thus we can see that the number of global contacts will be equal to the height of the virtual Salsa tree. In global contact table, a global contact ID and its owner ID are stored. Thus for a Salsa node, a tree is binary virtual tree structure with a known global contact at each level. Thus a lookup or searching becomes very similar to a binary search which we are going to explain in the next section.

3.2 Lookup Process, L

Let S be the *source node*, that is looking for a target ID in ID-space and starts the lookup procedure. To detect the malicious behavior Salsa makes use of redundancy. Let the redundancy level be R. There is a possibility that the attacker may somehow manipulate all the redundant lookups and return the same result. Salsa thus implements one additional check that is bounds check. Let B be the bounds check limit for S. The general procedure can be divided into two parts, *recursive lookup* algorithm that is very similar to binary search algorithm and *redundant lookup*, and is as follows:

Recursive lookup:

1. *Node S initiates the lookup for target ID T*
2. *S determines the location of T in Salsa virtual tree.*
3. *S requests its global contact to find out the owner of the target ID.*
4. *The global contact checks if T is in its own group or not.*
5. *If it is, it returns information of the owner of the target ID to S.*

6. *If it is not, the global contact itself becomes source node. It does the same steps from 2 to 6. This is done recursively until the owner of the target ID is found.*

The important property of consistent hashing is that the target owner will be closer than any other node to the target ID. Salsa makes use of this property to identify the true owner. This means that only one of the redundant requests needs to return the correct result for the query node to get the IP address of the true target owner and so if multiple results are returned, the closest node can be selected. As lookup is the main functionality in the Salsa system, let us explain it with Figure 3.1. This lookup procedure is very similar to recursive binary search. Say the query node S is in group G0 and it is using a redundancy of say four. It is looking for target T in G5. S will delegate this task of lookup to four of its neighbors in G0. Each of these four nodes will first check whether it is in the same group or not. If yes, it is done. If not, each of them will delegate this task to its global contact and the recursive lookup proceeds as described earlier. At the end S will get four results. If all the nodes involved in the four lookup paths to target owner were honest and there were no malicious nodes, all were supposed to provide the same result, which is the closest one from T. Whatever these results are, S will take the closest one and then do the bounds checking, the final investigation. For example, say bound B is 9, target ID T is 25 and the returned owners' IDs with redundancy 4 are 30, 30, 36, and 37 where third and fourth results are returned by malicious nodes. S will select the closest one, 30, and find that the distance between T and this owner's ID is 5 which is inside the bound 9. So S will select it. But if the bound B were 4, then this owner's ID would fail to pass the bound as the distance 5 is more than the bound 4. In this case, S would do the whole lookup procedure until it gets an owner whose ID is inside the bound B. To successfully manipulate results, a malicious node has to return all the lookup results as same and pass the bounds check. That is it should return same IDs

```
Input  : requestedNode, targetNode, treeLevel
Output: Information about the targetNode
1.1  key ← SNode[requestedNode].groupNo;
     ; // "SNode" represents an object of Node type
1.2  target ← SNode[targetNode].groupNo;
1.3  low ← 0;
1.4  high ← (numberOfGroups - 1);
1.5  for l ← (treeLevel − 1) to 0 do
1.6      mi ← (low + high)/2 + 1;
1.7      if key < mi then          /* requestedNode on lower subtree */
1.8          if target >= mi then  /* targetNode on higher (different)
             subtree */
1.9              intermediate ←
                 SNode[requestedNode].GlobalFingerTable[l];
1.10             break;
1.11         end
1.12         high ← (mi - 1);
1.13     else                     /* requestedNode on higher subtree */
1.14         if target < mi then     /* targetNode on lower (different)
             subtree */
1.15             intermediate ←
                 SNode[requestedNode].GlobalFingerTable[l];
1.16             break;
1.17         end
1.18         low ← mi ;
1.19     end
1.20 end
1.21 if SNode[intermediate].groupNo = SNode[targetNode].groupNo then
     /* found the target node */
1.22     return Information about the targetNode;
1.23 else /* still not found it, so lookup recursively through the
     Salsa virtual tree */
1.24     return recursiveLookup(intermediate, targetNode,
         SNode[intermediate].treeLevel);
1.25 end
```

Algorithm 1: recursiveLookup[19]

as close to the target ID as possible. When the target owner itself is malicious, we expect to get the correct result. Also for proper working of the system, Salsa defines thresholds on the group population. Thus whenever group population exceeds the upper limit group is also split and in case if it drops below lower threshold groups are merged.

3.3 Dynamics of Salsa

We now focus on the dynamics of the Salsa system which includes the algorithms for node joining the system or leaving the system. One issue for any peer-to-peer system is that an attacker who adds a large botnets worth of nodes could dominate and control functionality of the system. For Tor [16] like server-based systems, with presumably fewer nodes, we note that a higher barrier to entry is required to ensure that most nodes are at least associated with a unique owner.

- Joining of Node: In Salsa the nodes should, if possible, join through trusted friend which is a node in the network. This way a small network could also be built into a larger one. For This new node N, its friend can perform lookups to identify a subset of the nodes in Ns group, as determined via a hash of Ns IP address. N can then contact these nodes with a join message, to which the nodes respond with a full list of the group members. There will be a performance and security tradeoff between the number of lookups performed by the friend and the correct identification of the full group. However, if the lists are mismatched users can guess it as a possible attack, so the attacker must be confident that N only contacts attackers and not other nodes in the group. Otherwise, the attacker nodes must also send the full list. If the new node N has no friends using the system, Salsa proposes to advertise a subset of nodes on, a website or a bulletin board. For example, a node may post its own IP and those of its

global contacts. Since the global contacts are random and verifiable by hashing the IP with the level of the tree, N has some assurance that they are not an attackers hand-picked selection.

- Node Leaving/Exit: Salsa expects that the node should notify the other group members and nodes that have recently connected to it as a global contact whenever it leaves. Since then nodes can update their global contacts before the next round of requests, smooth transition can be made possible. The disconnect message could be a spoofed leave message and are intended to be a denial of service attack or a way to redirect traffic to corrupt nodes. Thus proper handshaking needs to be done. But a node may also disconnect quietly. In that case, the nodes that had used it as a global contact will find out in the next round of requests and will then determine the identity of the new global contact. Salsa proposes a fully distributed algorithm for updating global contacts.
- Splitting a Group: After joining if a node finds that its group population is more than a threshold value; it will invoke a splitting process for that group. First, it will imagine the group as two halves and compute the number of nodes in each half. If it finds that the population in each of the halves is greater than the threshold value for merging groups, it will split the group and build its local and global contact lists accordingly. Splitting a group is completely local to a node in that the node just discards IDs belonging to the other half from its own local contacts list. The advantage of making it local is that, any node can split a group and can have its own view of the virtual tree structure. As a result, other nodes in the same group are not affected by the splitting procedure. This method also has the benefit that malicious nodes can not influence the group structure to other nodes, to get their own advantage. Salsa sets the value of

threshold for group splitting, based on the group population distribution of simulation results. [19]

