THE DYNAMICS OF SALSA: A STRUCTURED APPROACH TO LARGE-SCALE ANONYMITY

by

SAFWAN MAHMUD KHAN

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
August 2008

To my wife Farjana Sikder Jemy, my father Nesar Uddin Khan and mother Sufia Khanam - 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 this research project and pursue my thesis in Information Security. It was an excellent experience to do research under his guidance where I learnt a lot from him. As a mentor, he is a wonderful person. I also wish to express my gratefulness for him for providing me financial support for my studies. Dr. Wright is one of the most respectful persons in my life and will always be the inspiration of my any future success.

I would like to thank Dr. Nikola Stojanovic and Dr. Gergely Zaruba for their interest in my research and being part of my thesis committee.

I will miss my days in iSec lab at UTA where I spent a memorable part of my life with the members of the lab. I wish to give special thanks to Tara Mallesh for her continuous cooperation in my research work. I also want to show appreciation to Bikas Gurung and Qi Dong for their friendly accompany and cooperation.

Most importantly I would like to thank my parents and my wife. My parents' dreams inspired me to come so far till now. Their high ambition taught me to think always for the bigger. My wife, without whom it would be impossible to overcome huge mental stress here in my US life, is always my best accompany and inspiration behind everything. I wish to give her big thanks for helping me in my thesis writing. I also want to mention my younger brother Sheedy and elder sister Tonni's names for their support. I am always grateful to these persons for being with me in all my ups and downs of my life.

July 16, 2008

ABSTRACT

THE DYNAMICS OF SALSA: A STRUCTURED APPROACH TO LARGE-SCALE ANONYMITY

SAFWAN MAHMUD KHAN, M.S.

The University of Texas at Arlington, 2008

Supervising Professor: DR. MATTHEW WRIGHT

Anonymity provides a technical solution for protecting one's online privacy. Highly distributed peer-to-peer (P2P) anonymous systems should have better distribution of trust and more scalability than centralized approaches, but existing systems are vulnerable to attacks as they require nodes to have global knowledge of the system. To overcome these problems and to provide more secure, distributed organization for P2P anonymity systems, prior work proposed the Salsa system. Salsa is designed to select nodes to be used in anonymous circuits randomly from the full set of nodes, even though each node has knowledge of only a small subset of the network. It uses randomness, redundancy and bounds checking while performing lookups to prevent malicious nodes from returning false information without detection. In this thesis, we further investigate the dynamics of the Salsa system and propose to handle important system-level functionalities without giving advantages to attackers. We propose algorithms for joining and leaving of a node, splitting a group when its size reaches a maximum threshold, merging of two groups when a group's size reaches a minimum threshold and updating global contacts of nodes locally. We have introduced more randomness in the lookup procedure and made bounds checking more flexible. Finally, we implemented these dynamic events and developed a complete continuous time simulator for Salsa. Using this simulator, we present simulation results that show that still Salsa continues to have good lookup success in a dynamic environment with modest overheads in the system. These results also demonstrate the stability of Salsa in the presence of many peers joining and leaving.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT v
LIST OF FIGURES x
LIST OF TABLES xi

Chapter

1. INTRODUCTION 1
    1.1 Privacy 1
    1.2 Anonymity 2
    1.3 Our contribution to anonymous systems 3
    1.4 Organization 4
2. BACKGROUND 5
    2.1 Single-proxy Systems 5
    2.2 Mix-based Systems 6
        2.2.1 Tor 6
    2.3 Peer-to-peer Anonymous Systems 7
        2.3.1 Crowds 7
        2.3.2 Tarzan 8
        2.3.3 MorphMix 8
    2.4 Attacks on P2P anonymous Systems 9
        2.4.1 Intersection attack 9
        2.4.2 Control of the initiator's circuit 10
        2.4.3 End-to-end timing analysis attack 10

2.4.4 Predecessor attack 11

2.4.5 Denial of Service attack 11

2.4.6 Sybil attack 12

2.5 Distributed Hash Table 12

2.6 Chord 13

3. DESIGN OF SALSA: A STRUCTURED APPROACHED TO LARGE-SCALE ANONYMITY 15

3.1 Salsa Architecture 15

3.2 Lookup procedure, ***L*** 17

4. IMPROVEMENTS TO SALSA 20

4.1 Some initial Improvements 20

4.2 Dynamics of Salsa 21

4.2.1 Joining of a node, ***J*** 21

4.2.2 Leaving/Exit of a node, ***E*** 25

4.2.3 Updating the global contacts, ***UG*** 26

4.2.4 Splitting a Group, ***SG*** 26

4.2.5 Merging two Groups, ***MG*** 27

5. SIMULATION DESIGN 29

5.1 Static system design 29

5.2 Continuous time simulation design 29

6. RESULTS 33

6.1 Improved lookup result in static systems 33

6.2 Group population distribution 33

6.3 Success rate of lookup in dynamic systems 35

6.4 Message Statistics 35

6.5 False and True Positives and Negatives 40

6.6 Attempts per successful lookup 42
7. CONCLUSION 45
REFERENCES 46
BIOGRAPHICAL STATEMENT 48

LIST OF FIGURES

Figure Page

3.1 Logical organization of Salsa 16

5.1 Flowchart of continuous time simulation design 32

6.1 Percentage of failed lookups for different redundancy levels in static system 34

6.2 Distribution of group population in Salsa for 256 groups and 10,000 nodes 34

6.3 Lookup success rate in dynamic system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 36

6.4 Message overhead in dynamic system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 37

6.5 Message per lookup in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 38

6.6 Message per join in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 38

6.7 Message per leave in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 39

6.8 Message per mergein the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels 39

LIST OF TABLES

Table Page

6.1 False and True Positives and Negatives for 20% malicious nodes with α=2 41

6.2 Attempts per successful lookup for 20% malicious nodes and α=1 for different redundancy levels 43

6.3 Attempts per successful lookup for 20% malicious nodes and α=2 for different redundancy levels 44

CHAPTER 1

INTRODUCTION

Prior work proposed Salsa for large scale peer-to-peer anonymous systems to overcome some problems found in other peer-to-peer anonymous systems and to make these systems more secure and distributed. In this thesis, we further investigate the dynamics of the Salsa system and propose to handle important system-level functionalities without giving advantages to attackers. We propose algorithms for joining and leaving of a node, splitting a group when its size reaches a maximum threshold, merging of two groups when a group's size reaches a minimum threshold and updating global contacts of nodes locally. We implement these events and evaluate the performance of the Salsa system in a dynamic environment. Before proceeding further, we first discuss some basic issues in this chapter related to this thesis.

1.1 Privacy

The word *privacy* describes many concerns in the modern world. Privacy is a personal and subjective condition. One can not decide on what the level of privacy should be for others. Privacy can simply be defined as the right to be left alone. According to the famous actor Marlon Brando, "Privacy is not something that I'm merely entitled to, it's an absolute prerequisite." People need different types of privacy for different reasons. Here our main concentration is on information privacy, the ability of an individual or group to keep information about themselves from becoming known to people other than those to whom they choose to give the information. In today's world of Internet, where there is information all around and many malicious entities are trying to get that, it is very difficult to protect privacy.

1.2 Anonymity

Anonymity is a technical solution to provide online privacy. These days, Internet has become so ubiquitous that it has become the most widespread target for attackers. They might be able get a lot of information observing activities over Internet. Thus people need anonymity to get privacy , e.g.:

- Socially sensitive communications, such as to discuss about some diseases
- Corporations to hide sensitive and important business deals
- Political activists to discuss political issues or plan rallies
- Law enforcement authorities to observe criminal and terrorist websites
- Government to hide negotiations or procurement patterns

Encryption might be useful to protect sensitive information in storage or while transfer. But it is not good enough as attackers may get the encryption keys and thus decrypt the information. Anonymity provides one more level of protection as if the attackers were able to decrypt the information, it would be very difficult for them to find out the source or destination of that information.

Most Web users want the personal information they share to not be shared with anyone else without their permission. An annual survey conducted by the Graphics, Visualization and Usability Center of the Georgia Institute of Technology showed that 70% of the Web users surveyed cited concerns about privacy as the main reason for not registering information with Web sites. 86% indicated that they wanted to be able to control their personal information [10]. So privacy over Internet has become a critical issue. A major obstacle to privacy in Internet is that it uses IP addresses to route information over networks. These addresses can be used to track who is sending and who is receiving the information. This type of tracking traffic of information over the Internet is called *traffic analysis*. Through traffic analysis attackers can track back the traffic's source or destination that leads to a privacy breach. Therefore to achieve anonymity, our goal is to hide initiator's IP address. One way to do this is by sending messages through proxies. Proxies are used to carry information from source to destination creating some levels of indirection in the communication. In Tor [3], initiator uses three proxies between itself and the receiver to achieve anonymity. Anonymity can be provided over the Internet for browsing as a client to sever systems. In this research work, we concentrate on peer-to-peer (P2P) system as an application for anonymity.

1.3 Our contribution to anonymous systems

Current deployed anonymous systems use a relatively small set of proxies or servers to forward messages and provide anonymity [3, 7]. Thus they suffer from scalability problems and the proxies can be targets of the attackers. Some systems provide anonymity over peer-to-peer systems, potentially using more proxies [4, 5]. In [15], the authors argue that more users or peers ensure more scalability. In the P2P systems anyone can act as a proxy as well as a sender or receiver of messages, thereby ensures more scalability and possibly more anonymity. However, existing P2P systems have serious security vulnerabilities. To address these vulnerabilities, previous work proposed Salsa, a structured approach to organizing highly distributed P2P anonymous communication systems for scalability and security [14]. Salsa is designed to select nodes to be used in anonymous paths randomly from the full set of nodes available, even though each node has knowledge of only a small subset of the network. It uses randomness, redundancy and bounds checking while performing lookups to prevent malicious nodes from returning false information without detection. In this thesis, we look into the dynamics of the Salsa system. We propose algorithms for all the dynamics of the system without giving advantages to the attackers and implement those. We also introduce more randomness in lookups and make bounds checking more flexible. We show that even in the dynamic environment of the systems where there are many nodes leaving and joining, Salsa is still stable and gives a good success rate for lookups helping to establish strong anonymity.

