CIRCUIT POOLS: PREVENTIVE INFORMATION LEAK ATTACKS IN SALSA THROUGH CACHED LOOKUPS

by

ANAHITA DAVOUDI

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 ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2012

ACKNOWLEDGEMENTS

First of all, I would like to thank my advising professor, Dr. Matthew Wright, for giving me the opportunity to work with him in the iSec lab and pursue my Master's thesis under his supervision. This has been a great experience. I would also like to thank Dr. Wendell A. Davis and Dr. Stephen R. Gibbs for being part of my thesis committee and giving helpful insights for my thesis. A big thank you goes to my lab mates, who have been very helpful during my time in the iSec lab. Most importantly, I would like to thank my family for being extra supportive and striving to give me the best in life.

August 10, 2012

ABSTRACT

CIRCUIT POOLS: PREVENTING INFORMATION LEAK ATTACKS IN SALSA THROUGH CACHED LOOKUPS

ANAHITA DAVOUDI, M.S.

The University of Texas at Arlington, 2012

Supervising Professor: MATTHEW WRIGHT

An improvement in preventing information leak attacks has been proposed by enhancing the path building process using pair of nodes obtained from lookups done by each initiator randomly. Using a pool of pairs locally produced for each initiator by random lookups done by that node; improves anonymity for the circuit initiator. The percentages of how many pools each pair belongs to, availability of each pair, and number of circuit attempts are analyzed for different number of both pool size and redundancy level. With these modifications and additions, the performance of the system has improved against the Conventional Continuous Stage Attack, and the probability of success for the Bridging an Honest First Stage Attack has been decreased. For the improved Salsa system with certain values for redundancy level and percentage of malicious nodes, this probability is significantly reduced comparing to the original Salsa system. The third type of attack studied here, the Bridging an Honest Stage Attack, was shown not to be affected by the implemented modifications in the design of the system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

One of the low-latency anonymous networks, Tor [2, 5], distributes the transactions over the user nodes in the network, to increase the anonymity of the initiator of a message from attacks. Tor uses a distributed trust approach, using mutually distrustful users, and providing a more secure network. Some of the advantages provided by Tor include little synchronization or coordination between nodes, and a reasonable trade-off between anonymity, usability, and efficiency.

Salsa [7] is one of the possible designs for peer-to-peer anonymity systems and uses distributed hash table lookups to select random relays for building its paths and circuits. Circuits in Salsa, similar to the case for Tor, are built through randomly chosen nodes (or relays). The redundant lookups that Salsa uses make it vulnerable to information leak attacks, compromising the path structure anonymity, since attackers can detect most of the lookups.

In this thesis, new concepts are introduced to improve the original design of Salsa and increase the anonymity of the initiator of a lookup in the Salsa system.

The probability of the detecting the initiator is examined by using the pairs of nodes randomly chosen from cached lookups obtained by initiator to employ in building the circuit. The probability distribution of number of pools that each pair belongs to is used to find the entropy of detecting the initiator for different size of pools. The bigger the size of pool, the higher is the anonymity of the system. The three information leak attacks presented in the Prateek Mittal et al. [1] are discussed for the new Salsa system design.

The thesis is organized as follows. In chapter 2, the Salsa system structure is discussed in the background. The improvements that have been made to the Salsa system are presented in chapter 3. Then in chapter 4, Salsa discrete event simulator and simulation tests are

described. The simulation results of implementing the changes in Salsa system are presented and analyzed in chapter 5. The conclusion of thesis is presented in chapter 7.

CHAPTER 2

BACKGROUND

In this chapter, the Salsa system, a design for peer-to-peer anonymity systems that uses distributed hash table lookups to select random relays for building its paths is described. Salsa is designed to select nodes to be used in circuits randomly from the full set of nodes, even though each node has knowledge of only a small subset of the network. Salsa lookup process uses randomness, redundancy and bound checking to prevent malicious nodes from being selected or returning false information about other nodes.

2.1. Salsa Architecture

Salsa uses ID space based on the hash of the node's IP address. The nodes are placed in sorted order around a circular ID space and each node, N_i, owns the fraction of the ID space between itself and its preceding node N_{i-1}. If a node requests a specific ID, say X, which is called the target ID between N_{i-1} and N_i, it will be routed to N_i, since it owns that ID space [7]. The system divides the ID space into groups of contiguous IDs and the groups are organized as a virtual binary tree. A node belongs to a group if its ID is in that group's ID space. In Figure 2.1, the Salsa system binary tree is presented that uses a 7-bit ID space (0-127) and a 3-bit group ID space (0-7).

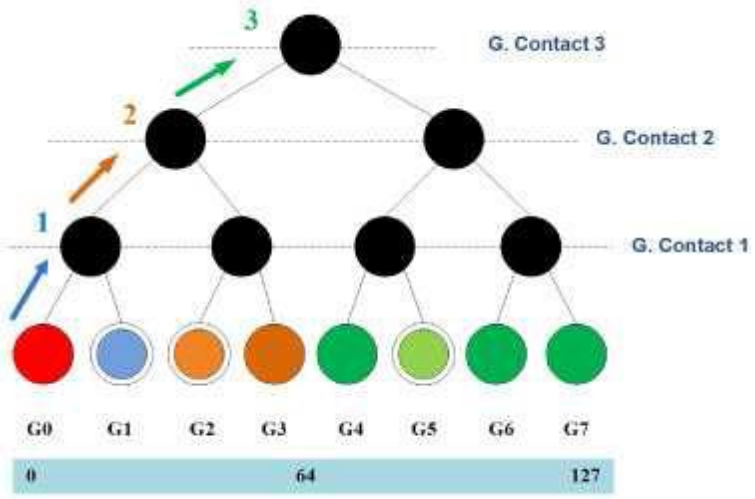

Figure 2.1 Salsa binary tree

Routing information is stored in *local contact table* and a *global contact table*. IDs and owned ID space all nodes in the same group are stored in a node's local contact table. In a node global contact table, information of a contact for each virtual tree level of Salsa is stored. Here is the description of how the tree in figure 2.1 works. If a node is in group G0 it goes one level up in the tree and selects a global contact from the other sub-tree, G1. For selecting a global contact, it uses the ID-space and selects a random ID from the space. 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, G2-G3, randomly. Here a node from G2 is selected. It again goes one level up and selects its last global contact from the other sub-tree randomly, from G4-G7, here G5. As the height of the tree is three, the node will have three global contacts in its table [8].

2.2. Lookup Procedure, L

Salsa uses redundancy and bound checking to protect against malicious nodes when looking up a particular ID. It is assumed that node I is the query node starting the lookup procedure by looking for a target ID in ID-space. Node I uses R as the redundancy level and B as the bounds. The lookup procedure consists of two main parts, a recursive lookup similar to binary search and a redundant lookup [3, 6, 7].

Salsa uses redundancy and bounds check to protect against malicious nodes when looking up a node. Assume that I is the query node that it is looking for a target ID in ID-space and starts the lookup procedure. Redundancy level and bounds here are *R* and *B* respectively [7]. Here is algorithm for recursive lookup [8]:

Recursive Lookup:

1. A random node I starts the lookup process.
2. Node I calculates the part of the virtual Salsa tree where the target ID is in.
3. Node I requests its global contact in the target ID space to find the owner of the target ID.
4. The global contact computes whether the target ID is in its own group or not.
5. If the global contact was in its group, it returns the information of the target ID to node I.
6. If the global contact is not in its group, the global contact node would repeat steps 2 – 6.

Redundant Lookup:

1. Node I selects R neighbor of its group randomly from its local contact table.
2. Node I requests these R nodes to find the owner of T.
3. Each of the R nodes finds the owner of T using recursive lookup.
4. Node I calculates the distance of these results from T finding the node with the minimum distance name N which have and ID of ID_N .

5. Node I checks if $(ID_N - T) < B$

A malicious node needs to make all of the results from a lookup the same and also as close to the target ID as possible, so that they pass the bounds check. In this way, it can successfully manipulate the results. Another possible attack scenario happens if the target owner itself is malicious. In this case, the correct results will be obtained.

Salsa node churn includes nodes continuously joining and leaving the system over time. Algorithms for managing the churn without giving advantages to attackers have been described by Khan et al [8].

2.3. Circuit Building Mechanism

In the Salsa structure, nodes are used as proxy routers to build a path between the initiator and the recipient to provide anonymous communication. These nodes are chosen from the global pool of nodes in a random way, but each of these nodes only have local knowledge of a small portion of the network, and they are only aware of their own previous and next node in the path [3, 7].