- Merging Groups: After joining or during a lookup, if a node finds that the population of its group is smaller than a threshold value, it will invoke the merging procedure. First the node will broadcast its local contact list to all of its group members. After receiving this list, each of these nodes will individually match their own local contact lists with the received one. If a member node finds mismatches, it will contact each mismatched node. If the node is alive, it will broadcast its presence to all other members of the group. The broadcast ensures that other member nodes do not need to contact it again to crosscheck the mismatch, thus reducing the message overhead to some extent. This process automatically updates the local contact lists of all group members. If all the group members agree with the number of the contacts in the local contact list of the merge initiating node, the merge procedure continues. The initiating node will contact its global contact in the other half and merge the two groups with the global contact's help. All the members of the new two groups will update their local and global contacts as well as group ranges. Again, the threshold value for merging groups is a configurable parameter. [19]

CHAPTER 4

REPUTATION SYSTEM DESIGN

In this chapter we present the design of our proposed reputation system and the algorithms we employ to calculate reputation scores. We first discuss the basic idea of our proposed reputation system. Then we explain the design in the context of a static environment in which the peers are fixed. We then present a modified system for a dynamic environment in which the peers join and leave. Finally we propose to calculate reputation scores for cases with no information based on the distance between the target ID and ID returned by a lookup. We mainly discuss the motive behind these designs and investigate the effects of different factors on reputation scores.

4.1 Basic Idea of The Proposed Reputation System

In this thesis, we use reputation system for making the peer to peer networks more robust. The proposed reputation system is not a typical reputation system that collects ratings of a node from its peers and uses these ratings to get good service from the reputed peers. Instead in the proposed system, the peers by themselves learn the reputation scores of local contacts and use them to do successful lookups.

Salsa makes use of redundant lookups to prevent malicious nodes from returning false information without detection. As we discussed in Chapter 2, Salsa selects redundant local contacts and delegates the task of lookup. Once the results are obtained, it applies bounds checking and selects the closest result as the best or true result. Since Salsa already has this redundancy, we in our design do not need to use

any exploratory messages such as the ones used in [15] just to get the reputation scores. We use the redundant lookup results to build reputations. The node lookups in Salsa follow recursive path through its logical tree structure. The node tries to find the target among local contacts. If not found, it contacts its global contact to delegate the task. This goes on until the actual target is found. The results are returned recursively back on the same path. The node who delegates its lookup request to its local contacts is not aware of their global contacts and hence must be unaware of the exact path that request might have taken. However, as Salsa is DHT based structured network we can approximate the path through tree. Thus we propose to maintain reputation scores in the form of tree data structure called reputation tree that approximates the underlying Salsa tree structure. Each peer will generate its own reputation tree based on the knowledge of actual network structure – number of groups, height of tree, and number of local contacts. Each node of this tree is collection of that peer's local contacts and their corresponding scores. Thus, the reputation scores of the local contact are effectively the scores for global contacts for that local contact to reach that part of the tree. When the peer receives a result from local contact say L for lookup of target node say T, it would approximate the path from its own group to the group of target node. As the reputation system recursively follows the lookup path through reputation tree, it will update the score of L at each node. This score is 1.0 if the lookup is successful or 0.0 if the lookup fails. Therefore if L has some malicious node in its path to target T and if the result returned is failure then the reputation score of L will decrease in that part of the tree through which the path goes.

We present a sample reputation tree for node 8 in Figure 4.1. Let the node 8 be a source node S. For this example, we consider local contact 15 only. S uses 15

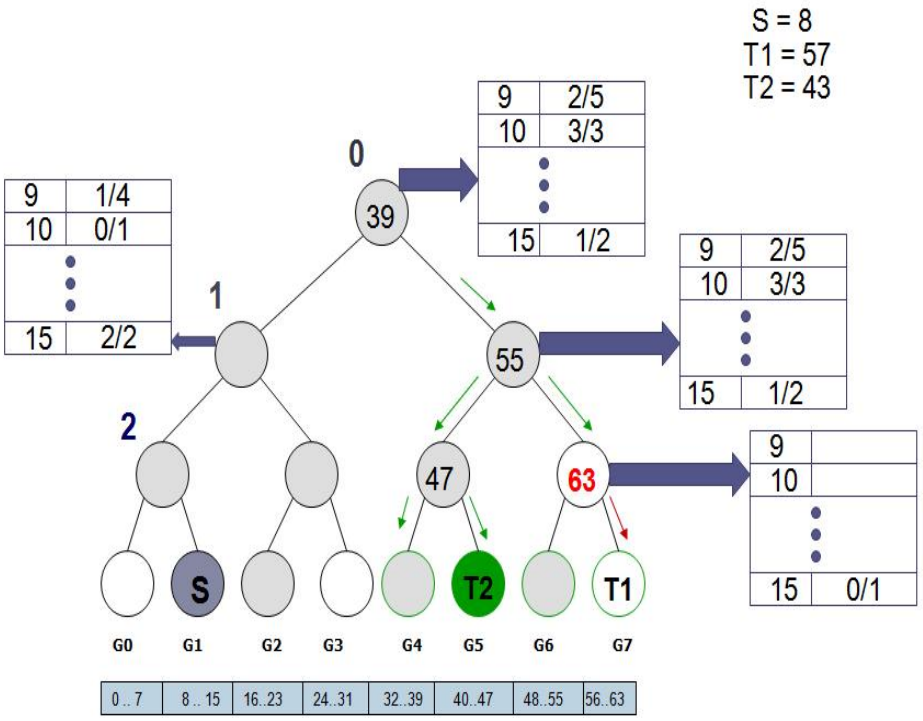

Figure 4.1. Example of Reputation Tree.

to do lookup in its own half of the tree. 15 returns correct result twice and hence the reputation score becomes 1.0 (2/2) for it. S then looks up target node $T1$ (i.e. 57) from group G7. It again selects 15 to do this lookup. Assume that 15 has not been used previously to do any lookup in this part of tree. 39 is node 15's global contact at level 0. 15 asks 39 to do a lookup for $T1$. Then 39 delegates this lookup to its own level 1 global contact i.e node 55. 55 then delegates this task to 63 which is a malicious node. The target node $T1$ is in the group of 63. 63 returns a false owner ID. This path is approximated in reputation tree. The path is said to be approximated, since the node S is not aware of the nodes 39, 55 or 63. Thus, the score of 15 becomes 0 along the path of lookup query. S then does lookup for another target $T2$ i.e node 43. Node 15 is selected again. There is no malicious node along this lookup path [39-55-47], and thus the result returned is true. The score for node 15 becomes 0.5

(1/2) at level 0 and level 1 nodes of reputation tree. This process is explained in algorithm 2.

```
2.1  Input  : Height of tree m, Possible number of groups gpt, Number of
              Local contacts n of the given node
     Output: Reputation tree rt
2.2  score = 0 ; /* Build initial reputation tree                    */
2.3  rt = SalsaReputationSystem(m,gpt);
2.4  rt.generateBinaryTree(n);
2.5  Select random local contact (without-replacement) L for lookup;
2.6  result = Lookup(L) ;
2.7  if result == success then ; /* There could be 'k' such local
     contacts that will return successful lookup.                   */
2.8
2.9  |  score = 1
2.10 end
2.11 score = 0
2.12 lookupPath = rt.approximateLookupPath(targetGroupNumber);
2.13 updateReputationScore(L, score,lookupPath);
```

