Secure Routing on Structured P2P Overlay
Simulating Secure Routing on Chord DHT

Mebratu Tsehayu
Secure Routing on Structured p2p Overlays
Simulating Secure Routing on Chord DHT

Mebratu Tsehayu
Abstract

Fully distributed systems offer the highest level of freedom for the users. For this reason, in today’s Internet, it is recorded that more than 50% of the packets moving in and out belong to this type of network. Huge networks of this kind are built on the top of DHTs, which follow a more structured communication compared to the other small peer-to-peer networks. Although nature always favors freedom and independence, security issues force consumers to set up their network in a centrally controlled manner. One of security threats posed on such networks is lookup attacks. A lookup attacks are kind of attacks which targets on disrupting the healthy routing process of the DHTs. Even though the freedom of peer-to-peer networks comes at the cost of security, it is quite attainable to make the network more secure, especially, it is quite achievable to gain performance on this level of attack according to the experiments carried out in this thesis. The secure routing techniques introduced have been found to outperform those without the techniques under investigation. The simulation performed for default Chord overly and the modified Chord, yielded interesting results, for dropper nodes, random lookup routs and colluding sub-ring attacks.

Keywords: p2p, secure routing, DHT, Chord
Acknowledgements

“Everything under the sun has a beginning and an end”, however, I have been asking myself why would the end is a lot better than the begging? In the beginning you only expect to reach the end, however, the end takes you to a new beginning, which you might not have imagined. That’s the surprise of life. The end would never have been attainable without the guidance, support, and advice of the special peoples in my life. It would have been neither a beginning nor an end, had it not been for the people around me. I wish that I could mention them all here by name, to thank all of my friends, families, colleagues. There are some people who deserve special gratitude for what they have done during this part of my life. Professor Tinging Zhang, my examiner and a lecturer in many of my graduate courses, and Dr Stefan Forsström at Mid Sweden University. A special thanks also goes to my friends living in Sundsvall. Last but not list, I would like to thank my family, living in my home country, and all my friends who supported me to achieve my goals.
## Contents

Abstract ........................................................................................................................................... i

Acknowledgements ....................................................................................................................... ii

1  Introduction................................................................................................................................... 1
  1.1  Background and problem motivation ................................................................. 2
  1.2  High-level problem statement ......................................................................... 3
  1.3  Concrete and verifiable goals ............................................................................... 4
  1.4  Scope ......................................................................................................................... 4

2  Theory ......................................................................................................................................... 6
  2.1  Unstructured Peer-to-peer Overlays ................................................................ 6
    2.1.1  Gnutella ........................................................................................................... 7
    2.1.2  Napster ............................................................................................................ 7
    2.1.3  Freenet ............................................................................................................. 8
  2.2  Structured Peer-to-peer Overlays ................................................................. 9
    2.2.1  Chord ............................................................................................................. 9
    2.2.2  Content-Addressable Network .................................................................. 13
    2.2.3  Pastry ............................................................................................................ 14
    2.2.4  Viceroy .......................................................................................................... 16
  2.3  Security Threats Posed on Structured P2P Overlay network ......... 17
    2.3.1  Attacks on Data Forwarding ................................................................. 17
    2.3.2  Attacks on ID Mapping ............................................................................. 18
    2.3.3  Attacks on routing table maintenance .................................................. 19
  2.4  Related Secure Routing Techniques ......................................................... 19
    2.4.1  Redundant Routing ..................................................................................... 20
    2.4.2  Octopus ........................................................................................................ 21
    2.4.3  Halo .............................................................................................................. 22

3  Methodology ......................................................................................................................... 22
  3.1  Secure Lookup Techniques ........................................................................... 22
    3.1.1  Candidate Lookup Attacks for the Evaluation ................................ 22
  3.2  Proposed Secure Routing ............................................................................. 23
  3.3  Mechanism for Implementing Proposed Secure Routing .................. 23
  3.4  Analysis of Proposed Secure Routing Solution ................................. 24
  3.5  Theoretical Analysis of the thesis ............................................................ 24

4  Design ....................................................................................................................................... 26
  4.1  The Approach in Proposed Solution – Secured Node ...................... 26
    4.1.1  Hop Verification ......................................................................................... 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.2</td>
<td>Backtracking Mechanisms</td>
<td>30</td>
</tr>
<tr>
<td>4.1.2.1</td>
<td>Modified closest preceding node</td>
<td>32</td>
</tr>
<tr>
<td>4.1.2.2</td>
<td>Updated Find Successor method</td>
<td>33</td>
</tr>
<tr>
<td>4.2</td>
<td>Joining Nodes</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Finger Table Update</td>
<td>35</td>
</tr>
<tr>
<td>4.4</td>
<td>Assumptions and Threat Models</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Implementation</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>The Existing Chord</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>The Modified Chord</td>
<td>42</td>
</tr>
<tr>
<td>5.3</td>
<td>Threat Models</td>
<td>44</td>
</tr>
<tr>
<td>5.4</td>
<td>Simulator</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>Results</td>
<td>49</td>
</tr>
<tr>
<td>6.1</td>
<td>Dropped lookups</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Random lookup routs</td>
<td>53</td>
</tr>
<tr>
<td>6.3</td>
<td>Sub-ring lookup routs</td>
<td>55</td>
</tr>
<tr>
<td>6.4</td>
<td>Standard Deviation and Pruning Parameter Effects</td>
<td>58</td>
</tr>
<tr>
<td>7</td>
<td>Discussions</td>
<td>62</td>
</tr>
<tr>
<td>7.1</td>
<td>Results</td>
<td>62</td>
</tr>
<tr>
<td>7.2</td>
<td>Ethical Deliberations</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions</td>
<td>65</td>
</tr>
<tr>
<td>8.1</td>
<td>Achievements and Gimmicks</td>
<td>65</td>
</tr>
<tr>
<td>8.2</td>
<td>Contribution and Impact</td>
<td>67</td>
</tr>
<tr>
<td>8.3</td>
<td>Future Work</td>
<td>67</td>
</tr>
</tbody>
</table>