2.3.1 Salsa Path Building

Here how an initiator builds the path between itself and the recipient in the original version of Salsa is described. The initiator chooses R random IDs and redundantly looks up the nodes that correspond to these IDs; these nodes make up the first stage and establish keys with them. Each of the nodes in the first stage looks up r additional nodes that will constitute the second stage. The nodes in the first stage act as relays in the circuit built through the nodes in the second stage. Through these circuits, the nodes in the second stage are instructed by the initiator to look up a final node, and hence the final node will be added to the system. In Salsa path-building, it is only the initiator that looks up the nodes in the first stage redundantly. The figure 2.2 [1] shows an example of how paths are built in Salsa for redundancy = 3 and path length=3.

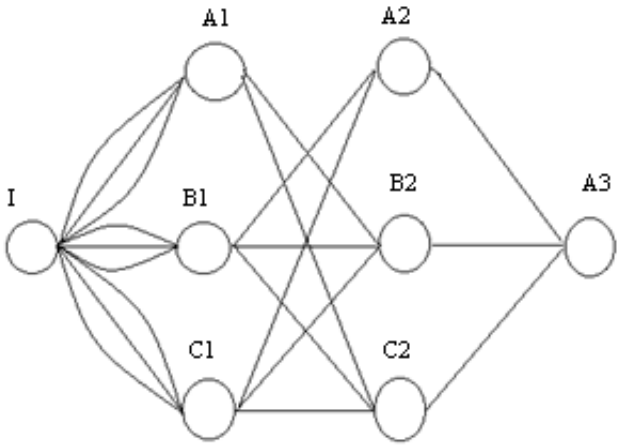

Figure 2.2 an example of path building in Salsa for r = 3, l = 3.

2.3.2 Active Path Compromise Attacks on Salsa

The probability that the nodes involved in the path building mechanism of Salsa are compromised can be biased by active attacks on the lookups. Another possible attack that can affect the Salsa path building is a public key modification attack. In such an attack, all the nodes in one stage must be compromised, so that they can modify the public keys of the nodes being looked up next. Thus, since the initiator can no longer detect an incorrect public key, the bound checking algorithm of Salsa is circumvented and it cannot verify if the IP address is in the right range. The messages that are passed to the node and encrypted with the modified public key will then be forwarded to corrupt nodes. In this way, the attackers can emulate the next stages and then reach a last node. This attack and other attacks that target the lookup procedure in Salsa are active attacks. If the first and last nodes are compromised, the path will be compromised using end-to-end timing analysis. To reduce the chance of that the attacks succeed in compromising the paths, the redundancy in the mechanism employed to build the path must be increased. This is because with larger redundancy, there will be less chance for the active attacks on lookups and public key modification attacks.

The next three attacks described below are passive information leak attacks. It is shown in these attacks that an increase in redundancy results in a higher chance of success for the

information leak attacks. Thus, there is a tradeoff between robustness against active attacks and robustness against passive information leak attacks [1].

2.3.3 Conventional Continuous Stage Attack

In Salsa, if there is at least one attacker node in each stage of a path, that path is compromised. Assuming, as shown in the figure 2.3 [1], that nodes A_1, A_2, and A_3 are attacker nodes. To build a path, node A_1 looks up node A_2, and A_2 looks up node A_3. In this way, the anonymity of the system is compromised because the attacker has passively bridged the first and last stages. Recall that nodes in each stage lookup r times for nodes in the next stage in order to build a path. Thus, increasing the redundancy increases the effectiveness of this attack, because with more lookups, there is more chance to have attacker nodes present at each stage.

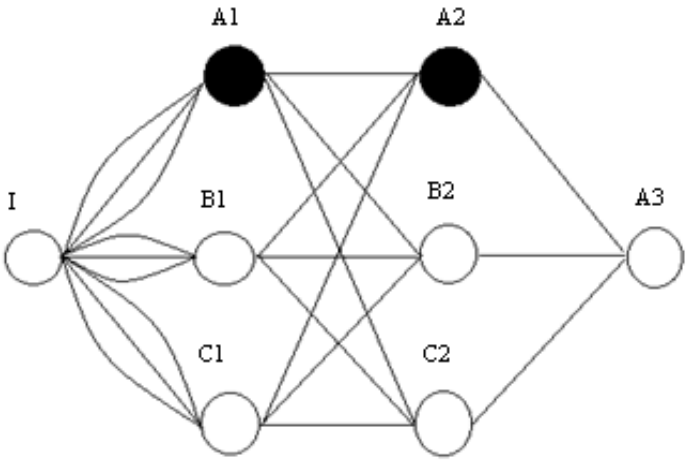

Figure 2.3 Information leak attack on Salsa: conventional continuous stage attack

2.3.4 Bridging an Honest First Stage

In this attack, the attacker looks for nodes that have looked up nodes that the attacker has deduced to be the node in the first stage of the path. As shown in the figure 2.4 [1] since the honest nodes in the first stage all have key establishments with the malicious nodes in the

second stage, the attacker knows their identity [1]. Thus, to find the initiator, the attacker can look for the node that has looked any of them up; the chance of the attacker detecting the initiator this way is $1- (1 - f)^9$. This attack can occur when at least one of the nodes in the second stage is compromised. In the case that there is one compromised node in the second stage, the chance of success for this attack is $1 - (1 - f)^3$ for the figure 2.5 [1].

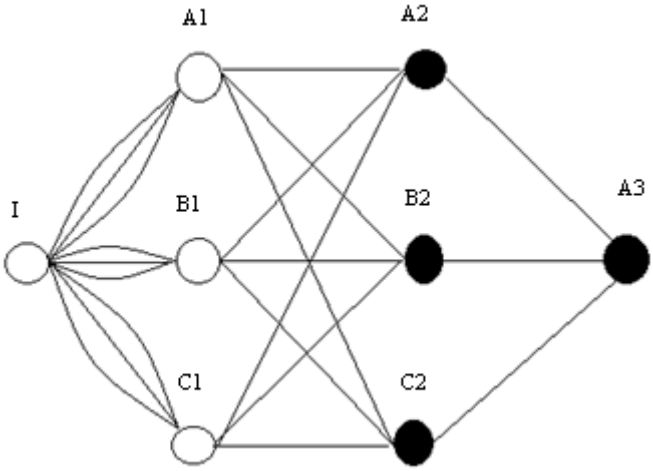

Figure 2.4 Information leak attack on Salsa: bridging an honest first stage

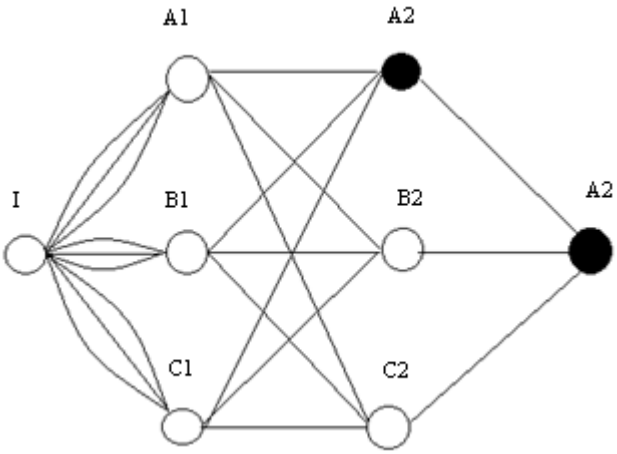

Figure 2.5 Information leak attack on Salsa: bridging an honest first stage

There are cases that the attackers recognize a wrong node as the initiator, and a false positive occurs. These cases happen when the node in the first stage of a compromised path is part of more than one path.

2.3.5 Bridging an Honest Stage

Another attack that the original version of Salsa is vulnerable to is bridging an honest attack. In this attack, there is a stage of honest nodes between two stages of malicious nodes, yet they will be able to recognize that they are on the same path; thus in this attack, there must be at least one malicious node in the first stage and one malicious node in the third stage. Figure 2.6 [1] shows this deployment of malicious and honest nodes.