Algorithm 2: The Reputation System Algorithm

4.2 Static Environment

In static environment, a reputation score is the success rate of local contact 'L'. We thus maintain count of total number of lookups done by each node and the number of successful lookups. Using these counts we calculate or update reputation scores as given in Algorithm 3. We assume that system has 20% nodes as malicious. Thus, we initialize all the reputation scores with 0.8. Also note that counts of total number of lookups and number of successful lookups are both initialized to 1.0. These values are chosen to make system less strict in approach. Otherwise if they were 0.0 and if a local contact returns the first result as false then its score will become 0.0 from initial value of 0.8. This would be a huge drop in its reputation. Also assigning

```
    Input  : local contact lc, score s, α, lookupPath p
    Output: Updated Reputation Tree
3.1 for node n in lookupPath p do
3.2 |  n[lc].totalLookups = n[lc].totalLookups + 1;
3.3 |  n[lc].numberOfSuccess = n[lc].numberOfSuccess + s;
3.4 |  n[lc].repScore = n[lc].numberOfSuccess/n[lc].totalLookups;
3.5 end
```

Algorithm 3: Update Reputation Score For Static System

values as 0.8, would give fair chance to all nodes at start. Being a static system, the reputation system is initialized at start and not modified later as all peers are fixed. It is also the reason why use of success rate as a reputation score works fine.

4.3 Dynamic Environment

In a dynamic environment, peers can join or leave at any time. As we have already discussed, this may lead to merging or splitting of groups. Thus for such environment we need to modify the design of reputation system.

- Joining Event: If the new node joins a network, the peer need not modify its reputation system (which is local) unless the node joins its own group. If it joins the same group then all the peers in the group should modify their reputation tree structure to accommodate this new local contact.
- Leave Event: Similar to join event, all the peers should modify their tree structure whenever the peer in their own group leaves the system. Note that the node may leave silently. In that case peer may not update it until they discover the leaving event.
- Group Split Event: The event takes place when the group population crosses certain threshold limit (Salsa recommends 80). When the group splits, the change is local. That is the node which initiates this operation generates its own

view of tree. Thus we need not reconstruct the reputation tree to accommodate the newly formed virtual groups after split. But when the group splits peers modify their local contact table. Thus they all have to modify their reputation tree to reflect fewer local contacts. This can be easily done with same logic of node leave. Here some peers are not leaving the system but are leaving the group which ultimately has same meaning from reputation system point of view. Because the peer will then no longer be concerned about the selection of those ex-local contacts.

- Group Merge Event: When the groups merge, peers in the other part of tree are not aware of this change. Thus those peers cannot reconstruct their reputation trees to reflect the change in actual network. We thus propose to keep the structure fixed. But the peers in the merged groups now have new local contacts. This is same as many join events happening at a time. Thus these peers change their reputation tree to accommodate the new local contacts.

4.3.1 Reputation Score For Dynamic Environment

In dynamic environment, a reputation score is not a success rate of local contact 'L'. But we use a following formula to calculate reputation score:

$$NewReputationScore = \alpha * score + (1 - \alpha) * OldReputationScore \tag{4.1}$$

α is a strictness factor between 0 to 1. *score* is 0 if the lookup fails and is 1 if it is successful. Thus for dynamic system we no longer need a count of number of lookups and number of successful lookups. We update the score using algorithm 4. Like static system we initialize reputation scores to 0.8 for all local contacts.

```
    Input  : local contact lc, score s, α, lookupPath p
    Output: Updated Reputation Tree
4.1 for node n in lookupPath p do
4.2 |  n[lc].repScore = α * s + (1-α)* n[lc].repScore;
4.3 end
```

Algorithm 4: Update Reputation Score For Dynamic System

As we can see from the formula, if α is high then the latest lookup result will have more influence and the peers returning good result can quickly establish a good reputation. But if α has low value then old reputation has more influence and hence it becomes difficult to improve a reputation that was once degraded. In short we can choose a suitable α to control the strictness of our reputation system.

This scheme has an advantage when a malicious node tries to deceive reputation system by behaving good initially to gain good reputation and then behaving bad suddenly. If α is high then a bad result will immediately have effect on reputation score.

4.4 Node Selection Using Reputation System

The reputation scores are used to select the local contacts for lookup process. The local contact itself might be a good node but may have bad reputation for certain part of tree. It may have good reputation for some other part of the tree. Thus while selecting local contact we must consider the reputation score of local contact based on the location of target node. The system should select the contact of good reputation randomly without replacement. To do this, we propose to use a weighted reputation score. It is calculated as:

$$WeightedScore(Li) = (RepuScore(Li) + CumulativeScore(Li-1))/TotalScore \tag{4.2}$$

Local contact	Score	Weighted Score
L0	0.1	0.1/0.7 = 0.14
L1	0	0.1/0.7=0.14
L2	0	0.1/0.7 =0.14
L3	0.2	0.3/0.7=0.42
L4	0	0.3/0.7=0.42
L5	0.4	0.7/0.7=1
Total Score =	0.7	

Figure 4.2. Calculate Weighted Reputation Score.

$$CumulativeScore(Li-1) = \sum_{j=0}^{i-1} RepuScore(Lj) \tag{4.3}$$

The system then chooses any random number $r \epsilon [0,1]$. It goes through the weighted reputation scores of local contacts. If $r <= WeightedScore(Li)$, Li node is selected. The system also makes sure that it has been selected without replacement. The use of weighted score ensures that the local contacts with low score are not selected. The example of this selection is illustrated in Figure 1.2.

But the weighted reputation scores of single node of reputation tree are not effective to select local contacts. We therefore propose to calculate multiplicative weighted reputation scores. The system approximates the path from source to target node and calculates the multiplication of all scores along the path. From the resultant scores it calculates the weighted reputation scores which are then used for random node selection as explained before. Algorithm 5 explains this process.

Input : Reputation Tree R, Number of Local contacts n of the given node
Output: Local Contact ID For Redundant Lookup $localContactId$

5.1 Initialize $multiplicativeWeightedReputationScores$, $weightedReputationScores$;
5.2 Let P(p_1, p_2..,p_k) be the path in R from source to target node where p_1, p_2..,p_k are 'k' nodes from the tree;
5.3 Let S(s_1,s_2...s_k) be the reputation scores for(p_1, p_2..,p_k) where s_k is array of scores for n local contacts;

`/* Calculate multiplicative Weighted Reputation Scores */`;
5.4 for $i \leftarrow 0$ to n do
5.5 | $multiplicativeWeightedReputationScores[i] = \Pi_{j=1}^{k} s_j[i]$;
5.6 end

5.7 $totalScore = \sum_{i=0}^{n} multiplicativeWeightedReputationScores[i]$;

`/* Calculate Weighted Reputation Scores */`;
5.8 $i = 0$, $previousScore = 0$, $currentScore = 0$;
5.9 while $i < n$ do
5.10 | $currentScore = previousScore + multiplicativeWeightedReputationScores$[i] ;
5.11 | $weightedReputationScores$[i] = $currentScore/totalScore$;
5.12 | $previousScore = currentScore$;
5.13 end