1.4 Organization

In Chapter 2, we discuss the background concepts and related works to Salsa to justify our research work described in the later chapters. We present the basic design of the Salsa system in Chapter 3. Then we explain in Chapter 4 separate system changes from simulations perspective on Salsa in details. In Chapter 5, we describe the simulation design and the setup of the systems that we tested. We present and discuss the results of our simulation in Chapter 6. Chapter 7 concludes the thesis.

CHAPTER 2

BACKGROUND

In this chapter, we present some related works on anonymity and describe their techniques briefly to help us explain the design concepts of Salsa in Chapter 3. We also introduce some nomenclature that will be used throughout the rest of the thesis. We call the node attempting to send a message, e.g. an email or a HTTP message, the *initiator* and the intended recipient of that message the *responder*. 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. There are two categories of attacks, *active* and *passive*. An *active attack* attempts to alter system resources or affect their operation. A *passive attack* attempts to learn or make use of information from the system but does not affect system resources. A *malicious* node could be an active attacker or a passive attacker. Whatever it is, still the fact is that it is malicious which main goal is to learn information and then make use of that.

2.1 Single-proxy Systems

The most basic approach to achieve anonymity is to interpose an additional party, a *proxy*, between the initiator and responder to hide the initiator's identity from the responder. Examples of such systems may include the Anonymizer *(http://www.anonymizer.com)* and the Lucent Personalized Web Assistant *(http://lpwa.com)*.The proxy may introduce some delay and reordering of the messages to make it harder to correlate between the incoming and outgoing messages for traffic analysis. As with any other centralized system, it has a number of disadvantages, such as being single point of failure. Another problem is that the user has to trust the single proxy, as it sees all the traffic going through it. An insider attack is trivial for these systems. Also, this type of system is entirely vulnerable to an attacker who is in control of the proxy, as he can monitor and record all the communications through it. Denial-of-Service (DoS) attacks are also feasible.

2.2 Mix-based Systems

Single-proxy systems add only one level of indirection between the initiator and the responder. In 1981, David Chaum proposed the concept of a Mix-based system [1], based on which later several systems have been built up to provide anonymity. Examples of this type of system include Tor [3], Freedom [7] and Onion-routing [11]. The general approach to all of these systems is to build a virtual circuit using several proxies between the initiator and the responder. Based on the chosen circuit, first the initiator encrypts message for responder, then for the preceding proxy or onion router on the route and so on back to the first onion router to whom he will send the message, like the layers of an onion. This is called layered encryption. To protect against traffic analysis attacks, a mix-based system employs fixed-length messages and layered encryption of the messages. In addition, the proxies in the systems delay and reorder incoming messages and use cover traffic to hide real messages. Some systems are careful to process each message only once to counter replay-attacks.

2.2.1 Tor

Tor [3] works on real-world Internet with an international network. In Tor, the initiator chooses a path or route through network and builds a virtual circuit of some proxies in which each proxy, called an *onion router*, in the circuit knows only its successor and predecessor, but no other nodes in the circuit. Tor offers initiator and relationship anonymity. The onion routers can be chosen over several domains to make the task of traffic analysis harder like diverse routing [9]. Many streams can share one circuit so that there are not too many circuits and a malicious node's chance comes down to take part on several circuits to have more knowledge about the system that could be used to help him break the anonymity produced by the system. Tor is not secure against end-to-end timing attacks and also does not provide defense against DoS attacks. The onion routers themselves can be targets of direct attacks. And as usual with other mix-based systems, it suffers from scalability problem.

2.3 Peer-to-peer Anonymous Systems

As the number of mixes is relatively small compared to the number of users, mix-based systems suffer greatly from scalability issues. Also these mixes are usually advertised and well-known and so are open targets for attackers. Legal attacks are another possibility here, where some governments do not like the idea of anonymity over Internet and so prohibit institutions from operating mixes. To overcome these issues, peer-to-peer (P2P) anonymous systems can helpfully extend the idea of mix-based system into a P2P environment. In a P2P-based system, there is no distinction between a user and a proxy as all participants are equal peers. These systems should have a better distribution of trust and more scalability than the centralized or mix-based approaches. Current P2P anonymous systems include Tarzan [4], Crowds [5] and MorphMix [6].

2.3.1 Crowds

Crowds [5] provides anonymity for web transactions. Crowds, named for the notion of *blending into a crowd*, operates by grouping users into a large and geographically diverse group, called *crowd*, that collectively issues requests on behalf of its members. It provides anonymity for the initiator. Crowds does not use any cover traffic to make it more efficient. To join the system, a *jondo* (the user or node) contacts a server, called the *blender*, to request membership in the crowd. Since the joining procedure is centralized, the system suffers from a number of drawbacks of a centralized approach, such as the single point of failure and the fact that the blender sees all nodes' joining. They also create a virtual circuit of jondos, where each jondo chooses randomly and independently the next jondo from the entire set of crowd nodes, since they all are equal peers. So a malicious node gets the list of all other nodes in a crowd when it joins through blender. Crowds uses a static circuit for a user so that collaborating jondos can not get more information being on different paths if dynamic paths were being used.

2.3.2 Tarzan

Tarzan [4] is a P2P anonymous IP network overlay. It achieves anonymity with layered encryption and multi-hop routing. It provides anonymity to either client or server. It is the first anonymizing communication system that is both self-organizing and fully decentralized. A Tarzan node initially only knows a few other nodes, but later, through a *gossiping* protocol, it learns about all other nodes in the system. A big drawback of this system is that every node has this global knowledge of the system.

2.3.3 MorphMix

MorphMix [6] is for anonymous Internet usage. It helps to protect the initiator's privacy while using Internet. Here the initiator only selects the first intermediate node of the circuit and each node along the circuit chooses the following nodes. Therefore one node does not need to know all other nodes in the system, which is one of most important features of the system as it becomes very difficult for the malicious nodes to get global knowledge of the system to conduct attack. Also it uses the idea of *witness* for selecting the intermediate nodes, except the first one, to prevent the selection of malicious nodes in a circuit by one malicious node. With the help of witnesses they propose a collusion detection mechanism for detecting compromised circuits by malicious nodes. However recent research [12] shows that witnesses can be deceived by intelligent selection of nodes and so MorphMix is vulnerable to collusion attacks. The creator of MorphMix argues that the large number of mixes and the dynamism of the system help the system to protect against traffic analysis attacks without using cover traffic, making the system more efficient than Tarzan. On the other hand, all the mechanism used for collusion detection limits the scalability of the system to some extent. Also MorphMix is better-suited for short-lived activities like web-browsing than long-lived activities.

2.4 Attacks on P2P anonymous Systems

There are several attacks observed on P2P anonymous systems. Here we discuss some of them which might also be applicable for the Salsa system. In this section, we briefly present the basic mechanism of these attacks and analyze them on different systems.

2.4.1 Intersection attack

The basic premise of the intersection attack is that some users will communicate with certain websites frequently over time. If there are cookies or user IDs for these websites, they can be used to identify a particular user over time. This gives the attackers an advantage of long-term intersection attacks. In anonymous systems, the attacker repeatedly takes snapshots of the nodes that send messages and the nodes that receive messages over a period of time, say at times t_1, t_2, $t_3....t_n$. Then he tries to find correlations among those initiators and responders by intersecting the set of nodes they collected over that time period. The times at which some nodes receive messages will correspond to the times at which the initiators send messages. Some other nodes are not sending messages. Thus, the remaining nodes in the intersected set are more likely to be initiators corresponding to those responders than other nodes, thus reducing the degree of anonymity of these possible initiators. More the nodes the attackers can observe, the greater the ability to perform the intersection attack. One general approach to protect this attack is to introduce cover traffic.

Tor and Tarzan require nodes to have global knowledge of the system which makes these systems vulnerable to intersection attacks. To prevent this attack to some extent, Tarzan uses cover traffic using its proposed technique *traffic mimic*. Crowds gives out list of users of a crowd to a node when it joins the system. Thus collaborating malicious nodes can collect information about many nodes in the system and thus vulnerable to this attack. MorphMix avoids giving global knowledge to avoid this attack. But still breaking their collusion detection mechanism, attackers may get some advantage to do this attack.

2.4.2 Control of the initiator's circuit

In P2P anonymous systems, if the attackers can control all proxies of the virtual circuit of an initiator, they can easily find the initiator and thus can break initiator's anonymity. This type of attack depends on how the system builds its virtual circuits. If the initiator chooses the first proxy and from there on the next proxies are chosen by the succeeding proxies instead of the initiator's itself, the chance of this attack is enhanced.

Crowds is very vulnerable to this attack, as on a path each jondo chooses the next jondo independently. So if the initiator's first jondo is a malicious one, it controls the whole circuit. MorphMix tries to avoid this attack by using a collusion detection mechanism using witnesses. But as we discussed in Section 2.3.1, these witnesses can be deceived and thus the circuit can be controlled. So MorphMix is also vulnerable to this attack.

2.4.3 End-to-end timing analysis attack

If an attacker can watch patterns of traffic at the initiator and the responder, he will be able to confirm the correspondence between them with higher probability.

So to be successful to perform this attack, the attacker should be able to control or observe the first and last proxies of a circuit. The greatest protection to defend this attack is to hide relationship between the first proxy and the initiator, and the last proxy and the responder. Cover traffic may be a good solution for this kind of attack.