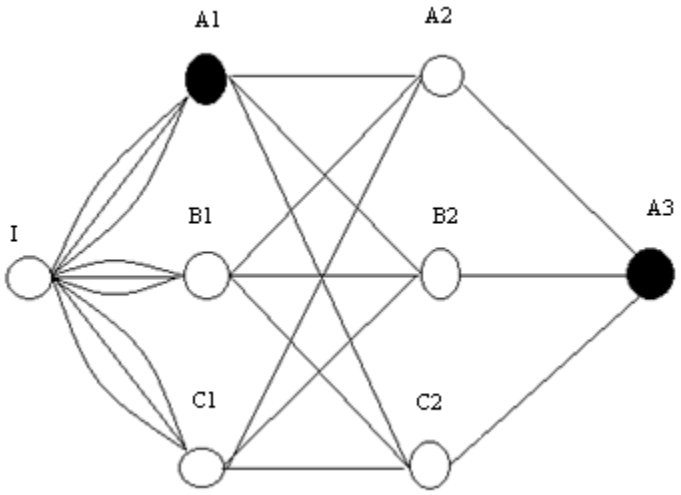

Figure 2.6 Information leak attack on Salsa: bridging an honest stage

As shown in figure 2.6, all the nodes in the second stage are honest, but the malicious node A_1 knows their identity, as it has looked them up as part of building the path; also, at least one of the honest nodes in the second stage will look up the malicious node A_3 in the third stage. Now, the two nodes A_1 and A_3 know that they are part of the same path, because there is one common honest node between them.

CHAPTER 3

IMPROVEMNET TO SALSA

In this chapter, the modifications and improvements to the Salsa system are presented and discussed. New concepts of using pairs instead of nodes and Pool structure and filling the pool for each initiator using lookups are introduced.

3.1 Pair Building Mechanism

Randomly choosing a pair of nodes for second and last node in the circuit, instead of separately selecting them, gives higher anonymity to the initiator when it is building a circuit. The pair building process consists of 5 steps:

1. A random initiator picks up a random ID from whole set of nodes that already exist in the system.
2. Then initiator would look up the random ID to find its owner ID using the owner function.
3. The owner ID would become an input for the hash function. This function will give another random ID already existing in the system as the output. The hash function use hash of ID of owner to give the random ID.
4. The initiator looks up the second random ID and finds the owner ID of it by using the owner function.
5. Now the ID of the first owner and second owner makes a pair.

This pair is added to the pool of the initiator that started the lookup and is local to that particular initiator. The pairing process happens in the lookup event each time that the random node initiates a lookup process.

3.2 Hash Function

The need for using a hash function to relate the first node to the second node of the pair is based on the fact that by looking up a particular node, it is always return a unique node as the second node in the pair set.

The owner ID of the first node that has been looked up by the initiator becomes an input for a hash function. The output of this function is another random ID to be used by the lookup process to find the second node. Below is the explanation for the hash function used in the pairing mechanism:

1. The pairhash function gets the owner of the first node as the input.
2. Owner of the first node divided by the number of nodes would give an integer number.
3. For using the hashcode function in java, this integer number is changed to a string.
4. The hashcode of this string is produced.
5. The hashcode is divided by pt (number of points in the ID space) and remaining is returned as the result, which is another random ID.

3.3 Pool Structure

The pool structure is local to each initiator. Every initiator makes a pool of pairs for itself by looking up random nodes and their corresponding partner nodes in the network. The pool sizes are similar for all initiators. Here an example is given for an Initiator I filling its pool:

Node I performs a look-up of a random ID from a whole set of nodes that already exist in the system. Then initiator would look up the random ID to find its owner ID using the owner function and it returns node A. The owner ID would become an input for the hash function. This function will give random ID already existing in the system as the output. The hash function use hash of owner ID to give the random ID. Initiator would look up the second random ID and find the owner ID of it using the owner function to find owner ID B. Now ID of the first owner and

second owner would make a pair. The pair (A, B) will be added to the pool of pairs for the Initiator. The same procedure happens for pair (C, D) and (E, F).

3.4 Improved Circuit Building Mechanism

After an initiator filled up its pool with pairs, then it is able to make a circuit using its pool. The Circuit Building algorithm consists of the following steps:

1. Initiator node I does a redundant lookup for a random node X.
2. Node I initiates a circuit through X.
3. Node I selects a pair (A, B) from its pool.
4. Node I extends its circuit from X through A and then through B.

3.5 Lookup minimum distance

In the original Salsa system when a node does a lookup, the distance of (target – query node) is checked to be smaller than a bound value. The procedure to determine this value has changed for the improved Salsa system. In the new system, when a query node does a lookup for a target node, the distance between them is checked to be smaller than $(f^* (2^b/n))$, which f is a system parameter, b is address space in bits and n is number of nodes in the system. The parameter f is set such that 10% of lookups are less than $(f^* (2^b/n))$ of target.

This gives the opportunity that nodes with larger ID space are chosen in the lookup process. By enforcing the new bound checking, node with ID space will be discarded and new lookup will be done. By changing the lookup minimum distance there will be more chance that they will be for the same ID space which results that each pair would go to more pool for different initiator.

The first step is to find the entire owner IDs on the ID space and calculate the distances between them. Since here address space is 30 bits, using 1000 nodes in the system, there will

be nearly 2^{20} IDs/node. If the ID space for node A is smaller than ($f*2^{20}$) (target – query node < $f*2^{20}$)) then node A would not be used and another lookup would be performed.

CHAPTER 4

SIMULATION DESIGN

To verify effectiveness of the proposed system against Information Leak, various numbers of experiments was performed with changes in redundancy level and size of pool parameters to check the results achieved in the Salsa system analysis.

4.1 Salsa System Event Simulator

The Salsa system includes 1000 nodes and uses 30 bits for the ID space, and a 20-bit group address space. In the simulator 32 groups for the 1000 node system with a tree height of five (2^5= 32) has been used. For each test, 10 separate systems are simulated. Value two for α is used as the bound check parameter. In each simulation test a fix percent of malicious nodes from 0 to 20% is used. Beyond 20% of malicious nodes, the attacker has substantial advantages regardless of the way the system is organized. The Salsa system is implemented in Java, but Matlab was also used to simulate the mathematical analysis. The system is simulated under varying degrees of attack with the percentage of malicious nodes ranging from 0 to 20. The tests are capped at 20% malicious nodes to simulate a realistic scenario. Beyond 20% malicious nodes, the attacker has substantial advantages regardless of the way that our system is maintainer.

Salsa event simulator [4] implemented such that at first all nodes join, and then all the pools are filled with pairs by doing redundant lookups. After filling the pools with pairs from the node's lookup results, nodes would start leaving, joining and doing more lookup events over the time distribution of the salsa dynamic system.

In the Salsa system simulator, these events are simulated:

1. Joining of a node, J
2. Leaving of a node, E

3. Lookup Process, L
4. Circuit Building, C
5. Updating the global contacts, UG
6. Splitting a group, SG
7. Merging two groups, MG

When one event occurs, no other event can take place at the same time, and they occur in the order they are generated and run sequentially. A join event may invoke another event, such as lookup. In this case, the main event and the sub-event happen sequentially after each other. The goal here is to achieve all the statistics related to a cumulative distribution function for the pair building mechanism, pool structure, circuit building mechanism, messages that will be generated for lookup, leave event; and their total overhead messages.

4.2 Salsa Queue

The Queue, Q, has been used for the simulating the Salsa system. At first 1000 joins would occur so all the nodes will exists in the system. Then enough lookups will be done at the beginning to fill entire pools with pairs. One simulation minute after each lookup, a lookup event L is created at a constant rate and is put into the queue.

After the 1000 joins at the beginning and lookups to fill the pools, the join event J occurs randomly at a rate of λJ with an exponential distribution. After each join event is executed, the next one is generated and put into the queue. The same procedure happens for a leave event E at random times with rate λE which follows an exponential distribution. Whenever there is at least one event in Q, it is dequeued and an L, J or E event is obtained until it becomes empty. Circuit building event, C, happens with a Poisson distribution with a mean of two minutes after all the joins happened and pools are filled with pairs obtained by lookups that initiator has done before.

A node updates its global contact with a UG event if it finds its global contact is no longer active. If the group is too small or too big for a node to join that group, then join procedure invokes the SG or MG procedures respectively. They are executed immediately after the join procedure is finished. The same procedure happens for lookup L.

Circuits are built following a Poisson distribution with mean of 2 minutes. This distribution is a discrete probability distribution to show the probability for a certain number of events that occurring in a fixed range of time, knowing that these events occur with a known average (lambda) rate. In our Simulation Design we assume $\lambda = 2$ minutes in circuit building process.

CHAPTER 5

RESULTS

In this chapter, the simulation results are presented and discussed.