`/* Local contact selection */` ;
5.14 $index = -1$,$i = 0$;
5.15 while $i < n$ do
5.16 | ran = random(0,1) ;
5.17 | if $weightedReputationScores[i] >= ran$ then
5.18 | | if i *not selected before for this lookup* then
5.19 | | | $localContactId = i$;
5.20 | | | return $localContactId$;
5.21 | | end
5.22 | end
5.23 end

Algorithm 5: Node Selection Using Multiplicative Weighted Score

4.5 Calculating Distance-based Reputation Scores

When the source node S receives R lookup results, it chooses the closest one as the correct result. It might happen that all the redundant lookups return the same ID. In that case, all the results could either be good or could be bad. So far in our discussion, we haven't considered this important factor of distance between the actual target node and the ID returned. Thus, even when all the redundant results are same (either success or failure), we assign same score all irrespective of their distances from target. Thus we propose a modification in the calculation of reputation score. According to this, the system determines the probability of node being good for a distance δ^* between the target ID and owner's ID returned by that node. Thus, when the results returned are same, the system uses this probability as *score* value.

Thus, for a local contact L, when all the redundant results are same, we use the following formula to determine the reputation score.

$$NewReputationScore_L = \alpha * P(G|\delta \leq \delta^*) + (1 - \alpha) * OldReputationScore_L \quad (4.4)$$

Probability of Goodness is calculated using Bayes' Theorm [App A] as follows:

$$P(G|\delta \leq \delta^*) = \frac{P(G) * P(\delta \leq \delta^*|G)}{(P(G) * P(\delta \leq \delta^*|G)) + (P(B) * P(\delta \leq \delta^*|B))} \quad (4.5)$$

where

P(G) = probability that the results are good

P(B) = probability that the results are bad

$P(\delta \leq \delta^*|G)$ = probability that distance is less than or equal to δ^* for good

result

$P(\delta \leq \delta^* | B)$ = probability that distance is less than or equal to δ^* when the result is bad

δ^* = Distance between the owner ID returned by L and target ID

CHAPTER 5

SIMULATIONS

We performed a number of experiments to observe how different factors affect the reputation scores. We also verified the effectiveness of the proposed reputation system through various experiments. We first explain simulation configurations and objectives for static environment. Then we give the details for dynamic environment and the objectives of various experiments conducted.

5.1 Static System Simulation Design

We use existing continuous time simulator for Salsa written in Java. For more details about the simulator, refer [19]. We modify this simulator to include *reputation system* module. The static Salsa system has a fixed number of honest and malicious nodes. The number of groups is fixed. For static environment simulation, only lookup events are generated by the simulator as no node joins or leaves the system. We used a 30-bit ID-space, a 20-bit group address space and simulated systems consisting of 10,000 nodes respectively. We used 256 groups for the 10,000 nodes case. We fixed percentage of malicious nodes at 20% to simulate a realistic scenario unless and otherwise stated. Beyond 20% malicious nodes, there is more advantage to attackers no matter how robust the system is organized. For simulations, reputation scores are calculated by one randomly selected node. The node selects the local contacts randomly without-replacement to do lookups. For all these experiments redundancy is fixed to four. For each test, we simulated 100 separate systems to get the good

estimate of readings. We studied the performance of the proposed system using a range of different parameters.

5.1.1 Static System Simulation Objectives

We performed a number of simulations with following objectives:

1. *To find out how many lookups need to be done to get stable or ideal reputation scores.* Since this is a static system, we are sure that reputation scores will stabilize after certain number of lookups. This experiment is important to evaluate the feasibility and correctness of the proposed design. Also, once the ideal or stable values are obtained, we can compare the reputation scores with them and observe the effectiveness of the algorithm. In this experiment, we ran simulations starting with 100 lookups. Then we increased the number of lookups in multiples of 10. Beyond 1,000,000 lookups, we didn't observe much improvement in the scores. We then ran and compared different sets of 1,000,000 lookups for the same system.
2. *To compare scores at different levels of the tree.* In this experiment, we again ran different simulations from 100 up to one million but observed how the reputation scores at different levels of the tree get stabilized. The actual Salsa tree height was eight. We observed the reputations for level 0 to level 5, counting from the root.
3. *Calculate weighted reputation scores at multiple levels of the tree. Use them to select local contacts for redundant lookups.* Once the reputation scores are obtained, our next aim was to utilize them to do more successful lookups. We consider a naive system that uses information from only one level of the reputation tree. The weighted reputation scores are calculated using the scores at that level. Here we ran 100 lookups to get reputation scores. Then we did 1000

lookups using the weighted reputation scores at level 0 (for local contact selection) and observed the number of failed lookups. We repeated this for weighted reputation scores at different levels. We chose the reputation scores based on the target node location. Similar simulations were run using the reputation scores of 1000 lookups.

4. *To use multiplicative weighted reputation scores to select local contacts for redundant lookups.* In the experiment described above, we consider reputation scores from only a single level of reputation tree. In this set of experiments, we study our complete static system that includes multiplicative weighted reputation scores; calculated using scores at the reputation tree nodes that are involved in the lookup path. Before node selection, we approximate a path to the target node and compute weighted reputation scores as given in Algorithm 5. In this experiment, we run either 100 or 1000 number of lookups to get reputations. Then we ran 1000 lookups and used multiplicative weighted reputation scores to select local contacts and measure the number of failed lookups. During the measurement phase, we do not continue to update the reputation scores, as we seek to evaluate the effectiveness of scores for that snapshot of the system
5. *To observe the effect of reputation when used in the Salsa system with higher percentage of malicious nodes (more than 20%).* So far in the simulation description, the percentage of malicious nodes in the system was fixed as 20%. Our main motive behind reputation system is to discourage the attacker by selecting a good local contact for lookup and have more successful results. In this experiment, we are trying to find the effectiveness of our system even if the malicious node percentage goes as high as 50%. Here we used redundancy as a main parameter to change the effectiveness of reputation system. We increased the percentage of malicious nodes in steps of 5 from 20% up to 50%. And in

each case we vary redundancy as 4, 7 and 10. To get good estimate, we took 100 readings. We observed the effect on number of failed lookups in each case.

5.2 Dynamic System simulation design

The continuous time simulation implements a dynamic system in which nodes join and leave the system over time. We use the same continuous time simulator [19] for Salsa. We used a 30-bit ID-space, a 20-bit group address space and simulated systems consisting of 5000 and 10,000 nodes respectively. We used 128 groups for the 5000 node system and 256 groups for the 10,000 nodes case. For each test, we simulated 100 separate systems. In different tests we change the different parameters α , redundancy and malicious node percentage and observe their effects on reputation scores and in turn on the number of failed lookups.

In a dynamic environment of Salsa, we consider these events:

a) Joining of a node, J

b) Leaving/Exit of a node, E

c) Updating the global contacts, UG

d) Splitting a group, SG

e) Merging two groups, MG

f) Lookup Process, L

We treat each of the above events separately, i.e., one event occurs at a time. They are performed in the same order as they are generated. One event may invoke other events within its execution, such as a join may invoke lookups. In such cases, the sub-event is placed right after the current event in the execution queue. This means the event and its sub-event are executed one after another consecutively. In continuous time simulator, join and leave events are generated with exponential distribution.