Tarzan uses cover traffic to protect this attack [4]. MorphMix does not use cover traffic but the large number of proxies and dynamism of MorphMix helps to minimize this attack.

2.4.4 Predecessor attack

Crowds [5] introduced the idea of this attack and it was extended in [13]. In this attack, the attacker tracks an identifiable stream of communications over a number of paths or circuits. Each time a circuit is created, the attacker logs any node that initiates a message that is part of that identifiable stream. Thus, the attacker takes part in different paths over time to observe the predecessors on the path of proxies and thus link the initiator and the responder. This attack exploits the process of circuit initialization. Timing analysis may also speed up the attack.

Protecting traffic analysis mechanisms [4, 6] employed by Tarzan and MorphMix can also slow down this attack. To protect against this attack, Crowds uses a static path for a user so that collaborating jondos can not get more information being on different paths if dynamic paths were used.

2.4.5 Denial of Service attack

This is a common attack in P2P anonymous systems. Any malicious node within the system may refuse to provide service. Also it may try to generate an excessive volume of messages intentionally to overload the system, thus hampering the service. Both attacks are detectable and curable to some extent. However, whenever the number of malicious nodes grows, this attack can become more severe. Most P2P anonymous systems do not try to protect this attack aggressively, as attackers with lots of resources can always flood the system.

Crowds says that they do not defend against DoS attack as some malicious nodes in a crowd may refuse to forward a message [5]. Tor proposes mechanism like onion routers require clients to solve a puzzle while beginning a new connection to protect this attack.

2.4.6 Sybil attack

A significant attack in P2P systems is the Sybil attack [8]. The basis of this attack is that one malicious entity may have multiple identities. For example, one aatacker may have several IP addresses in a system, i.e. he may own several nodes. Using this attack, at relatively low cost, at attacker may construct a lot of identities and use them to manipulate or undermine the system. The existence of *botnets*, in which the attacker controls a large number of corrupted nodes widely distributed in the Internet, makes the Sybil attack as especially dangerous threat.

To defend this attack, Tarzan proposes that the period to compromise all circuit relays must be longer than the circuit's duration and circuits should not be repeatedly constructed through the same small set of largely-compromised proxies [4]. MorphMix uses *hierarchical address selection* to select nodes as proxies at random from different subnets to minimize the Sybil attack. In [9] the authors describe some techniques to prevent Sybil attacks. They use some trust metrics based on social networks with the help of a distributed hash table. Their main technique is to use diverse set of nodes for lookup and they propose *diversity routing*, *mix routing* and *zig-zag routing.* Recently [16] proposes a novel protocol *SybliLimit*, a near-optimal defense against Sybil attacks using social networks.

2.5 Distributed Hash Table

A *hash table*is a data structure that associates 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. Distributed hash table (DHT) provides the same 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. Its main features include:

- Decentralization: The nodes collectively form the whole system without any central coordination, though every node has limited knowledge about the system.
- Scalability: As everything is done locally, this is largely scalable. Adding a node in the system causes relatively little activity as a whole as most other nodes in the system are not affected.
- Fault Tolerance: As it is decentralized, there is no single point of failure and also some failures or departures of nodes in the system do not affect most of the other remaining nodes.

All these attractive features inspire to use DHT in the Salsa system.

2.6 Chord

Chord [2] 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. With high probability, it balances load over all the nodes in the system. It lets nodes to join and leave the network with minimal disruption. It provides weak form of authentication. The most significant application of consistent hashing has been to form the foundation of DHTs as described in 2.5. As a part of the Chord DHT, each node maintains a routing table, called the *finger table*. Chord improves the scalability of consistent hashing by requiring a node to store only a small amount of routing information about other nodes in the finger table, not all the nodes in the system. Salsa has some features similar to Chord.

CHAPTER 3

To overcome the drawbacks in current P2P anonymous systems we described in Chapter 2 and to provide more secure, distributed organization for P2P anonymity systems, prior work proposed a novel structured overlay design called *Salsa* [14]. Salsa is designed to select nodes to be used in anonymous circuits randomly from the full set of nodes, even though each node has knowledge of only a small subset of the network. It uses randomness, redundancy and bounds checking while performing lookups to prevent malicious nodes from returning false information without detection.

3.1 Salsa Architecture

Salsa uses a Chord-like ID-space based on the cryptographic hash of the node's IP address to make it a fully distributed system for their several advantages as described in Chapter 2. The nodes are placed in sorted order around a circular ID-space and each node N_i is said to own the fraction of the ID-space between itself and it's preceding node N_{i-1}. If a node requests a specific ID, say χ which is called the *target* ID, which is between N_{i-1} and N_i, will be routed to N_i as it owns that ID-space. N_i is said to be the owner of that target.The system divides the ID-space into groups of contiguous IDs and the groups are conceptually organized as a virtual binary tree (Figure 3.1). A node belongs to a group if its ID is in that group's ID-space. For numbering the groups, Salsa uses a *group ID-space* that is separate from the node ID-space and probably 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).

Salsa nodes store routing information in a *local contact table* and a *global contact table*. The *local contact table* stores the IDs and owned ID space of all other nodes in the same group. 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

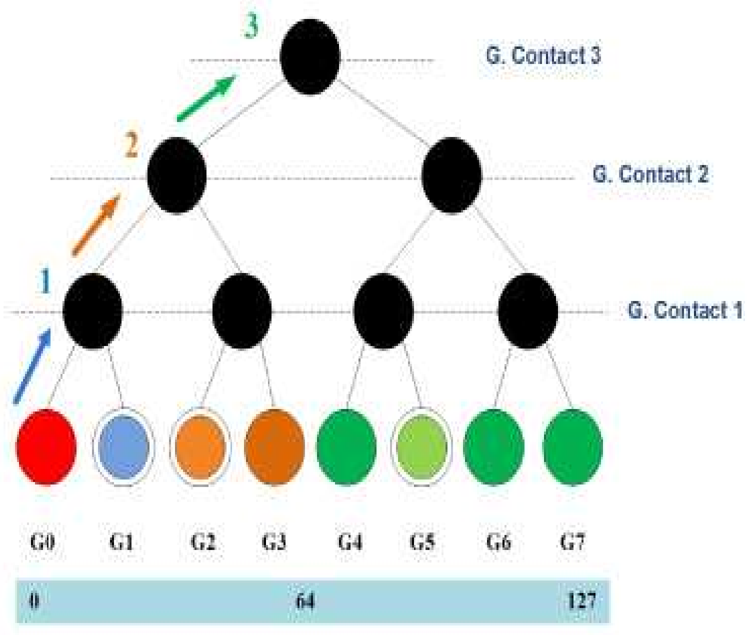

Figure 3.1. Logical organization of Salsa.

3.1. Say a node N is in group G0. It goes one level up in the tree and selects a global contact from the other sub-tree, from G1. For selecting a global contact, it uses the ID-space and selects a random ID from the space. For example, here for its first global contact, 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. Here N has selected a node from G2. It again goes one level up and selects its last global contact from the other sub-tree randomly, from G4-G7. Let us say that it has selected a node from G5. As the height of the tree is three, N will have three global contacts in its table. For each global contact, it will store the global contact ID and its corresponding owner node. This binary virtual tree structure with the global contacts makes the lookup process in Salsa very similar to binary search which we are going to explain in the next section.

3.2 Lookup procedure, *L*

To lookup a particular ID, Salsa uses redundancy and bounds check to protect against malicious nodes. Let S be the *query node*, meaning that it is looking for a target ID in ID-space and starts the lookup procedure. Let the redundancy level and bounds S uses are R and B respectively. 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. *A node, say N, starts the lookup.*
2. *N computes on which part of the Salsa virtual tree the target ID is.*
3. *N requests its global contact of that part of the tree to find out the owner of the target ID.*
4. *The global contact computes whether the target ID is in its own group or not.*
5. *If it is, it returns information of the owner of the target ID to N.*
6. *If it is not, the global contact itself acts as N now and do the same steps from 2 to 6 recursively until it finds the owner of the target ID. (Pseudo code in Algorithm 1)*

Redundant lookup:

1. *S selects a target ID T randomly from the ID-space that it is looking for.*
2. *S selects R neighbors of its group randomly from its local contact table.*
3. *S requests these R nodes to find the owner of T.*
4. *Each of the R nodes independently finds the owner of T through the Salsa virtual tree structure using the Algorithm 1.*
5. *S calculates the distance of these results from T and will select the node with the minimum distance. Let this node, N have ID* ID_N .
6. *S checks whether* $(ID_N - T) < B$.
7. *If it is, S accepts N as the owner of T.*
8. *Otherwise, S repeats steps from 1 through 9.*

```
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      middle ← (low + high)/2 + 1;
1.7      if key < middle then    /* requestedNode on lower subtree */
1.8          if target >= middle then          /* targetNode on higher
             (different) subtree */
1.9              intermediate ←
                 $SNode[requestedNode].GlobalFingerTable[l]$;
1.10             break;
1.11         end
1.12         high ← (middle- 1);
1.13     else                    /* requestedNode on higher subtree */
1.14         if target < middle then              /* targetNode on lower
             (different) subtree */
1.15             intermediate ←
                 $SNode[requestedNode].GlobalFingerTable[l]$;
1.16             break;
1.17         end
1.18         low ← middle;
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

A particularly nice property of consistent hashing that Salsa uses is that the target owner will be closer than any other node to the target ID. 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 procedure or activity 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 R. It is looking for target T in G6. S will request R of its neighbors in G0 to find the owner of T. Each of these R nodes will first check whether it is in the same group or not. If yes, it is done. If not, each of them will conduct the *recursive lookup* described earlier on the tree. At the end N will get R results back. If all these R nodes were honest and there were no malicious nodes on their way to lookup the owner of T, all were supposed to provide the same result, which is the closest one from T. Whatever these R results are, S will take the closest one and then do the bounds checking, the final investigation. For example, say bound B is 7, target ID T is 26 and the returned owners' IDs with redundancy 5 are 32, 32, 37, 32 and 37 where third and fifth results are returned by malicious nodes. S will select the closest one, 32, and find that the distance between T and this owner's ID is 6 which is inside the bound 7. So S will select it. But if the bound B were 5, then this owner's ID would fail to pass the bound as the distance 6 is more than the bound 5. In this case, S would conduct the whole 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, so it has to return same results as close to the target ID as possible. It is also possible that the target owner itself is malicious, in which case we expect to get the correct result.

CHAPTER 4

IMPROVEMENTS TO SALSA

In this Chapter we present modifications to the previously proposed Salsa system [14] and discuss their implications. We introduce more randomness in the lookup procedure and use flexible metrics for bounds checking. We mainly investigate the dynamics of the Salsa system and propose algorithms to handle important system-level functionalities without giving advantages to attackers.

4.1 Some initial Improvements

We introduce more randomness into the lookup procedure. While selecting some R number of redundant nodes from the local contact table for lookup, query node now does not choose the first R or the last R contacts from the table. Instead it chooses any R contacts randomly from the whole contact list. Previously for bounds checking, nodes were using a fraction of the group size as the bound. In some cases there were no nodes at all within the bound due to which many lookups failed the bounds checking. Also this bound was equal for all the nodes in a system. Now we use a flexible metric α for bounds checking for which the equation is:

Bounds_Checking_Distance= $\alpha \times$ (Group Range of the query node $\div$ No of local contacts of the query node)

α implies the nodes per bound a query node wants. So using this metric, we can avoid the situation where there might be no nodes at all in some cases. Also this equation is dependent on the group range and the number of local contacts of the query node. Therefore bounds checking will be different for different query nodes.

4.2 Dynamics of Salsa

Structured P2P systems like Salsa require significant coordination due to node churn. This churn includes nodes continuously joining and leaving the system over time. We now focus on the dynamics of the Salsa system. We describe algorithms for distributed management and system operations in the dynamic Salsa environment. We design algorithms to handle churn without giving advantages to attackers. In this Section we present these algorithms one by one in turn. We already described lookup procedure in Section 3.2, so we are not going to repeat it here.

4.2.1 Joining of a node, *J*

When a node wants to join the Salsa system, it first contacts a friend node that is already in the system. Alternatively the joining node may pick one from an advertised set of Salsa nodes. In either case the joining node can verify the chosen node and its global contacts by hashing each one's IP address and its level in the Salsa tree. Say the node that wants join is N. N will ask the chosen node to find out information such as which group it should join and what the group size and group local contacts are. Say N selects a friend. The friend will determine the joining node's ID in the ID-space by hashing its IP address. The friend will then find the owner of that ID through redundant lookups. The owner of the ID sends the correct group size and local contacts information if it is honest and if the joining node's ID is in the same group. The friend node does not depend only on the information from a single node as it might be malicious. It collects information from different nodes from different ranges around the joining node's ID. These redundant queries protect the friend node against biasing by the malicious nodes. Due to the uniform random nature of the hashing algorithm and based on some experimental results that we present in Section 6.2, a node can assume almost similar group size throughout the system. But as groups may split or merge by power of 2 over time, the friend node can assume the size of the group where N is joining is either the same size of its own group or the double or half of it. Based on this, the friend node will randomly choose three more IDs from three ranges around the joining node's ID - one assuming the same group size, one assuming half and one as double. The friend node will then find out the owners of these three IDs through redundant lookups and request each owner node to send the following information:

- Whether the joining node N's ID is in the same group or not.
- Its group size.
- Its local contact list.