5.1 pair Availability

We assume that when a pair is created by the initiator and is added to its pool, the pair life has started. Later on, if any of the pair leaves the system, that pair would not be available anymore and its life cycle is ended. So the time between a pair creation until one of its node leave the system, is called the pair life duration. When a node leaves the system the leave process checks if that node was used in a pair. If it was, then when the node leaves the system, it will notify the initiator that its pair is not available anymore. If it was not, the node just notifies its group member that it has left the system.

In figure 5.1 the cumulative distribution function of pairs' life cycle is presented in seconds. The simulation was done for 1000 nodes, redundancy level of 3 and pool size of 50, 100, 150, 200 and 250 pairs in each pool. Increasing the size of each pool would make the life time of the pairs shorter since in bigger pools a leaving node would be able to cause more churn and more pairs would be unavailable.

Increasing the redundancy level would not change the pair life cycle since the redundancy of the lookup process would not have any effect on leaving time of a node that happens according to an exponential distribution in the leave process of the queue.

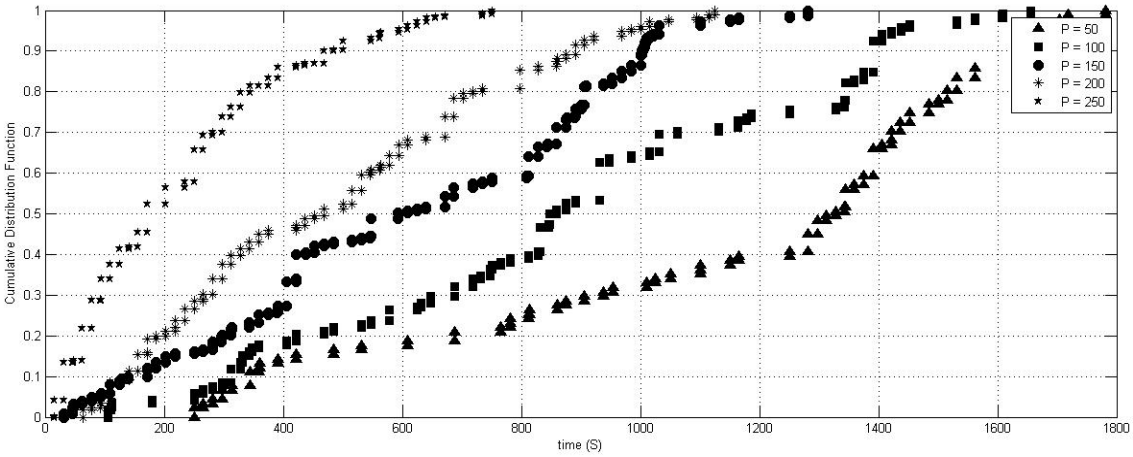

Figure 5.1 CDF of pair availability (in seconds), p = 50, 100, 150, 200, 250 and r = 3.

5.2. Pair Distribution in Pools

Figure 5.2 is a plot showing the percentage of pairs that belong to a certain number of pools, for the case when r=3. As can be seen from the figure 5.2 in the smaller size pool, the pairs belong to more pools. For larger pool sizes, the pools are not filled as fast and the pairs do not to belong to a large number of pools. With an equal number of pairs in the system, when the size of pools is smaller, such as P=50, the pairs belong to all the pools, compared to the case with P=250, where it would take more pools in the system to have all the pairs appear in all the pools.

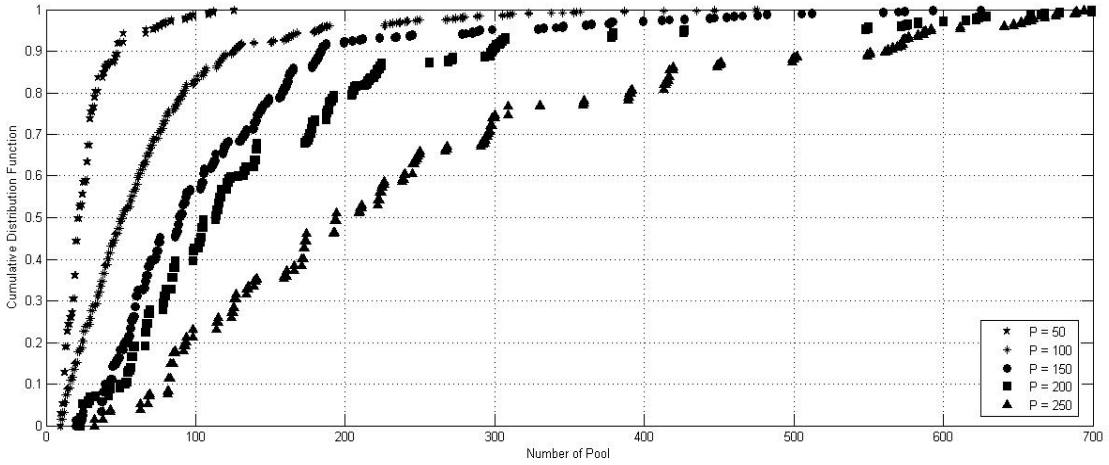

Figure 5.2 CDF of number of pools that each pair appears in, p = 50, 100, 150, 200, 250 and r = 3.

The figure 5.3 shows the percentage of pairs that belong to a certain number of pools for the case of r=7. Compared with the previous case with r=3, it takes more pools in the system to have the pairs belong to all of the pools for each of the pool sizes. This is because with more redundancy more pairs are built.

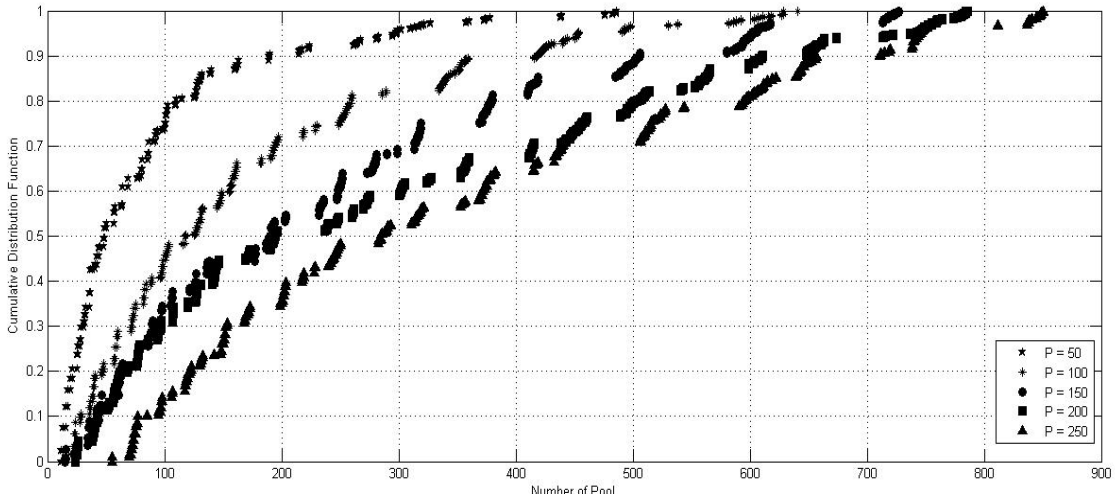

Figure 5.3 CDF of number of pools that each pair appears in, p = 50, 100, 150, 200, 250, and r = 7.

The figure 5.4 shows the percentage of pairs that belong to a certain number of pools for the case where r=10. In this case the curves are stretched along the axis, representing the number of pools, compared with the cases of r=3 and r=7, because with more redundancy, hence more lookups, more pairs are built in the system, and it takes more pools to have all the pairs be in all the pools.