5.2.1 Dynamic System Simulation Objectives

We performed a number of simulations with following objectives:

1. *To observe the effect of system dynamism.* In this experiment, we vary the number of join and leave events. We vary average number of both join events and leave events from 500 to 2000. We observe the effects of these changes on the number of failed lookups for a Salsa environment without our reputation system and for a Salsa environment with our reputation system.
2. *To observe the effect of damping constant α.* For dynamic environment we calculate reputation score using the formula:

   NewReputationScore[L] = α * *score* + (1-α) * OldReputationScore[L] Thus we can see that damping constant α has a major impact on reputation score. In this experiment, we change the damping constant α from 0.0 to 0.9 to observe this effect and in turn the effect on number of failed lookups.

3. *To observe how the reputation system responds to different percentage of attackers.* In the experiments described so far, we assume that the percentage of malicious nodes in the system is fixed to 20%. Usually for higher percentage than this, the attackers have more advantage. Thus in this experiment, we vary this percentage from 5% up to 30% to observe the effect on number failed lookups when reputation system is used. We keep values of α, redundancy, and network dynamism fixed.
4. *To observe the effect of redundancy on the reputation system.* In this last experiment we are interested in finding the behavior of reputation system for different values of lookup redundancy. Since the reputation system uses the redundant lookups to collect reputation scores, the redundancy has significant influence

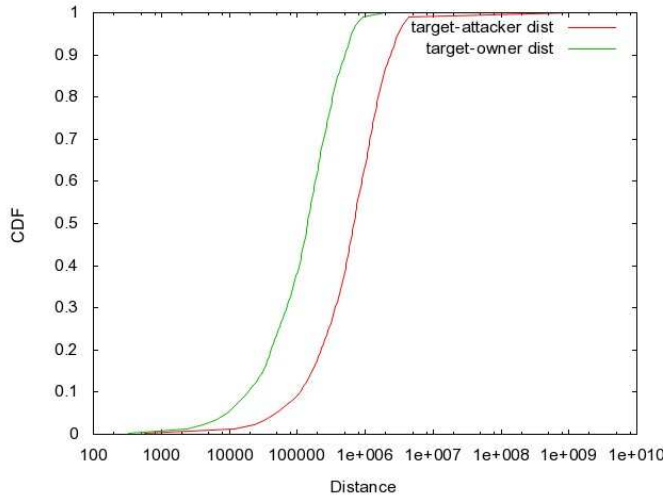

Figure 5.1. Distribution of Distance: Target-Owner and Target-closest Attacker.

on the scores. We keep values of α, malicious node percentage, and network dynamism fixed.

5.3 Distance-based Reputation Calculation

In this experiment, we first run the simulations for 1000 lookups in static environment to determine the distance between target ID and the owner ID. We also determine the distance between the target ID and the nearest malicious node ID. We plotted the Cumulative Distribution Function (CDF) for these distances as shown in figure 5.1.

As we can see, the plots have normal distribution. We used these readings to calculate distance-based reputation scores using formulas (4.4) and (4.5). In this experiment, we run 1000 lookups and observe the effect of Distance-based reputation scores on the number of failed lookups.

CHAPTER 6

RESULTS

In this chapter we present and discuss the results of experiments as described in the previous chapter.

6.1 Static Environment Experiments

6.1.1 Number of lookups to get stable reputation score

As discussed earlier, there is no significant improvement in reputation scores after 1000,000 lookups. Thus we run another set of million lookups for the same system. The "Distance" between the scores of both set is calculated and CDF (Cumulative Distributed Function) is plotted as shown in Figure 6.1. We can observe that the CDF starts from 0.4 that is 40% of the reputation scores are same (having distance of 0.0). The maximum distance is just 0.05 for which the CDF reaches 1. From this it is evident that the scores are really stable or ideal after million lookups. This is an important observation. In reality, such a high number of lookups is not

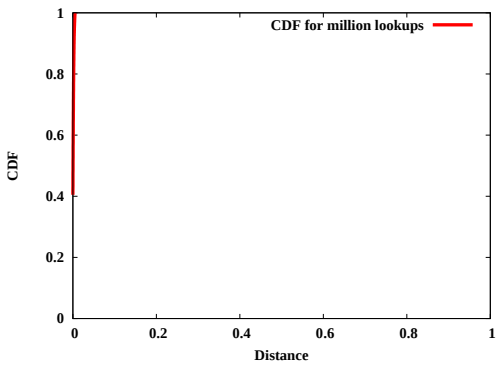

Figure 6.1. Stable Reputation Scores.

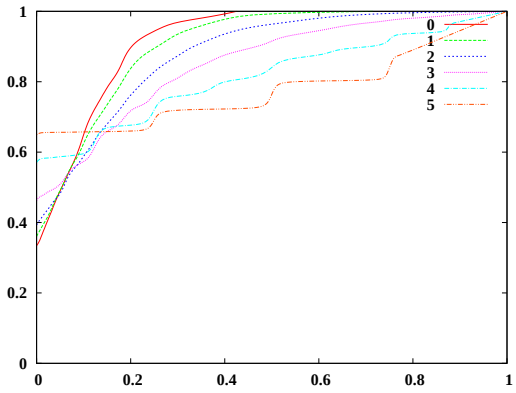

Figure 6.2. For 100 lookups.

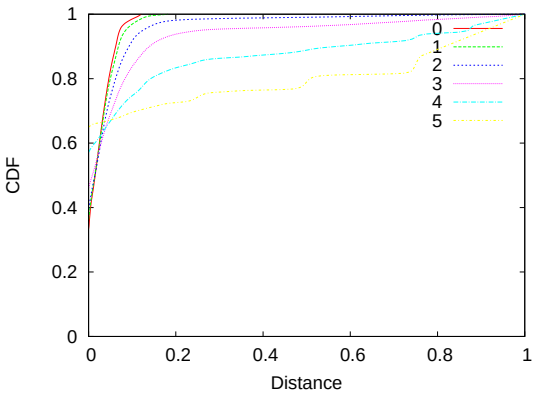

Figure 6.3. For 1000 lookups.

possible. But through simulations we can determine the relation between the actual scores and the stable scores.

6.1.2 Stabilization of Reputation Scores

In this experiment, we run simulations for different numbers of lookups from 100 up to 100,000. We also observe reputation scores at different levels of the reputation tree and how do the scores become stable. We then calculate *Distance* as a difference between the scores obtained and the stable scores. We then plot CDFs of these Distances as shown in Figure 6.2. For 100 lookups, level 5 scores are very much away from the ideal scores. As we move up the tree towards level 0, we get more accurate reputation scores. The same pattern can be observed among the tree levels for different number of lookups (Figure 3, 4, 5). Also, we can see that with increasing number of lookups the scores more accurate and the CDF curve almost becomes vertical.