Here we assume that all nodes return the correct information of the group size and contact lists and honestly say whether the joining node is in their groups or not. Note that, we are using redundant lookups to find out the owners of the IDs to avoid malicious results as many as possible, but we are not using bounds checking here as we do more checking to select information among the returned results later in the joining procedure. Also choosing the IDs randomly makes the malicious nodes' work harder. So at the end, the friend node has four sets of results getting from four different nodes. If the friend gets all *No* responses, i.e., all the four nodes say that the joining node is not in their groups, it will conduct an extra lookup by choosing the ID at exactly half of its own group size behind the joining node's ID. If the friend node still does not get any *Yes* response, it will repeat the above procedure once again for the joining node. If the repeated procedure does not result in a *Yes* response, the friend gives up. The joining node may ask another of his friends in Salsa or may go for another advertised node. It may try to join the system some time later too. If friend gets some positive responses from the contacted nodes, it will consider only the *Yes* messages, i.e. the results that say that the joining node is in the same group. The friend node takes the largest group size among the *Yes* responses so that if some results were returned by malicious nodes, they can not influence to make the joining node's group size smaller to get some advantages. The friend node will then concatenate all the local contact lists that responded *Yes*. It will also check whether the local contacts in a *No* response actually fall inside the selected group range and if so, will concatenate those results to the list of local contacts from the *Yes* responses. This is done as some malicious nodes may try to mislead by returning *No* responses, but we do not want to miss their contact lists. Then friend will send all the collected information to the joining node. The joining node now has the information necessary for it to join the Salsa system. Hereafter it builds up its global contact list according to Salsa tree structure. The joining node informs all of its local contacts of its joining information. Here we assume that when a node joins a group, *all* the members update their local contact lists and owned space information if needed. The joining procedure described above uses redundancy and randomness to protect the joining node from being misinformed by malicious nodes.Below we present the schema of the algorithm step by step. Below we present the schema of the algorithm step by step. Say the node wants to join is N.

- *Scenario 1: Friend Node*

  N knows a friend node F that is already in the Salsa system.

  1. *N asks F to find out the local contact list and the group range for N.*
     (a) *Determine location of N in the address space, say its* ***y****.*
     (b) *F knows its own group size* ***D*** *(for Delta).*
     (c) *F will find the owner of* ***y*** *say its* ***Y*** *through redundant lookup.*
     (d) ***Y*** *will also send its group ranges, say* ***GroupStart*** *and* ***GroupEnd****.*
     (e) *F will calculate the group size of* ***Y****,* ***GroupSize(Y) = GroupEnd - GroupStart****.*
  2. *Now F will calculate these three parameters:*

     A (for α) = y % 2D

     B (for β) = y % D

     G (for γ) = y % (D/2)
  3. *F will choose three different IDs from these ranges randomly:*
     (a) *Choose in between* ***(y - G)*** *and* ***(y - G + (D/2))***

   (b) *if **((y- B) ==(y-G))** Choose in between **(y - B + (D/2))** and **(y - B + D)** else Choose in between **(y - B)** and **(y - B + (D/2))***

   (c) *if **((y- A) ==(y-B))** Choose in between **(y - A + D)** and **(y - A + 2D)** else Choose in between **(y - A)** and **(y - A + D)***

4. *Say these three IDs (derived from step 3) are **y1**, **y2** and **y3** respectively. F will find the owners of these three IDs say, they are **Y1**, **Y2** and **Y3** respectively through redundant lookup.*

5. *F sends JOIN_MESSAGE to these 3 likely neighbors **Y1**, **Y2** and **Y3**.*

6. *Each of these three likely neighbors will do the following:*

   (a) *Checks if y belongs to its own group.*

   (b) *If so, whether its malicious or not, says "Yes" and sends F the list of local contacts in its group and also the range of the group; say (**z1**, **z2**).*

   (c) *If not, and whether its malicious or not, says "No" and sends F its own ID, the list of local contacts in its group and also the range of the group; say (**z1**, **z2**).*

7. *If F gets all "No " messages from these 3 nodes as well as **Y**, it will do an "extra" lookup, otherwise it will go to step 8:*

   (a) *It will choose the ID **(y-G)**, say its **y4**, in the case that **Y** is from other group and no one of **Y1**, **Y2** and **Y3** is in **y**'s group.*

   (b) *Now it will find out **y4**'s owner say **Y4** through redundant lookup.*

8. *Now F will do the following:*

   (a) *It will consider only the nodes which have replied with "Yes" message.*

   (b) *F will select the largest group size only among the "Yes" responses.*

   (c) *It will take the union of all nodes' local contact lists which responded "Yes".*

   (d) *Now F will check whether the nodes responded "No" are really in the selected range or not. If it is, F will also take its local contact list for N.*

   (e) *F will send the selected group range and collected local contact lists to N.*

9. *Now N sends JOIN_CONFIRMATION_MESSAGE to all of its local contacts.*

   (a) *All members in the group update their contact tables.*

10. *Now N :*

    (a) *Builds up its local contacts from the list that it got from F.*

    (b) *Builds up its global contacts according to Salsa structure.*

- *Scenario 2: Advertised Nodes*

  Subsets of some nodes in the Salsa are advertised with their IPs and global contacts

  1. *N will pick one from them as F.*
  2. *Verify its global contacts by hashing its IP and the level of the tree.*
  3. *Then it will do the same steps as the Scenario 1.*

4.2.2 Leaving/Exit of a node, *E*

For leaving, a node may leave the system silently or it may inform some or all other member nodes in its own group. So this procedure is local and does not require much coordination. It may affect the system in other way. Some malicious nodes may broadcast fake *LEAVE_MESSAGE* to perform a DoS attack. We can defend against it by using any traditional handshake protocol. The algorithm is as follows. Say the node wants to leave is N.

1. *First N will decide whether or not to inform its group members of its departures.*
2. *If N decides to leave silently, it will not go through the next steps and will just leave the system.*

3. *If it decides to inform others, N will inform its local group members with a LEAVE_MESSAGE.*
4. *All the group members who receive the LEAVE_MESSAGE will update their local contact list.*

4.2.3 Updating the global contacts, *UG*

Nodes do lookups through their global contacts. So it is necessary to update the list of global contacts over time. We have proposed a fully distributed algorithm for updating the global contacts. Each node locally updates its global contacts if it finds during a lookup that any of its global contacts is no longer active. The algorithm is as follows:

1. *While doing lookup, if a node finds that a global contact, that is, the owner of the corresponding global ID does not exist or no more alive in the system, it will do the following steps, otherwise will do nothing.*
2. *It will find the new owner of that global ID and update its global contact accordingly.*
3. *If it finds that the new owner of the global ID is from another sub-tree of the Salsa virtual tree structure which violates the property of the global contacts architecture, it will choose a new global ID from the same group or at least from the same sub-tree and find its owner.*

4.2.4 Splitting a Group, *SG*

After joining if a node finds that its group population, i.e. the number of nodes in its group, is more than a threshold value, it will invoke the procedure for splitting 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. Based on group population distribution simulation results, we have set the threshold value for splitting a group as 80 nodes. This is configurable for a system. If a node N finds its number of local contacts is more than 80, say 81, it will do the following steps:

1. *N will cut the size of its group range by half.*