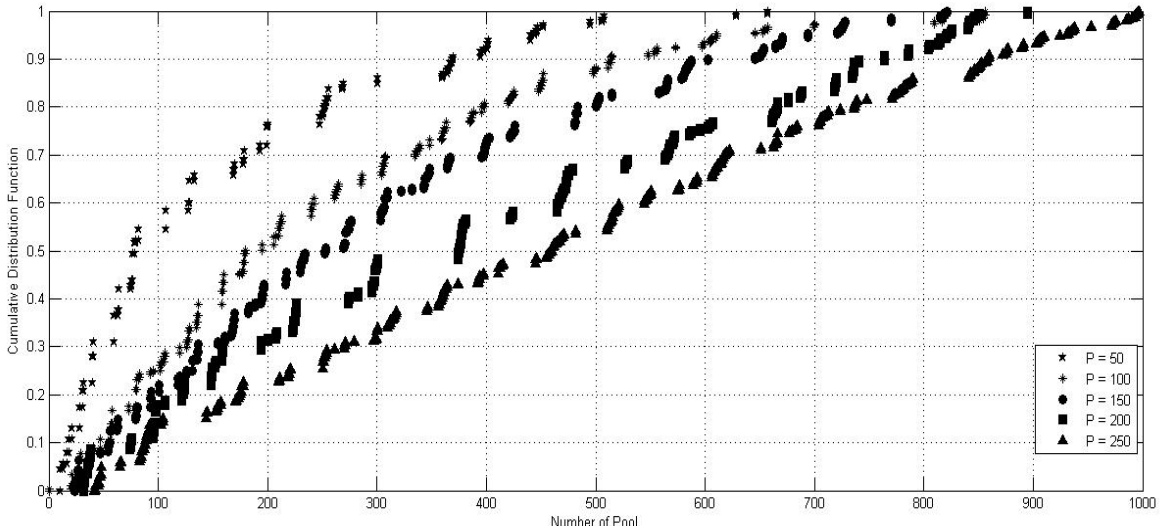

Figure 5.4 CDF of number of pools that each pair appears in, p = 50, 100, 150, 200, 250, and r = 10.

5.3. Circuit Failing Time

Each time that an initiator wants to make a circuit, which happens with a Poisson distribution and a mean time of two minutes, after looking up the first node, it would pick up a pair from its pool that has already been filled with the pairs generated by lookups. In the continuous time simulator and the fact that each event should be finished before the other one is started. When the initiator picks up a pair, there may be some pairs in the initiator's pool that are not available anymore when they are going to be used in the circuit building process. The reason for this is that one of the nodes of the pair left the system.

Each time that pairs are picked up from a pool but not available in the system (at least

one of the nodes left the system), the initiator tries to pick another pair. The figure 5.5 shows the cumulative distribution function of a number of tries that each time the initiator been through to that failed to build the circuit. The number of attempts to get to the available pair picked from the pool in the system for each circuit to be made by the initiator has been counted.

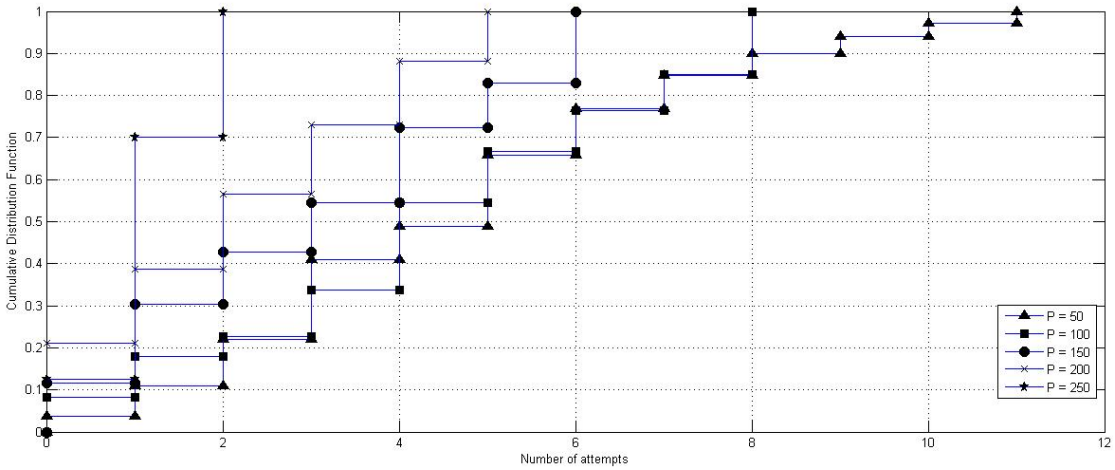

Figure 5.5 CDF of number attempts for building a circuit, p = 50, 100, 150, 200, 250, and r = 3.

5.4. Message Statistics

The efficiency of the system in terms of size of pool and circuit building process has already been discussed. Now we study the efficiency of the system in term of messages that are generated for the system to fill the pools for each initiator and building the circuits.

The simulation was done with 1000 nodes, tree height of five, group number of 32 and redundancy level of three, seven and ten. As can be seen in figure 5.6, the overhead messages that are produced per node per minutes in presence of lookup and leave events, are showed. In the original salsa system the overhead message are considered for joins, leaves, and lookups and merges events. The other two events, updating global contacts and splitting the groups would not have any message overhead since they are done by nodes individually. Here only the leave and lookup overheads are calculated since the changes that have been made in the system would only affect them.

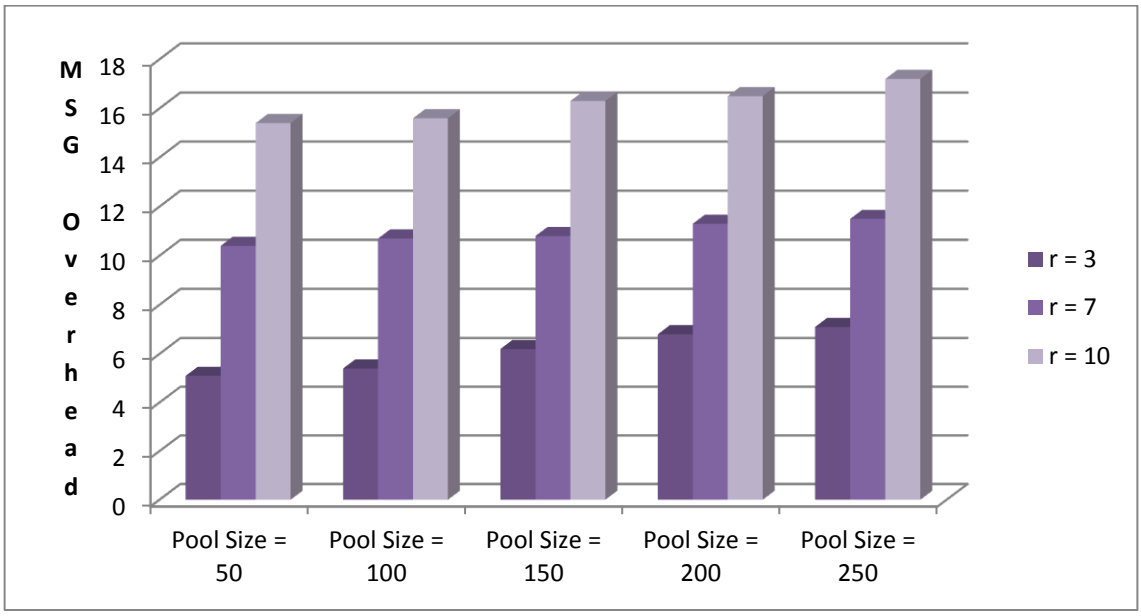

Figure 5.6 Message overhead for 1000 nodes, 20% malicious nodes, α=2, r = 3, 7, 10 and p = 50, 100, 150, 200, 250.

As shown in figure 5.6, redundancy and size of pool would affect the message overhead produced by lookup and leave events. Increasing redundancy for the same size of pool would increase the message overhead. The same will happen for figures 5.7 and 5.8. For the same redundancy level, the differences in message overhead for different pool sizes have been observed and the number of messages grows with pool size. The reason is that as the pool gets bigger, the lookup procedure goes through more lookups to fill the pools and also more churn results in more leave messages generated. The messages generated for the join events have not been discussed here since the size of the pool would have not any effect on the message generated in the join event.

The main factor in leaving message count is the pool size, as the leaving nodes need to inform all other their pair nodes in the pool when they leave in addition to where leaving nodes inform their neighbors that they are leaving (Figure 5.8).