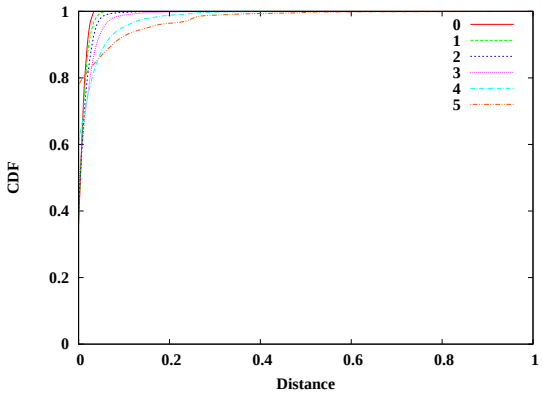

Figure 6.4. For 10000 lookups.

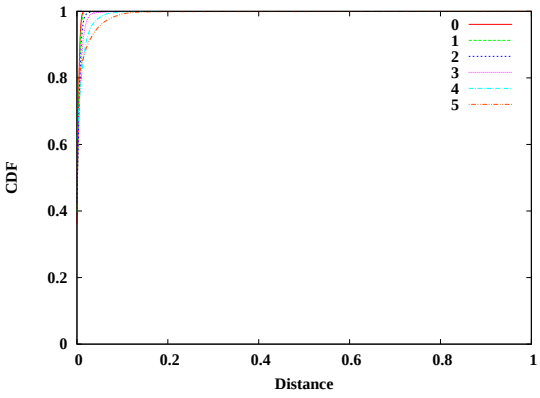

Figure 6.5. For 100000 lookups.

6.1.3 Lookups Using Weighted Reputation Scores

In this experiment, we use weighted reputation scores to select the local contacts for redundant lookups. We run 100 lookups to get the reputations and then we use these scores to do 1000 lookups. We then run 1000 lookups to get reputation scores and use these reputation scores to do another set of 1000 lookups. The results are as shown in Figure.6.6. We can see that for 100 lookups, the number of failed lookups decreases until level 1 and again it starts increasing. This shows that the scores are quite good until level 1. For 1000 lookups, the number of failed lookups decreases until level 4. This indicates that after 1000 lookups the reputation system could collect better scores until level 4. The simulation shows how the reputation scores get better and stable down the tree with more lookups.

6.1.4 Lookups Using Multiplicative Weighted Reputation Score

We now use multiplicative weighted reputation scores to select local contacts for redundant lookups as mentioned in Algorithm 5. We run 1000 lookups to get reputation scores and use these scores for another set of 1000 lookups. We set the redundancy for different experiments to 3, 4, 5 and 7. From Figure 6.7, we can observe

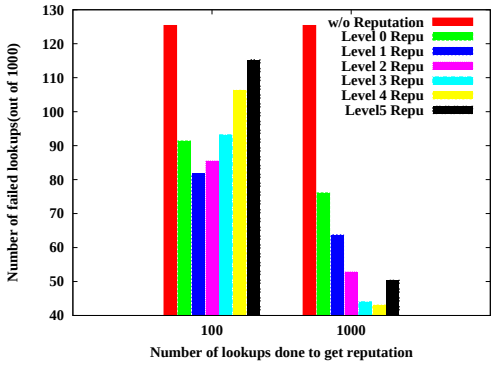

Figure 6.6. Improved Lookups using the scores from multiple tree levels.

that with an increase in redundancy, the number of successful lookups increases; for high redundancy 7, the failure rate almost drops to 0.08.

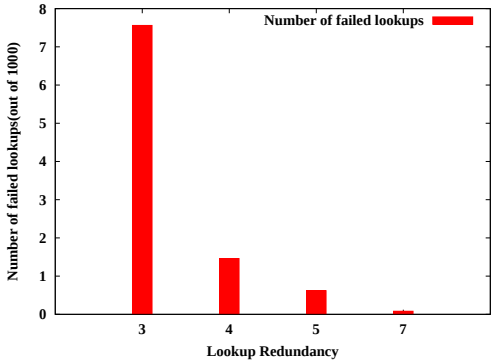

Figure 6.7. Reputation System Effectiveness for different lookup redundancy.

In another experiment (with redundancy 4), we run 100 lookups to get reputation scores. Then we use these reputation scores in 1000 lookups. For the same system, we run simulations with 1000 lookups to get reputation scores and use them for another set of 1000 lookups. As shown in Figure 6.8, for 1000 lookups without using reputation scores, the number of failed lookups is 127.22. With reputation scores obtained from 100 lookups, it is reduced to 33. And with reputation scores obtained from 1000 lookups, it drops further to 1.46.

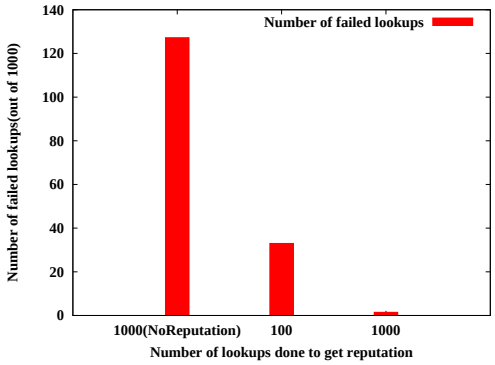

Figure 6.8. Number of Failed Lookups With and Without Reputation System.

6.1.5 Reputation System For Higher Percentage of Attackers

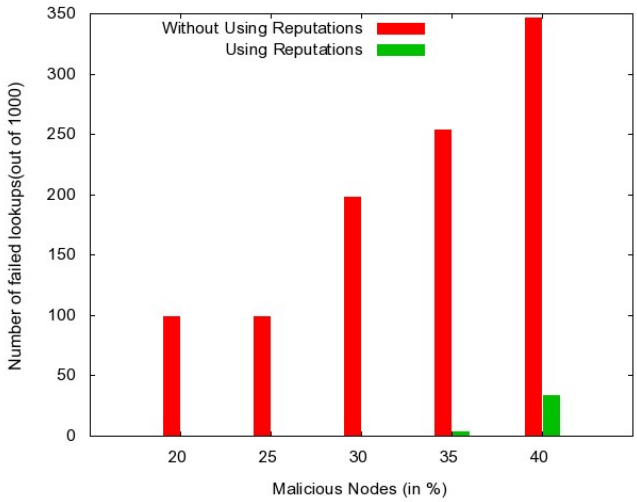

Figure 6.9. Different % of Attackers For Redundancy 4: Static System.

We now observe the effectiveness of the reputation system for networks with higher percentage of malicious nodes. We can see in Figure 6.9, that using reputations for a network with malicious node percentage of 40%, the number of failed lookups reduces to 33.94 from 346.12 (without using reputation). Thus even for higher percentage of malicious node, there is 90.2% reduction in the number of failed lookups.