2. *It will consider its own side as one group and other side as another.*
3. *N will find the number of nodes in its own side as well as the other side.*
4. *If both are greater than the threshold for merging a group N will split the group, otherwise it will not split the group.*
5. *If it splits the group, it will update its group size, local and global contacts according to the new tree structure.*

4.2.5 Merging two Groups, *MG*

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, based on group population distribution results, we have set the threshold value for merging groups to 20 nodes. This is a configurable parameter. If a node N finds its number of local contacts is less than 20, say 19, it will do the following steps:

1. *N broadcasts its local contact list to all members of its local contact list.*
2. *Each of these contacts compares the received list with their own local contact lists.*
3. *For each mismatch, the member node contacts the mismatched node to check if it is alive. If that node is alive, it broadcasts its ID to all the nodes in the group to inform its existence.*
4. *If any of the member nodes do not agree, N does not complete the merging procedure and stops here.*
5. *If all the member nodes of the contact list agree, N contacts its global contact in the sibling group with its local contact list to initiate a merge.*
6. *The global contact of N checks whether its group population exceeds the maximum threshold after merging. If not, it sends "Yes" to N with its local contact list else it sends "No".*
7. *If the global contact sends "Yes" to N, it broadcasts N's contact list and new group size information to all its group members.*
8. *If N receives "Yes", it broadcasts everyone of N's global contact's local contact list to update their contact lists and new group size.*
9. *Also all the members in these two groups will update their global contacts accordingly.*
10. *If N receives "No", the merging procedure is discontinued.*

CHAPTER 5

SIMULATION DESIGN

We performed a number of experiments to verify the effectiveness, stability and security of the Salsa system. Simulations include a static system design and a dynamic system implemented as a continuous time simulation.

5.1 Static system design

The static Salsa system has a fixed number of honest and malicious nodes. The number of groups is fixed. Dynamic node joining and leaving are not included in the static system simulation. The system was simulated in Java. We used a 30-bit ID-space, a 20-bit group address space and simulated systems consisting of 1000 and 10,000 nodes respectively. We used 32 and 64 groups for the 1000 node system and 128, 256 and 512 groups for the 10,000 nodes case. For each test, we simulated 100 separate systems and conducted 1000 lookups per system. . We used values 1 and 2 for α. We simulated the system under varying degrees of attack with the percentage of malicious nodes ranging from 0 to 20. We capped the tests at 20% malicious nodes to simulate a realistic scenario. Beyond 20% malicious nodes, we believe the attacker has substantial advantages regardless of the way the system is organized. We simulate the lookup procedure and compute some statistics of the system from the data collected. The results of the static system simulations are presented in Section 6.1 and Section 6.2.

5.2 Continuous time simulation design

The continuous time simulation implements a dynamic system in which nodes join and leave the system over time. The system was simulated in Java. We used a 30-bit ID-space, a 20-bit group address space and simulated systems consisting of 1000 and 10,000 nodes respectively. We used 32 and 64 groups for the 1000 node system and 128, 256 and 512 groups for the 10,000 nodes case. For each test, we simulated 100 separate systems. We ran each test until 1000 lookups were done in a particular system. We used values 1 and 2 for α. We primarily concentrate on the dynamics of the system in the worst case and therefore have used 20% malicious nodes for the continuous time simulation.

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., when one event occurs, no other events take place simultaneously. They are performed sequentially in order in which 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. Our goal is to find out the same statistics e.g. finding nodes through lookups that we did previously. However now the simulation includes all other events (a-j) mentioned above too. The events (a-j) were not simulated in the previous static simulation. We run the simulations and observe the behavior of the lookup procedure in the dynamic Salsa environment. We computed the overall statistics of the dynamic system and present them in Chapter 6. For the whole simulation we use a queue **Q**. We conduct 1000 lookups per system. For each system, the simulation runs until 1000 lookups are performed. A lookup event **L** is initiated 1 simulation minute after the last lookup, at a constant rate and is pushed into **Q**. Join events **J** occur randomly at rate λ_J with an exponential distribution. After **J** event is executed, the next **J** is generated and pushed into **Q**. Same as **J**, leave events **E** occur at random times with rate λ_E and follow an exponential distribution. After an **E** event is executed, the next **E** is generated and pushed into **Q**. Whenever there is at least one event in **Q**, we pop that from **Q** and obtain either an **L**, **J** or **E** event. The events are popped from the queue until **Q** becomes empty. While executing lookup **L** procedure, if a node finds its global contact is no longer active in the system, the node immediately updates that contact with a **UG** event. During the join procedure **J**, if a node finds that the group it is joining is too big or too small, it invokes the **SG** or **MG** procedures respectively. The **SG** or **MG** event is executed immediately after the join procedure is finished. Same as **J**, if a lookup **L** invokes **MG** event, it is executed immediately after the lookup procedure is finished. The whole design procedure is presented in Figure 5.1.

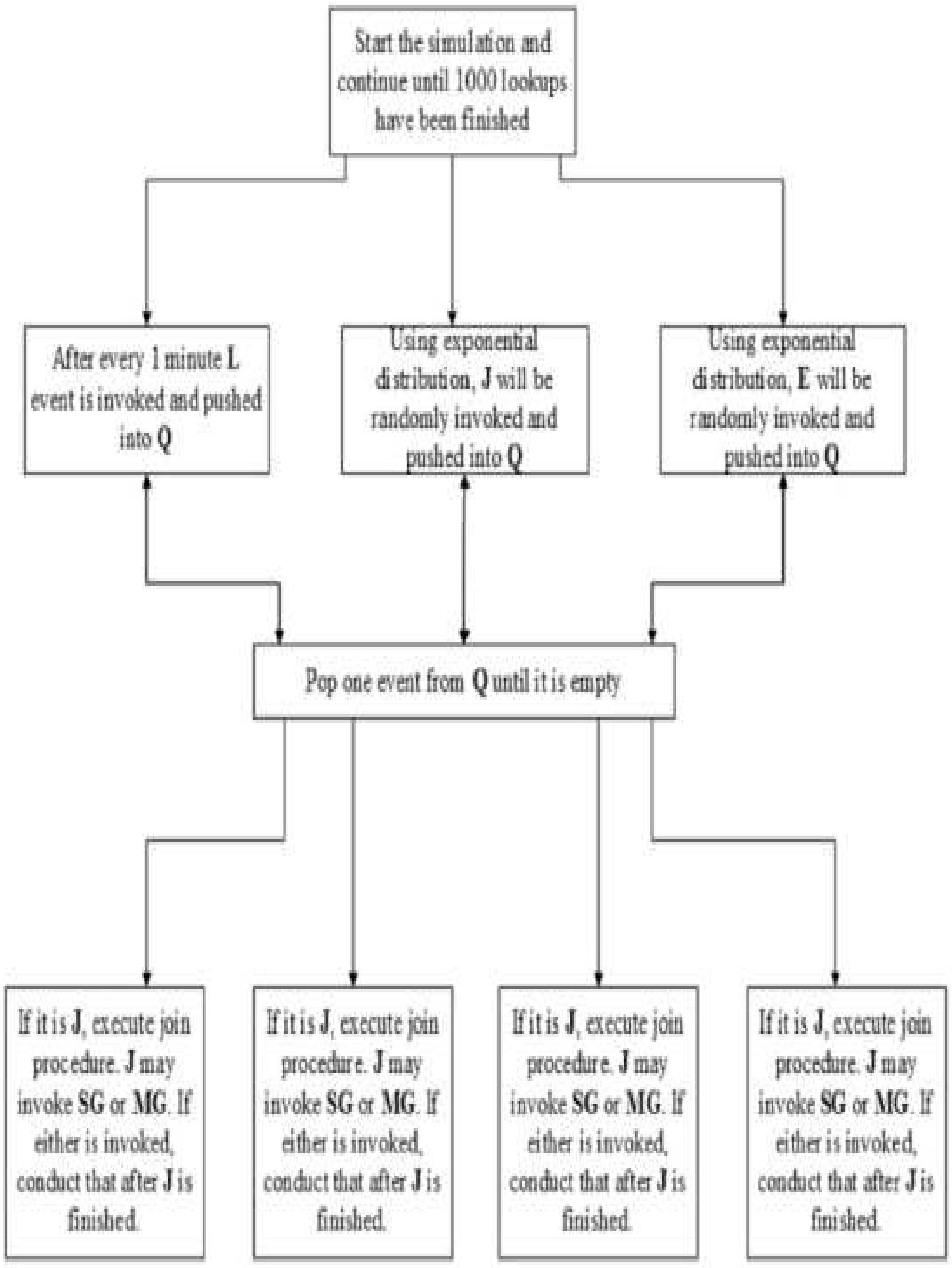
Start the simulation and continue until 1000 lookups have been finished
After every 1 minute L event is invoked and pushed into Q
Using exponential distribution, J will be randomly invoked and pushed into Q
Using exponential distribution, E will be randomly invoked and pushed into Q
Pop one event from Q until it is empty
If it is J, execute join procedure. J may invoke SG or MG. If either is invoked, conduct that after J is finished.
If it is J, execute join procedure. J may invoke SG or MG. If either is invoked, conduct that after J is finished.
If it is J, execute join procedure. J may invoke SG or MG. If either is invoked, conduct that after J is finished.
If it is J, execute join procedure. J may invoke SG or MG. If either is invoked, conduct that after J is finished.

Figure 5.1. Flowchart of continuous time simulation design.

CHAPTER 6

RESULTS

In this Chapter we present and discuss our simulation results.

6.1 Improved lookup result in static systems

As we discussed in Section 4.1, we use more randomness in lookup procedure and use more flexible bounds checks and measure the rate of unsuccessful lookups in the static system. The results are presented in Figure 6.1. Here, we clearly observe that with the increase in percentage of malicious nodes from 0 to 20, the lookup failure rate increases that is consistently closed to the percentage of malicious nodes in the system, as expected. In a system like Tarzan, it would be 20% at 20% attackers. We do slightly worse, such as for redundancy level 7 and 20% attackers the failure rate is 22.6%, which is acceptable. The results in Figure 6.1 also show the effect of using redundancy. If we increase the redundancy level from 5 to 7, the failure rate drops by approximately 2.61%. The value of α is set at 2 for this experiment and the number of groups and number of nodes are 256 and 10,000 respectively. So our redundancy and bounds checking are working fine.