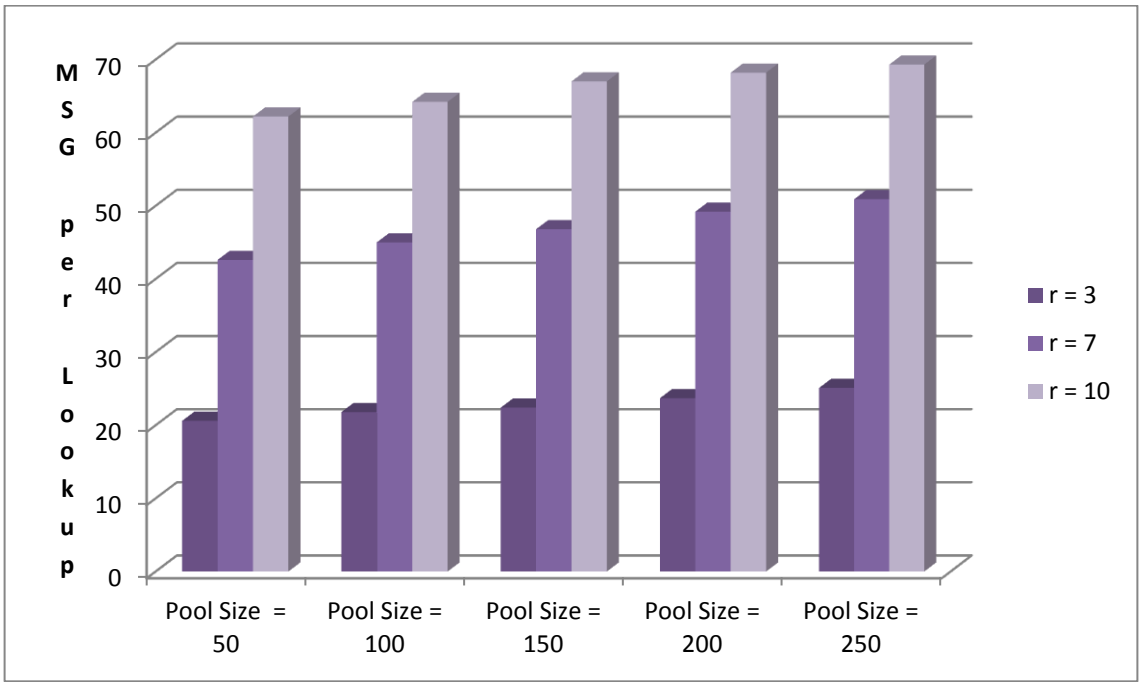

Figure 5.7 Message per lookup for 1000 nodes, 20% malicious nodes, α=2, r = 3, 7, 10 and p = 50, 100, 150, 200, 250.

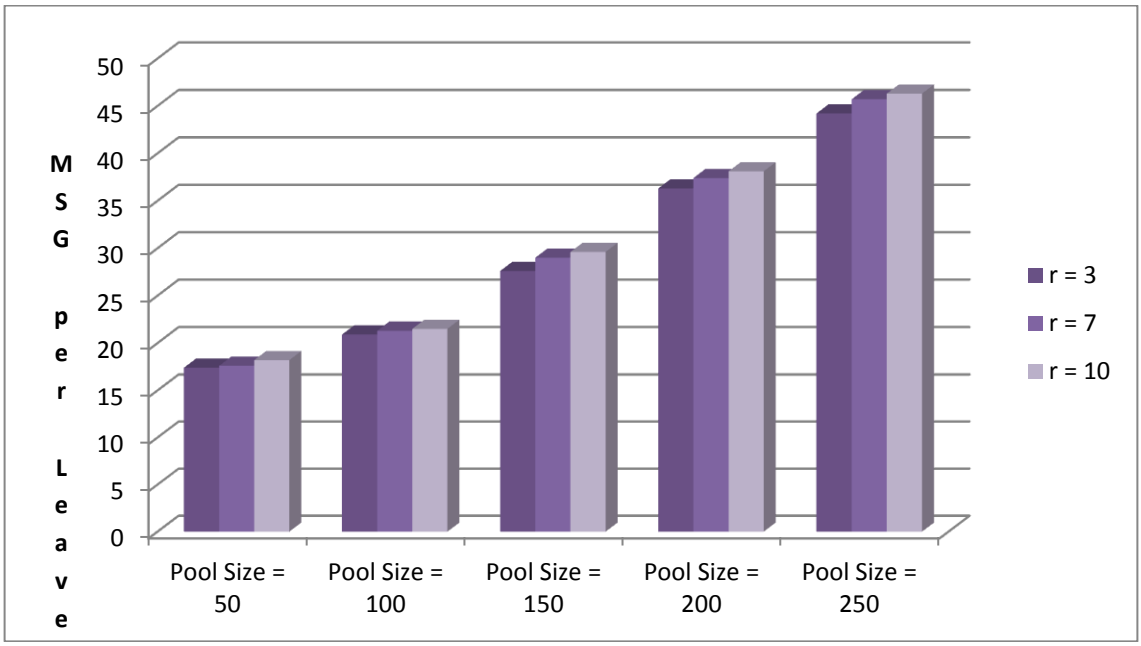

Figure 5.8 Message per leave for 1000 nodes, 20% malicious nodes, α=2, r = 3, 7, 10 and p = 50, 100, 150, 200, 250.

5.5. Mathematical Analysis for different sizes of pools

In this section, entropy is studied as another concept to quantize the chances of success of different attacks for the new system. For anonymous systems, the main target for most attackers could be to recognize the initiator of a message.

5.5.1 Conventional Continuous Stage Attack

The first attack, Conventional Continuous Stage Attack, does not happen in the new system because unlike the original system, here each circuit will be built independent of other circuits for the same initiator. Thus stages cannot be defined here.

5.5.2 Bridging an Honest First Stage Attack

For the second attack, Bridging an Honest First Stage Attack, the pair constructed from nodes A^* and B^* are assumed to be malicious. These nodes also know which pools they belong to, thus they can find the initiator as it is the owners of one of these pools. As mentioned before, the chance of discovering the identity of the initiator is 1/(number of pools the malicious nodes A* and B* belong to), since the initiator is the owner of one of these pools. The percentage of pairs that belong to a certain number of pools can be found out using the cumulative distribution function (CDF) plots obtained earlier, using this CDF and its corresponding probability distribution function, entropy, as the uncertainty in an initiator of a path, can be found.

To find out the entropy, first the curve of the CDF of how many pools each pair belongs to is fit into an exponential distribution with closet estimation.

First the entropy is calculated with r=3. There are five curves for five different values for the number of pairs in each pool. The table below gives the CDF and PDF for each of these curves, and then their entropy is calculated using the formula:

H(x) = $-\int (f(x).\log(f(x))\, dx)$ which f(x) is probability distribution function of x.

Table 5.1: CDFs and PDfs, p = 50,100,150,200,250 and r = 3

Number of Pairs in each pool	CDF	PDF
250	1-exp(-0.0065x)	0.0065 * exp(-0.0065x)
200	1-exp(-0.0074x)	0.0074 * exp(-0.0074x)
150	1-exp(-0.01x)	0.01 * exp(-0.01x)
100	1-exp(-0.015x)	0.015 * exp(-0.015x)
50	1-exp(-0.045x)	0.045 * exp(-0.045x)

Figure 5.9 shows the estimation of exponential distribution for CDF when r=3.

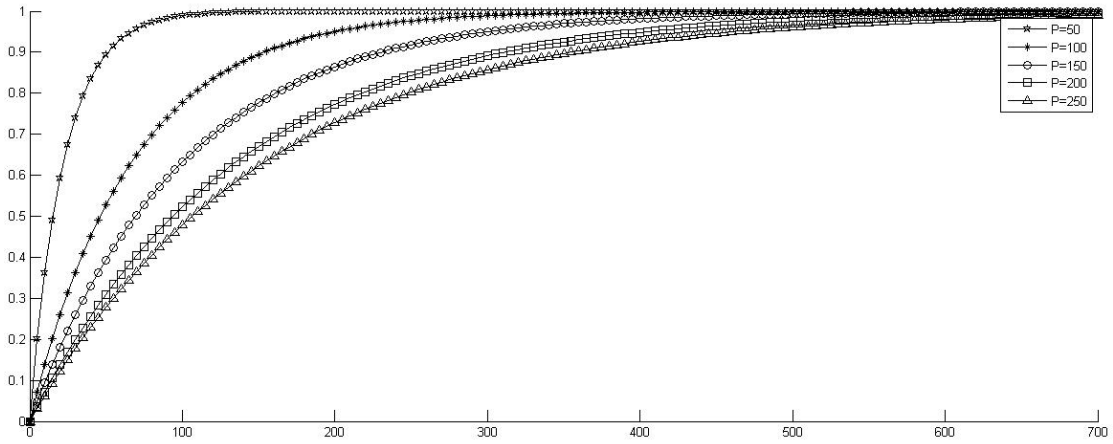

Figure 5.9 Estimation of exponential distribution for CDF, r=3.

The figure 5.10 shows the values of entropy for different number of pairs in each pool, for when r=3.