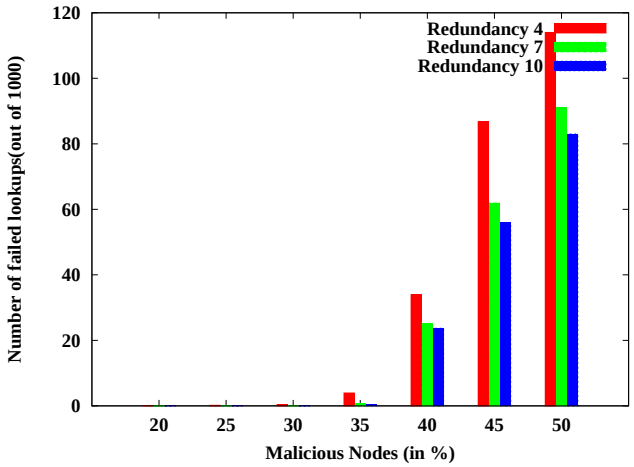

Figure 6.10. Different % of Attackers With Different Redundancies.

As shown in Figure 6.10, with increase in redundancy the number of failed lookups decreases further. For higher percentage malicious nodes, high redundancy of 10 becomes less effective as the number of failed lookups increases drastically from 0.35 (for 30%) to 23.69 (for 40%).

6.2 Dynamic Environment Experiments

In all these experiments, we run 1000 lookups and take readings of 100 systems to get the good estimate. Unless and otherwise stated, the value of damping constant α is 0.5, redundancy is four and percentage of malicious nodes is 20%.

6.2.1 Reputation system for different dynamic environment

The effect of dynamism of network on reputation system behavior can be seen in Figure 6.11. We change the average number of nodes joining/leaving the network and observe the number of failed lookups. We start the experiment with small dynamism of 500 in network. That is over the whole period of simulation, the average number of nodes that leave and join the network is 500 each. As we can see the number of

failed lookups is 5.61 when the reputation system is used. Without reputation system the number of failed lookups are 97.87. As the dynamism of the network increases, we see no significant change in the number of failed lookups for peers without the reputation system. However, for the peers using reputation system, the number of failed lookups increases with the increase in dynamism. The increase is gradual and number is still not significant if compared to peers without reputation system. With more dynamism, new nodes join over the time which may be malicious and the system takes time to learn the reputation. This is the reason why lookup failure rate increases slightly.

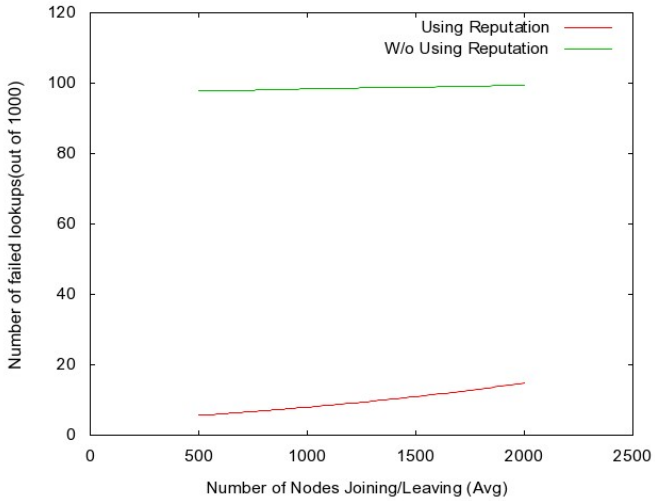

Figure 6.11. Reputation system behavior for different dynamic environment.

6.2.2 Effect of Damping Constant α

The effect of different α can be seen in Figure 6.12. With $\alpha = 0.0$, the reputation system is effectively not being used; since the current score value has no effect and the older value of 0.8 will remain forever. As α increases to 0.1, the number of failed lookups drop drastically to 23.89 from 93.71. Now the reputation system gathers

scores but is slow to adapt them to the environment. Thus, the number of failed lookups is still significant. For further increase in α, the number of failed lookups decreases further until it stabilizes over 0.6 and 0.7. The reputation system decreases the number of lookups by up to 90%. From 0.8, the number of failed lookups increases again. This seems logical as the system now relies too much on the latest result than on the previous reputation score. This shows that the value of α between 0.6 and 0.7 is best for the systems we tested. As expected, α has no effect for the peers without reputation system and number of failed lookups remain almost same.

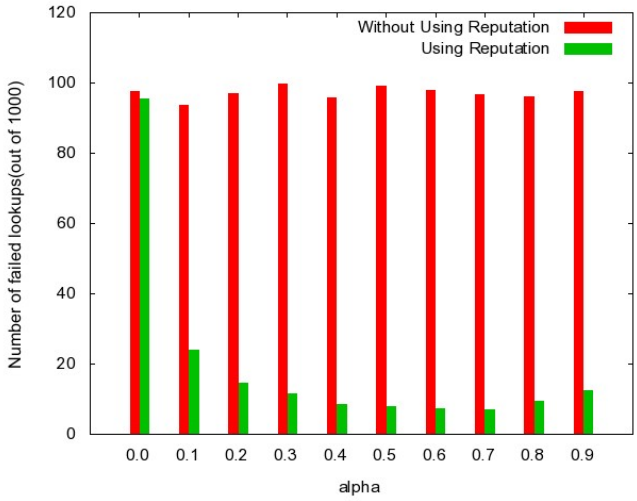

Figure 6.12. Effect Of Damping Constant α On Number Of Failed Lookups.

6.2.3 Reputation System For Different percentage of Attackers

The effect of different α can be seen in Figure 6.13. For very small percentage of 5%, if we use reputation system then number of failed lookups is zero. As this percentage increases, the number of failed lookups increases exponentially for the peers without reputation system. For 25%, we get 150 failures out of 1000 lookups and for 30% it becomes almost 220 failures out of 1000 lookups. When the reputation

system is used, there is still an increase in the failures with an increase in malicious node percentage. With 5% increase in malicious node percentage, the number of failed lookup increases by 37%. As for 30% malicious nodes, the number of failed lookups is almost 60 which is still 77% less than the number of failed lookups without using reputation system. This shows that even for peers with higher percentage of malicious nodes we can use reputation system to keep failed lookups as low as possible.

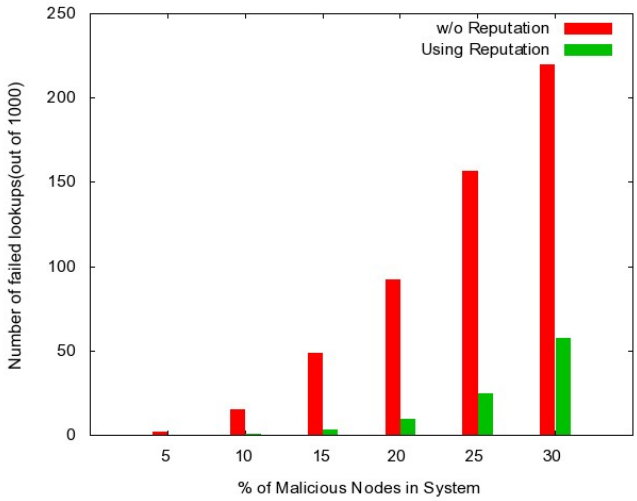

Figure 6.13. Reputation System For Different % of Attackers.