6.2 Group population distribution

Before proceeding to work on the dynamics in Salsa we wanted to observe the usual distribution of Salsa nodes over the ID-space and find out the group population distribution statistics. In Figure 6.2, we see that for 10,000 nodes and 256 groups, most of the groups' populations are near 40. 40 is the maximum and 39 the second highest. Few groups have populations of 38, 41 and 42. Thus the population does not vary much and nodes can assume almost similar group size over the whole system.

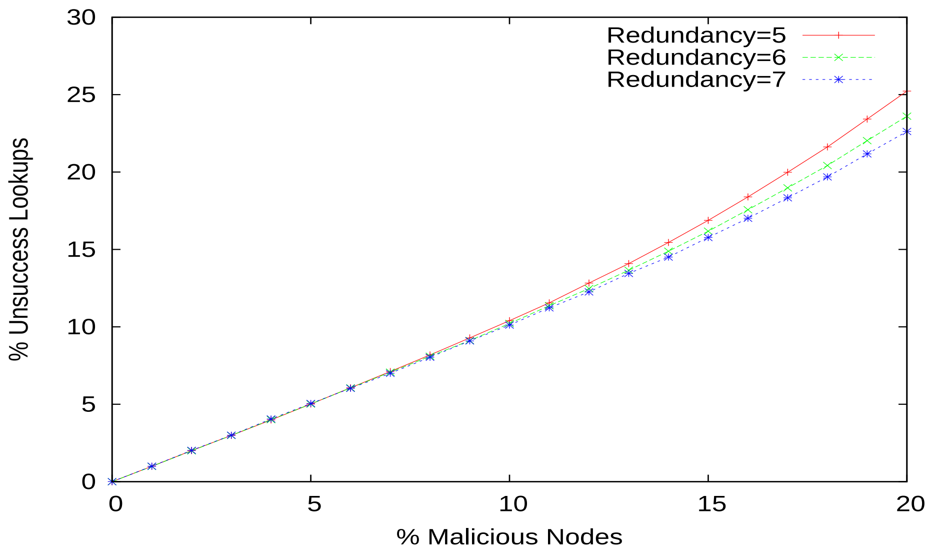

Figure 6.1. Percentage of failed lookups for different redundancy levels in static system.

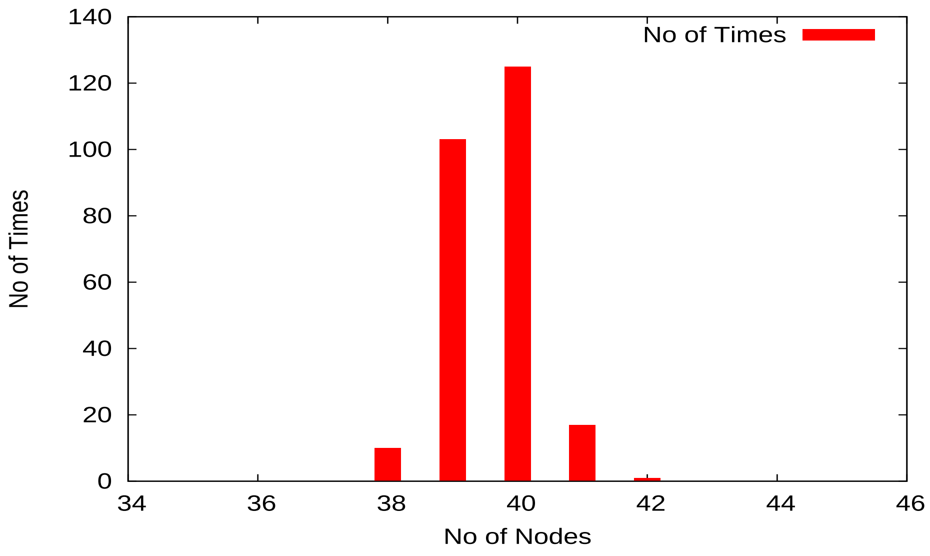

Figure 6.2. Distribution of group population in Salsa for 256 groups and 10,000 nodes.

This fact is very important for the joining procedure as well as for the group splitting and group merging procedures.

6.3 Success rate of lookup in dynamic systems

From this section on, for the rest of the chapter we present all of our simulation results in the dynamic Salsa environment implemented using a continuous time simulator. The details of the simulator were described earlier in Chapter 5. In Figure 6.3, we present the lookup success rate in dynamic Salsa with different redundancy levels and for different numbers of groups. Still success rate is good in the dynamic environment with respect to optimal. We also see the effectiveness of redundancy and bounds checking in presence of high churn node in the system. By increasing redundancy level from 4 to 9, we get a 5.45% higher success rate for 512 groups. Here another important finding is that the number of groups significantly impacts the success rate. For redundancy 6, difference between 128 groups and 512 groups' success rate is 1.73%. As the number of groups increase, the height of Salsa virtual tree increases and so do the number of paths for lookups. As a result, malicious nodes get more chances to be in the path and thus bias the selection. So the number of groups may act as a tuning factor in Salsa lookup success.

6.4 Message Statistics

So already we have shown the effectiveness and stability of Salsa for lookups even in presence of 20% of malicious nodes in the dynamic system. Now we study the efficiency of the system - what is the cost of the system in terms of messages to get the above success rates.

In Figure 6.4, we show overhead of messages per node per minute. Here we include all the messages for joins, leaves and merges. As other two events, updating global contacts and splitting groups, are done locally by nodes individually, there is no message overhead for these two. In Figure 6.4, we see that redundancy affects the

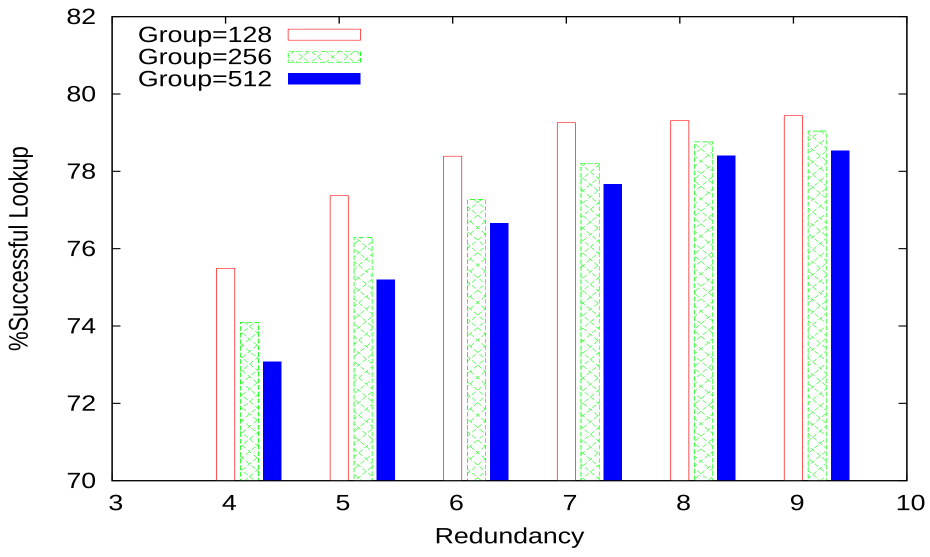

Figure 6.3. Lookup success rate in dynamic system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

message overhead. If we increase redundancy, message overhead grows. However the message overhead due to redundancy is minimal per node, as it increases 0.17 per node for redundancy level 4 to 9 for 512 groups. Same thing happens to the other results. For 512 groups, message per lookup (Figure 6.5) increases by 46.27, message per join (Figure 6.6) by 167.86, message per leave (Figure 6.7) by 0.04 and message per merge (Figure 6.8) by 0.09 which are comparatively low with respect to the success rate we are getting due to redundancy and bounds checking and if we consider the churn due to all the dynamics of the system. For same redundancy level we see the differences of message overheads for different number of groups (Figure 6.4). This is for the variation in number of messages for different events. In Figure 6.5, we see that the number of messages grows with number of groups. The reason behind this is clear: if the number of groups grows larger, the lookup paths become longer and so the number of messages increases. In Figure 6.6, message per join is greater for 128 groups than other twos. The number of lookups for nodes to get joining information and the responses from those nodes according to our proposed algorithm do not vary much with the number of groups that makes the number of messages in these cases

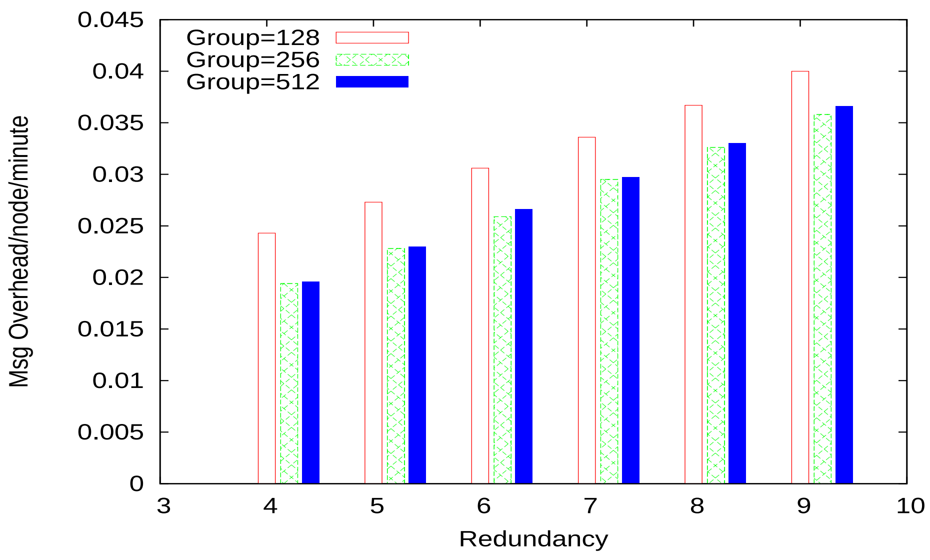

Figure 6.4. Message overhead in dynamic system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

almost similar. So the main varying factor in joining message count is the number of group population, as the joining nodes need to inform all other neighbors in the group where they join. As we perform these experiments for 10,000 nodes, for 128 groups the number of nodes is bigger than the other twos which reflect in the results in Figure 6.6. For the same reason, where leaving nodes inform their neighbors, at a same redundancy level it needs more messages for 128 groups than 256 and 512 groups respectively (Figure 6.7). In Figure 6.8, we observe that there is no message for merging for 128 groups as in these experiments we use merging threshold as 20 whereas the average group population in these case is around 78. Among other twos, according to our proposed algorithm for merging, 256 groups need more messages than 512. As in the merge procedure there is lot of communications among members inside a group and the other sibling group, the number of messages is more for 256 groups which have larger average group populations than 512 groups.

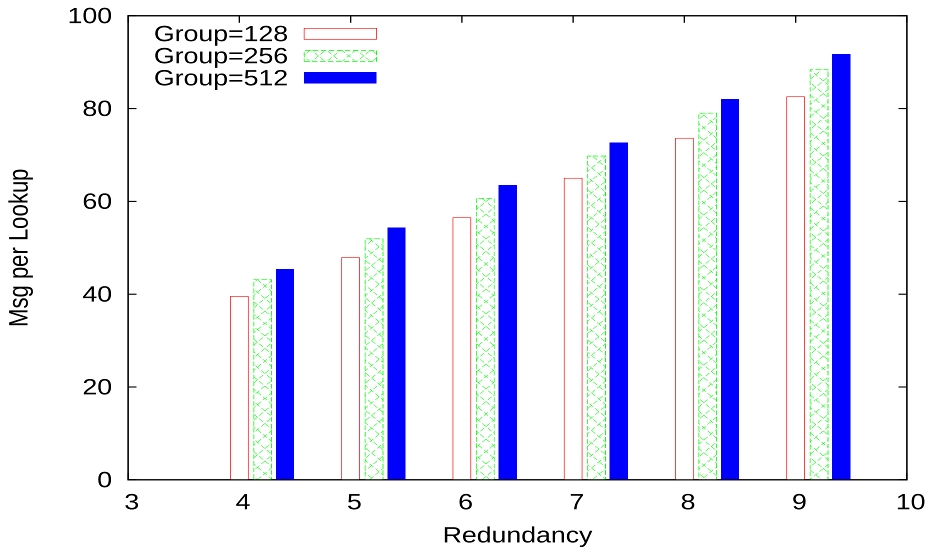

Figure 6.5. Message per lookup in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

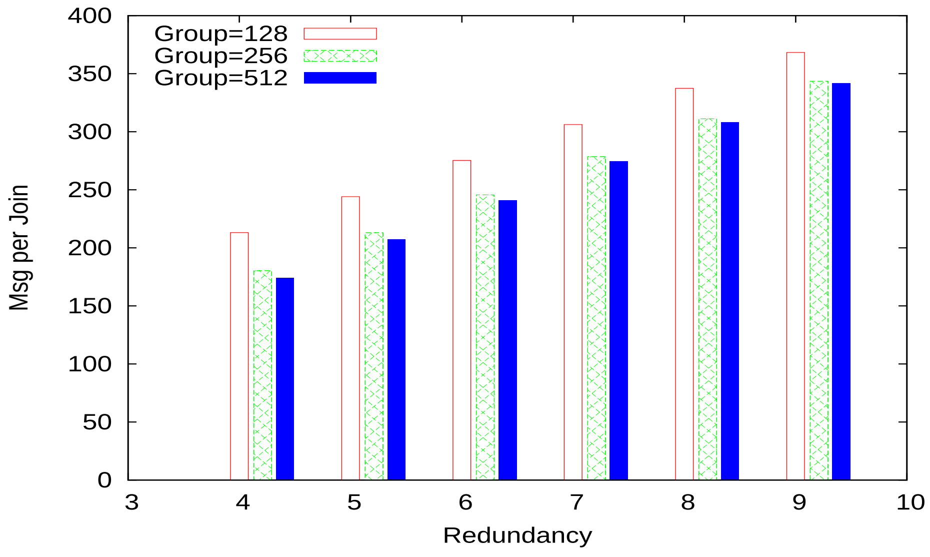

Figure 6.6. Message per join in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

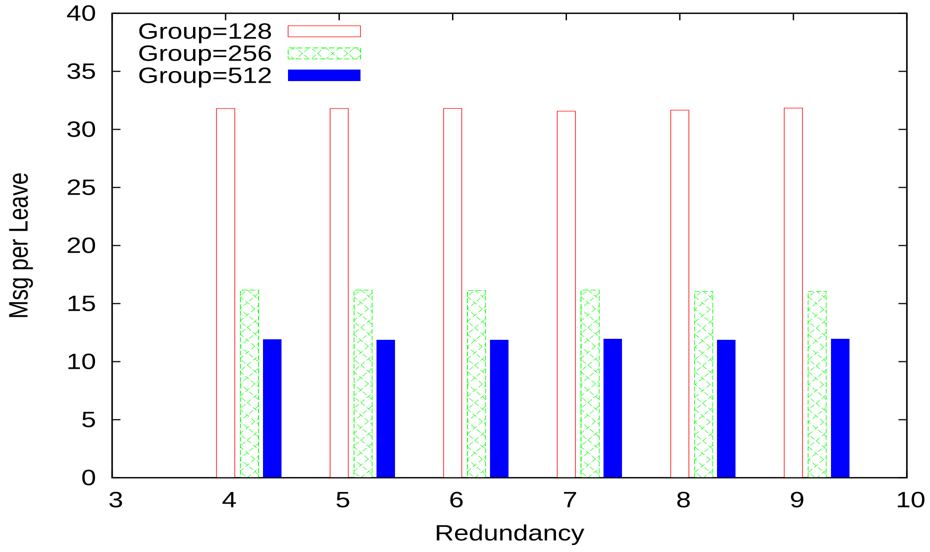

Figure 6.7. Message per leave in the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

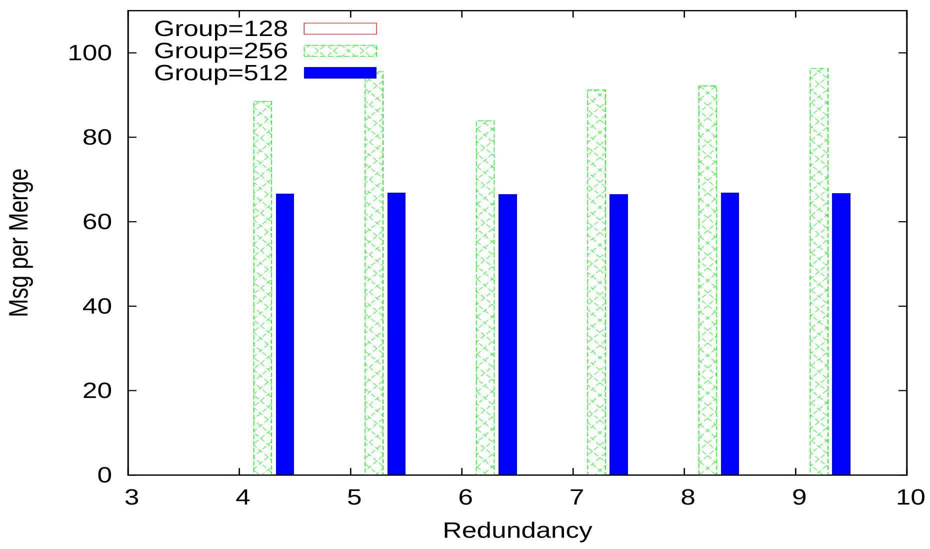

Figure 6.8. Message per mergein the system for 10,000 nodes, 20% malicious nodes and α=2 with different redundancy levels.

6.5 False and True Positives and Negatives

During a lookup, after getting back the redundant results and taking the closest one which is supposed to be the real owner of the target ID, we conduct bounds checking as we described in Section 3.2. Our bounds checks may give us four types of results. In some cases, the bounds check fails because the owner is too far from the target ID. So the result is rejected. But it may be that the node returned by the lookup is the actual owner. We define this case as a *False Positive.* On the other hand, the returned node may really be a malicious node in which case we define it as a *True Positive* (TP). A returned node passes the bounds check if it falls within the bound. If it is the actual owner, we define the case as a *True Negative* (TN), if not, we call it a *False Negative*(FN) result. In Table 6.1, we have presented some statistics for a bounds checking distance calculated with α=2 for different redundancy levels, number of nodes and number of groups. As an example, for 512 groups, we see in this table that with the increase of redundancy levels, FNs are decreasing which is helpful. However, FPs are increasing a bit, by 1.93%, from redundancy level 4 to 9 which is minimal and acceptable. Also we observe that, for all the cases TNs are much higher than the others and some TPs, which eliminate a good number of malicious nodes, are there too. Another interesting observation from this table is that TPs increases with the number of groups. For example, for redundancy level 4 and 10,000 nodes, we see that if the number of groups increases from 128 to 512, the percentage of TPs increases too. For the same redundancy level and number of nodes, if number of groups increases, the malicious nodes get more chance to bias the lookups, but the good thing is that our bounds checking is also detecting those to more extent. All these results show the strength of the bounds checking mechanism of Salsa.