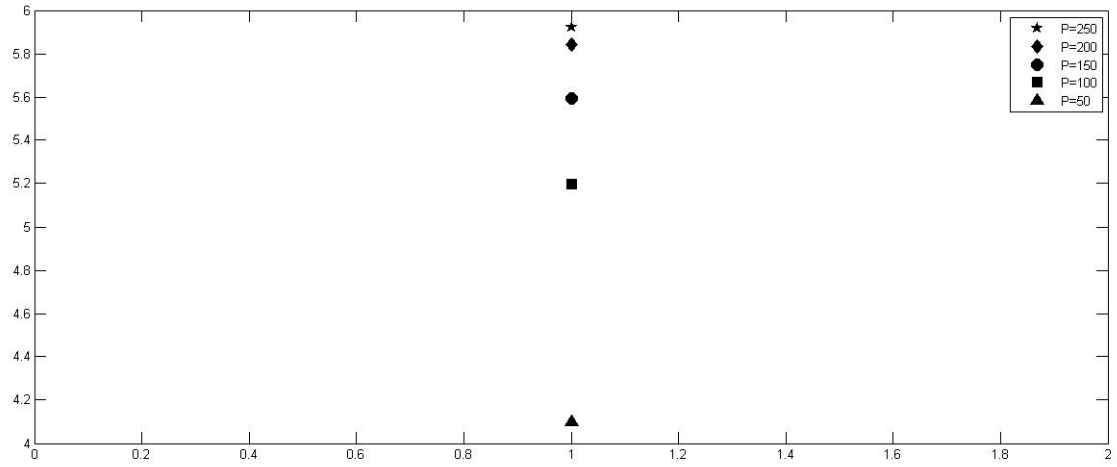

Figure 5.10 Entropy of the system, p = 50,100,150,200,250 and r = 3.

Next, the entropy is calculated with r=7. As was the case with r=3, here there are five different curves for five different values for number of pairs in each pool. The table 5.2 gives the CDF and PDF for each of these curves, and then their entropy is calculated using the same formula.

Table 5.2: CDFs and PDfs, p = 50,100,150,200,250 and r = 7

Number of Pairs in each pool	CDF	PDF
250	1-exp(-0.0032x)	0.0032 * exp(-0.0032x)
200	1-exp(-0.004x)	0.004 * exp(-0.004x)
150	1-exp(-0.005x)	0.005 * exp(-0.005x)
100	1-exp(-0.0068x)	0.0068 * exp(-0.0068x)
50	1-exp(-0.012x)	0.012 * exp(-0.012x)

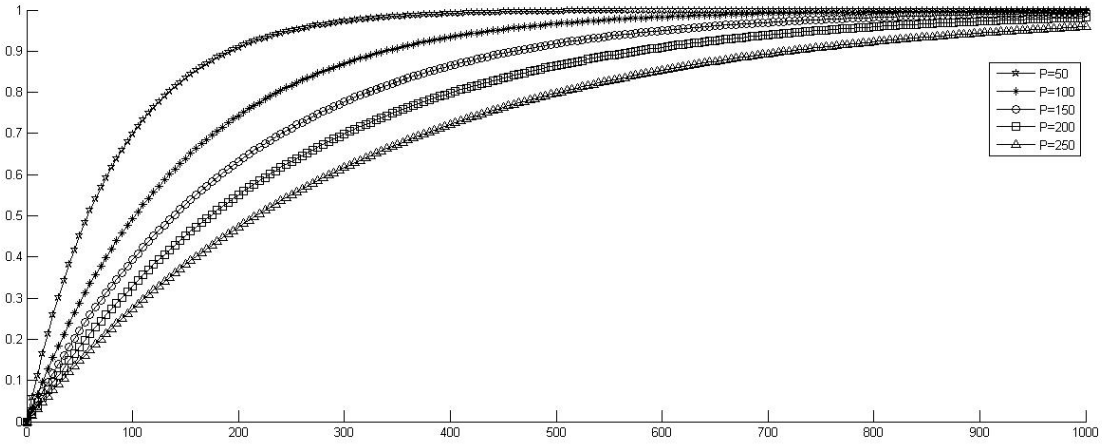

Figure 5.11 Estimation of exponential distribution for CDF, r=7.

Figure 5.12 shows the values of entropy for different size of pool, for when r=7.

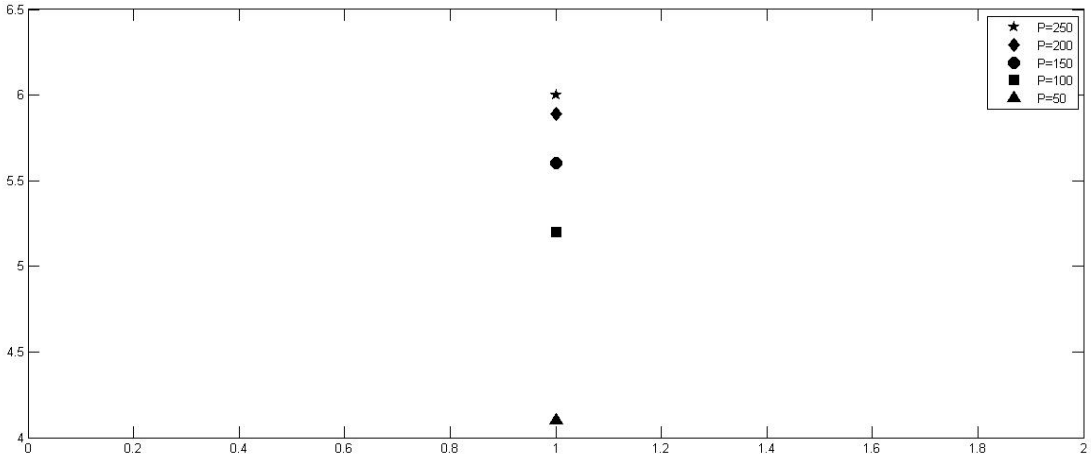

Figure 5.12 Entropy of the system, p = 50,100,150,200,250 and r = 7.

The third case is for when r=10. Here, the same five different values are used for the number of pairs in each pool, and the CDF and PDF are computed for each value, and using the PDF, the entropy is calculated and given in figure 5.10. The table 5.3 gives the formulas for CDF and PDF.

Table 5.3: CDF and PDf, p = 50,100,150,200,250 and r = 10

Number of Pairs in each pool	CDF	PDF
250	1-exp(-0.00215x)	0.00215 exp(-0.00215x)
200	1-exp(-0.00305x)	0.00305 exp(-0.00305x)
150	1-exp(-0.0035x)	0.0035 exp(-0.0035x)
100	1-exp(-0.005x)	0.005 exp(-0.005x)
50	1-exp(-0.0088x)	0.0088exp(-0.0088x)

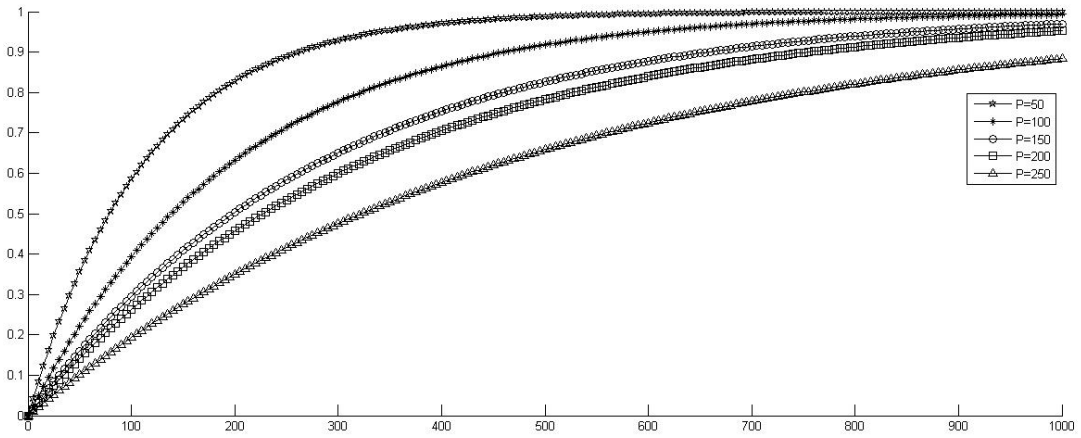

Figure 5.13 Estimation of exponential distribution for CDF, r=10.

The figure 5.14 shows the values of entropy for different number of pairs in each pool, for when r=10.