6.2.4 Reputation System For Different Redundancy Levels

The behavior of the reputation system for different lookup redundancy levels can be seen in Figure 6.14. The lookups are said to be failed, if none of the redundant lookups return a correct result. So if the node does more redundant lookups there are fewer chances of all the results being failed. Thus, from the graph we can see that for the peers with or without a reputation system, the number of failed lookups decreases with an increase in redundancy from four to nine. The proposed reputation system makes use of these redundant lookups to build reputations. So with an increase in

redundancy, system can collect more reputation information. As we can see for high redundancy of eight and nine,a peer using the reputation system has almost zero failed lookups.

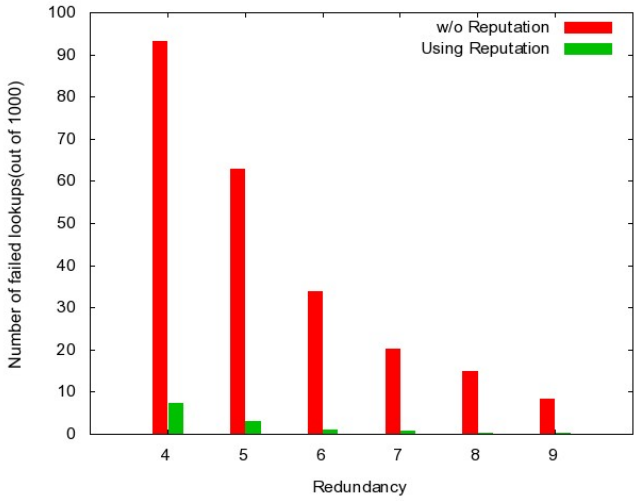

Figure 6.14. Number Of Failed Lookups For Different Redundancy Levels.

6.3 Distance-based Reputation Calculation

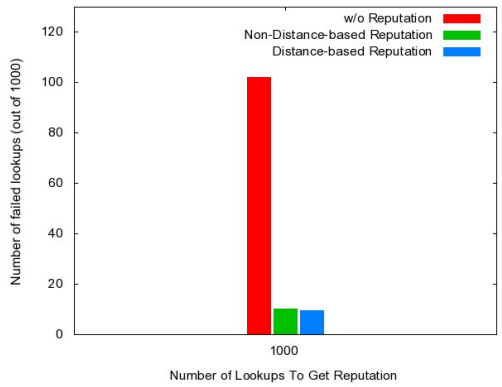

Figure 6.15. Effect of Distance-based Reputation Scores.

The effect of Distance-based reputation scores can be observed in Figure 6.15. We run 1000 lookups and take readings of 100 systems to get a good estimate of the number of failed lookups. Without using reputation system, we can see that the number of failed lookups is almost 100. In non-distance-based reputation system, we assign the reputation score one when all the redundant lookup results are same. Then the number of failed lookups is 10.26. When distance-based reputation scores are used, the number of failed lookups is 9.5. This is a reduction of 7%.

CHAPTER 7

CONCLUSION

Structured peer-to-peer systems are distributed communication systems popular for their efficient lookups and scalability even for highly dynamic environment. Even a small fraction of malicious nodes can bias the lookup results when they are present on a lookup path. In this thesis, we investigated this problem of reliably searching the insecure p2p networks. We proposed a reputation system to reduce the number of failed lookups and make the networks more robust. We studied our proposed work for the Salsa peer-to-peer communication system. First, we proposed a design for the static environment and showed the effectiveness of our system. We then extended the concept for dynamic environment. We showed that our system still has a high lookups success rate in a dynamic environment where there are many nodes joining and leaving, without giving advantages to the attackers. We discussed the design challenges and decisions of the algorithms proposed for determining reputation scores and tested them in simulation. We showed that the proposed system increases the lookup success rate by upto 90%.

APPENDIX A

BAYES' THEORM

In probability theory, Bayes' formula or theorm is used to show relationship between conditional probability and marginal/prior probabilities of events X and Y. This is especially useful to calculate probability of hypothesis based on the observed evidence.

The Byes theorm is:

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)} \tag{A.1}$$

where

P(X) = prior/marginal probability of X.

P(Y) = prior/marginal probability of Y.

$P(X|Y)$ = Posterior probability that is conditional probability of X given Y.

$P(Y|X)$ = conditional probability of Y given X.

REFERENCES

[1] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," p2p, pp.0099, First International Conference on Peer-to-Peer Computing (P2P'01), 2001.

[2] http://www.skype.com/

[3] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of SIGCOMM, pages 161172, 2001.

[4] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy,S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In Proceedings of 2005 ACM SIGCOMM Conference, 2005.

[5]F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP 01), Chateau Lake Louise,Banff, Canada, Oct. 2001.

[6] A. Kapadia and N. Triandopoulos. Halo: High-assurance locate for distributed hash tables. In C. Cowan and G. Vigna, editors, Network and Distributed System Security Symposium, Feb. 2008.

[7]M. Sanchez, P. Garcia, J. Pujol, A. Skarkemta, Cyclone: A novel design schema for hierarchical DHTs, in: Proc. Fifth IEEE Int. Conf. Peer-to-Peer Computing, P2P 2005, 2005.

[8] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, in ACM SIGCOMM 2001, San Diego, CA, September 2001.

[9]B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D.Joseph, and J. Kubiatowicz. Tapestry: A resilient globalscale overlay for service deployment. IEEE Journal on Selected Areas in Communications, January 2004.

[10]Beverly Yang and Hector Garcia-Molina, Designing a super-peer network, Proceedings of the 19th International Conference on Data Engineering (2003)

[11] Y. Yang, Q. Feng, Y. L. Sun, and Y. Dai. Reptrap: a novel attack on feedback-based reputation systems. In SecureComm 08: Proceedings of the 4th international conference on Security and privacy in communication netowrks, 2008

[12] J. Douceur, The Sybil attack, in Proceedings of IPTPS, Mar 2002.

[13] Y. Sun, Z. Han, W. Yu, and K. J. R. Liu. A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks. In Proceedings of IEEE INFOCOM, April 2006.

[14] A. Nambiar and M. Wright, *Salsa: A Structured Approach to Large-Scale Anonymity*, in Proceedings of CCS 2006, October 2006.

[15] Brent Lagesse, Mohan Kumar and Matthew Wright, *AREX: An Adaptive System for Secure Resource Access in Mobile P2P Systems*, p2p,, 2008 Eighth International Conference on Peer-to-Peer Computing, 2008.

[16] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: The second-generation onion router*, in Proceedings of the 13th USENIX Security Symposium, August 2004.

[17] A. Rowstron and P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in Proceedings of Middleware, Heidelberg, Germany, Nov 2001.

[18] Freedman, M. J., Freudenthal, E., and Mazires, D. Democratizing Content Publication with Coral. In Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI 2004) (San Francisco, Mar. 2004).

[19] S.M.Khan, MS Thesis, "The Dynamics of Salsa: A Structured Approach to Large-Scale Anonymity"

BIOGRAPHICAL STATEMENT

Apurv Ashok Dhadphale was born in Pune, India, in 1984. He received his B.E. degree in Computer Science and Engineering from University of Pune, India, in 2005 and his M.S. degree with a Thesis from the University of Texas at Arlington in 2009 in Computer Science. He has been part of the Information Security (iSec) Lab at UTA since January 2008. His research interest includes Computer Networks, Information privacy and Security, Data Mining and Designing Algorithm.