Table 6.1. False and True Positives and Negatives for 20% malicious nodes with α=2

% Redundancy	No. of nodes	No. of groups	%FP	%TP	%FN	%TN
4	1000	32	17.26	10.0	16.14	56.60
		64	16.75	10.18	16.32	56.75
	10000	128	10.36	11.19	19.23	59.22
		256	10.10	14.06	19.65	56.19
		512	10.15	16.16	19.83	53.86
5	1000	32	17.42	7.84	15.93	58.81
		64	17.24	7.94	16.00	58.81
	10000	128	10.91	8.13	18.33	62.63
		256	10.89	10.45	18.65	60.01
		512	10.77	12.18	19.11	57.95
6	1000	32	18.02	6.55	15.94	59.49
		64	17.88	6.80	15.75	59.57
	10000	128	11.33	6.25	17.81	64.61
		256	10.99	8.29	18.35	62.37
		512	11.23	9.66	18.46	60.65
7	1000	32	18.32	5.70	15.33	60.66
		64	18.31	5.81	15.67	60.21
	10000	128	11.38	5.09	17.33	66.21
		256	11.53	6.50	17.86	64.11
		512	11.60	7.80	18.00	62.61
8	1000	32	18.04	5.47	15.68	60.81
		64	18.34	5.53	15.49	60.64
	10000	128	11.49	4.21	17.44	66.86
		256	11.69	5.56	17.58	65.17
		512	11.62	6.72	17.63	64.03
9	1000	32	18.26	5.18	15.33	61.23
		64	18.07	5.23	15.47	61.23
	10000	128	11.61	3.82	17.39	67.18
		256	11.89	4.85	17.45	65.81
		512	12.08	5.77	17.64	64.52

6.6 Attempts per successful lookup

In this section, we present some results that help decide the α value for computing bounds checking distance in a Salsa system. In Table 6.2, we choose α=1 and see that the attempts per successful lookup ranges from 2.049 to 2.601. When we choose α=2, i.e., we increase the range for bounds, we see in Table 6.3 that attempts per successful lookup goes down and ranges from 1.489 to 1.857 which is a clear improvement over α=1. However, the α value also affects the number of false and true positives and negatives. Therefore based on trade-offs, we have to decide what value we should use for α. Moreover, both of these tables clearly show that if we increase redundancy level, the attempts per success lookup go down. So redundancy level is another decision factor for choosing α.

Table 6.2. Attempts per successful lookup for 20% malicious nodes and α=1 for different redundancy levels

Redundancy	No. of nodes	No. of groups	Attempts per successful lookup
4	1000	32	2.589
		64	2.601
	10000	128	2.326
		256	2.475
		512	2.577
5	1000	32	2.529
		64	2.523
	10000	128	2.196
		256	2.306
		512	2.397
6	1000	32	2.466
		64	2.442
	10000	128	2.126
		256	2.214
		512	2.297
7	1000	32	2.431
		64	2.439
	10000	128	2.100
		256	2.163
		512	2.217
8	1000	32	2.438
		64	2.423
	10000	128	2.065
		256	2.130
		512	2.179
9	1000	32	2.416
		64	2.433
	10000	128	2.049
		256	2.104
		512	2.137

Table 6.3. Attempts per successful lookup for 20% malicious nodes and α=2 for different redundancy levels

Redundancy	No. of nodes	No. of groups	Attempts per successful lookup
4	1000	32	1.767
		64	1.762
	10000	128	1.689
		256	1.780
		512	1.857
5	1000	32	1.700
		64	1.700
	10000	128	1.597
		256	1.666
		512	1.726
6	1000	32	1.681
		64	1.679
	10000	128	1.548
		256	1.603
		512	1.649
7	1000	32	1.649
		64	1.661
	10000	128	1.510
		256	1.560
		512	1.597
8	1000	32	1.645
		64	1.649
	10000	128	1.496
		256	1.535
		512	1.562
9	1000	32	1.633
		64	1.633
	10000	128	1.489
		256	1.520
		512	1.550

CHAPTER 7

CONCLUSION

Highly distributed peer-to-peer (P2P) anonymous systems should have better distribution of trust and more scalability than centralized approaches. Existing systems like Tor and Tarzan have vulnerabilities, as they require nodes to have global knowledge of the system. Tor suffers from centralized approach too. Crowds gives out the list of nodes and suffers from some centralized approach for joining procedure. MorphMix avoids giving nodes global knowledge, but recent research reveals it is vulnerable to collusion attacks.

To deal with these problems, Salsa has been proposed in [14]. Since it is based on DHTs, it is very scalable. Due to the use of consistent hashing and Chord-like ID-space, in this system attackers cannot control mapping of nodes to IDs on the ID-space which makes the Sybil attack more difficult. Since Salsa uses a virtual tree structure, every node in the system has limited knowledge about the whole system; this helps to protect against the intersection attack in P2P systems. It is also secure against some other attacks, as described in [14].

In this thesis, we introduced more randomness and redundancy and used more flexible bounds checks in lookup procedure and showed that Salsa 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 proposed algorithms to handle all the dynamics of the Salsa system. We discussed the design challenges and decisions of these algorithms and tested them in simulation. We showed that the proposed algorithms can handle churn in a dynamic Salsa system in which peers continue and Salsa gets good lookup success with modest message overhead.

REFERENCES

[1] D. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*, Communications of the ACM, vol. 4, no. 2, February 1981.

[2] 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.

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

[4] M. J. Freedman and R.Morris, *Tarzan: A peer-to-peer anonymizing network layer*, in Proceedings of the 9th ACM Conference on Computer and Communications Se- curity (CCS 2002), Washington, DC, November 2002.

[5] M. Reiter and A. Rubin, *Crowds: Anonymity for web transactions*, ACM Trans- actions on Information and System Security, vol. 1, no. 1, June 1998.

[6] M. Rennhard and B. Plattner, *Introducing MorphMix: Peer-to-Peer based Anony- mous Internet Usage with Collusion Detection*, in Proceedings of the Work- shop on Privacy in the Electronic Society (WPES 2002), Washington, DC, USA, November 2002.

[7] A. Back, I. Goldberg, and A. Shostack, *Freedom 2.0 security issues and analysis*, Zero-Knowledge Systems, Inc. white paper, Nov 2000.

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

[9] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek and R. Anderson, *Sybil-resistant DHT routing*, in Proceedings of ESORICS, Sep. 2005.

[10] An article on privacy on http://whatis.techtarget.com (http://search datamanagement.techtarget.com/sDefinition/0,,sid91_gci212829,00.html)

[11] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, *Hiding Routing Information*, in Proceedings of Information Hiding: First International Workshop, R. Anderson, Ed. Springer-Verlag, LNCS 1174, May 1996, pp. 137-150.

[12] P. Tabriz and N. Borisov, *Breaking the collusion detection mechanism of MorphMix*,in Proceedings of Privacy Enhancing Technologies Workshop (PET), June 2006.

[13] M. Wright, M. Adler, B. Levine, and C. Shields, *The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems*, in Proceedings of ACM Transactions on Information and System Security (TISSEC) 4(7), November 2004, pages 489-522.

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

[15] Alessandro Acquisti, Roger Dingledine and Paul Syverson, *On the Economics of Anonymity* in Proceedings of Financial Cryptography (FC '03), January 2003.

[16] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky and Feng Xiao, *SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks* sp, pp. 3-17, 2008 IEEE Symposium on Security and Privacy (sp 2008), 2008.

BIOGRAPHICAL STATEMENT

Safwan Mahmud Khan was born in Comilla, Bangladesh, in 1981. He received his B.Sc. degree in Computer Science and Engineering from University of Dhaka, Bangladesh, in 2005 and his M.S. degree with a Thesis from the University of Texas at Arlington in 2008 in Computer Science. His interests match mostly with Information Security and programming. He has been part of the Information Security (iSec) Lab at UTA. His research interest includes Information Security, Anonymity, Data Mining and Designing Algorithm.