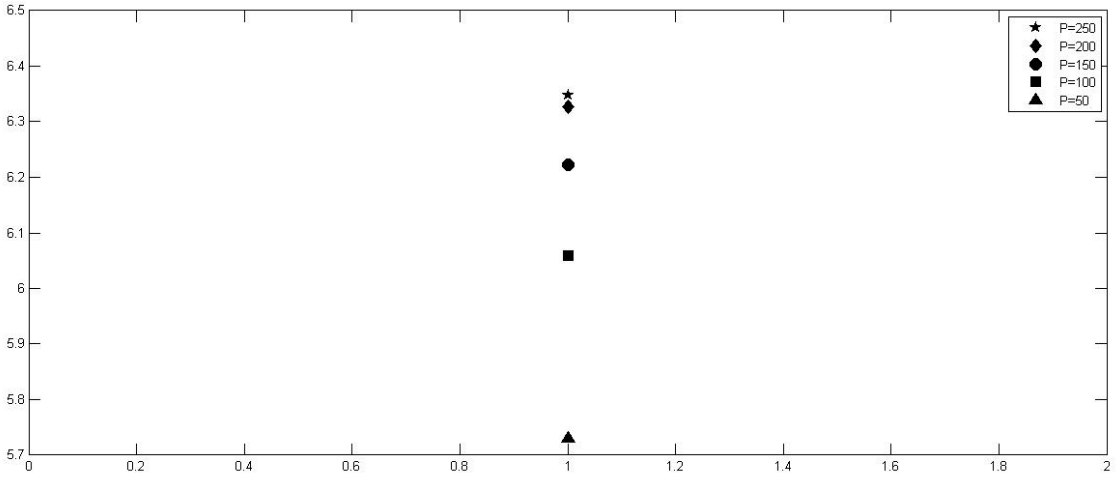

Figure 5.14 Entropy of the system, p = 50,100,150,200,250 and r = 10.

As seen in each of the figures given above, the larger the number of pairs in each pool, the larger is the uncertainty in finding the initiator of a path. This is because when there are more pairs in a pool. For a scenario when the number of nodes is fixed, there are more pairs that belong to a larger number of pools. Thus a malicious pair could belong to a larger number of pools, and hence there are more owners that could be potential initiators.

5.5.3 Bridging an Honest Stage

Another attack that the Salsa system is vulnerable to is bridging an honest stage attack. In this attack, first node and last node are malicious and the node in between is honest, yet they will be able to recognize that they are on the same path. The figure below shows this deployment of malicious and honest nodes. Node A of the circuit is honest, but the malicious node X knows its identity since they had key establishment as part of building the path. Also, node A and B are paired together so node B knows the identity of node A. node X and node B know that they are part of the same path, because there is one common node between them.

So here the problem that how often the first node and last node in the circuit are compromised has been considered. The Simulation was done with 1000 nodes in the network, 20% malicious nodes (f = 0.2) in the system and also set the redundancy level to 3. The

comparison was done for five different sets of pool sizes. The cumulative distribution function (CDF) is shown in figure 5.15. The x axis represents the percentage of malicious nodes over total nodes that have been used for choosing first node and last node to build the circuit.

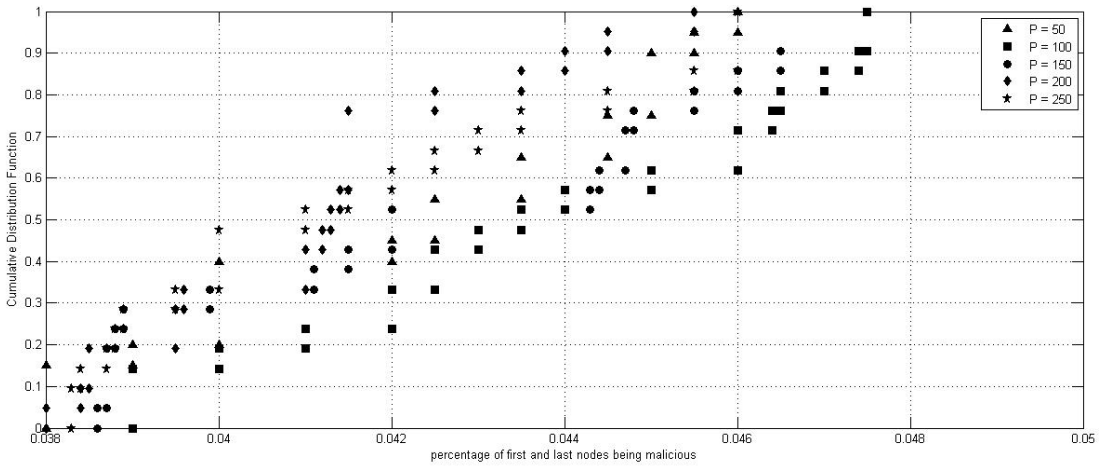

Figure 5.15 CDF of compromised first node and last node in the circuit.

The redundancy level does not have any effect in the probability of a node being compromised in first and last stage of building the circuit. So the figure 5.15 has been plotted the just for basic redundancy level of 3. As it can be seen from figure 5.15, the size of pool does not have effect on the how often the first node and last node in the circuit are compromised. In the original Salsa system this probability was close to $f^2 = (0.04)$. As can be seen here in figure 5.15, the probability is between 0.038 and 0.048 for the new Salsa system. So the probability for the first node and last node of the circuit being compromised for the improved system and the previous salsa system are close to one another.

CHAPTER 6

CONCLUSION

The new system, which is an improved version of Salsa, has some advantages over the original design of Salsa. First of all, here pairs are used instead of nodes. Stages are not defined since they were in the original Salsa, and thus the Conventional Continuous Stage Attack does not happen in the new version of Salsa. Increasing the redundancy level would not change the pair life cycle since the redundancy of the lookup process would not have any effect on leaving time of a node that happens according to an exponential distribution in the leave process of the queue.

Another advantage of the new system is that the chance of success for the Bridging an Honest First Stage Attack is significantly reduced compared to the original design of Salsa. The chance of success for this attack can directly be calculated using 1/(number of containing pools), and by increasing the number of pools that a pair belongs to, the chance of this attack can be reduced for initiators that use that pair in their circuit. Using the CDF plots obtained for the system, and approximating them with exponential distributions, the entropy of finding the initiator has been calculated, showing further that with more number of pairs in each pool, there is more uncertainty in the identity of the initiator.

As a minor disadvantage, there is a slightly larger amount of message overhead, because more lookups need to be performed for filling the pools for each initiator, and also because the joining and leaving of each node creates more churn in the system, resulting in more messages.

REFERENCES

[1] MITTAL, P. AND BORISOV, N. 2012. Information Leaks in Structured Peer-to-Peer Anonymous Communication Systems. In *Proceedings of the* 15th *ACM conference on Computer and communications security (CCS'08).* ACM, New York, NY, 267-278.List reference material here.

[2] BAUER, K.,MCCOY, D., GRUNWALD, D., KOHNO, T., AND SICKER, D. 2007. Low-resource routing attacks against Tor. *In Proceedings of the ACM Workshop on Privacy in the Electronic Society.* T. Yu Ed., ACM, New York, NY, 11–20.

[3] MITTAL, P. AND BORISOV, N. 2009. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. *In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09).* ACM, New York, NY, 161–172.

[4] NAMBIAR, A. AND WRIGHT, M. 2007. The Salsa simulator. http://ranger.uta.edu/~mwright/code/salsa-sims.zip.

[5] PANCHENKO, A., RICHTER, S., AND RACHE, A. 2009. Nisan: Network information service for anonymization networks. *In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09).* ACM, New York, NY, 141–150.

[6] WANG, Q., MITTAL, P., AND BORISOV, N. 2010. In search of an anonymous and secure lookup: Attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. A. D. Keromytis and V. Shmatikov Eds., ACM, New York, NY, 308-318.

[7] NAMBIAR, A. AND WRIGHT, M. 2006. Salsa: A structured approach to large-scale anonymity. *In Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06).* ACM, New York, NY, 17-26.

[8] KHAN, S. 2008. THE DYNAMICS OF SALSA: A STRUCTURED APPROACH TO LARGE-SCALE ANONYMITY. M.Sc. thesis, University of Texas at Arlington.

BIOGRAPHICAL INFORMATION

Anahital Davoudi was born in Iran. She received her B.Sc. degree in Computer Engineering from Amirkabir University of Technology (AUT), Tehran, Iran, and her M.S. degree with a Thesis from University of Texas at Arlington in Electrical Engineering. She has been a member of Information Security (iSec) Lab at UTA.