References...69
Terminology

Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>STDV</td>
<td>Standard Deviation</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>SHA</td>
<td>Secured Hash Algorithm</td>
</tr>
<tr>
<td>CAN</td>
<td>Content Address Network</td>
</tr>
<tr>
<td>API</td>
<td>Application Programmer Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Leave</td>
</tr>
</tbody>
</table>
1 Introduction

In the last nearly two decades, an immense achievements have been attained with respect to efforts to liberate computer networking from the traditional centralized scheme for file sharing purposes. In particular, file sharing has never been as efficient and advanced as in the current distributed systems file sharing applications. In this regard, the pioneering peer-to-peer network Napster, developed in 1999, is a sort of groundbreaker project for generations of structured p2p overlays that came later on.

The advent of the first generation of p2p networks has brought enlightenment in terms of how to add another logical networking layer on the top of an existing physical network. In logical connections, users in the top overlay network interact each other in a fashion quite different from the way hardware appears to communicate. In addition, from the very nature of p2p overlay networks, there is nothing called service level agreements as in the client-server architectures. Furthermore, there is hardly any distributed mechanism to enforce such agreements. What is more, in most cases, enforcing such rules requires a central body for authentication which will eventually result in partially distributed systems.

In parallel development, the security of structured p2p networks has increasingly been challenged by adversaries. Following the wide popularity of structured p2p networks, adversaries have deployed different attacking strategies to break into and exploit such paradigms. In this respect, social anarchies have found the open nature of fully distributed systems easy prey to exploit, compared to client-server architectures.
A lookup attack is one of the security threats that lie in the overlay network which prevents DHT-based p2p networks from being widely deployed. It is performed by exploiting the weakness in the routing mechanism of the overlay network even the existence of a fraction of malicious node situated in a random place in the network leads the DHT lookup request to process incorrectly by disrupting the lookup process. Hence there is a need for study the lookup attacks and provide a mechanism for resolving the problem.

In this thesis, we investigate the security threats posed on a lookup service of structured p2p overlay. Though not the ultimate goal, we shortlist the security threats and present how the attacks can be carried out. Finally, to remedy these concerns, we propose a secure routing technique and simulate the proposed secure routing technique.

1.1 Background and problem motivation

An overlay networks is a popular distributed network technology as they provide a high availability of distributed systems to a minimal cost. Distributed hash tables (DHTs) provide efficient and scalable lookup mechanisms for locating data in an overlay networks. However, the security of the network is a high concern because of the luck of central control, due to the nature of the system, any malicious node can join and leave the network. As a result, malicious peers may take the identity of others, modify, drop, misroute lookup requests, or even collude to deny the availability of target data.
Although the attacks on the structured peer-to-peer networks range from cryptanalysis to reverse engineering, particularly harmful attacks in this case have been those targeting the lookup service of the DHT algorithm. Furthermore, these attacks which target the lookup service, also known as routing attacks aspire to achieve denial of resources, which is the very aim of these security compromises. To that end, malicious nodes achieve disruption of proper responses by just refusing to respond to the coming requests. Malicious nodes could also provide incorrect information about the lookup and eventually the request will end up bouncing all over the network with no result. Finally, in a little bit more organized manner, malicious nodes could create their own ring with overly network, which in this case, any request coming to a malicious node could end up within this sub-ring resulting in no proper response.

Therefore, in this thesis, attacks on lookup services will be thoroughly covered and simulated also solutions to perform secure routing will be proposed and implemented.

1.2 High-level problem statement

Fully distributed systems offer flexibility and scalability when it comes to the ultimate purpose of networking. However, scalability and other advantages of distributed systems come at the cost of security. In this thesis, the aim is to carry out research on mechanisms to perform secure routing on structured p2p networks. Furthermore, the specific aim of the thesis has been to propose secure routing mechanisms for structured overlays. To this end, malicious activities in their general categories, i.e. drop, tamper and inject, have been investigated. In addition, mitigation techniques have been proposed, and then the proposed techniques have been simulated. Finally, the implemented techniques for secure routing have been analysed with respect to performance gains.

In this thesis the security issues regarding peer-to-peer DHT networks will be examined and modifications of mechanisms for defence against the routing level threat will be implemented and evaluated.
1.3 Concrete and verifiable goals

The goal of the thesis is not only to gain knowledge on the mechanisms of secure routing, but also to propose a new and better secure routing technique for structured overlays. Security threats on lookup services could be carried out on a level that could hinder the normal operations of a considerable number of nodes within a system. Therefore, as an ultimate goal, security threats on lookup services are worthwhile studying. Accordingly, the concrete and verifiable goals in this thesis have been outlined as in the following:

More specifically, the thesis proposes to meet these goals:

1. Investigate three secure routing techniques on structured peer-to-peer (p2p) overlay networks.
2. Propose secure routing mechanism for structured p2p overlays and Chord overlay in particular.
3. Simulate the proposed secure routing techniques. In this case, simulation of the attacks themselves is inevitable.
4. Analyze the performance gains of the proposed techniques on the lookup service. In this respect, performance gain refers to the security and anonymity aspect of the proposed technique.

1.4 Scope

The target of this thesis is to study threats posed on lookup service of Chord overlay in particular. In other words, the aim is to come to conclusions on security considerations for structured p2p overlay lookup service threats. In actuality, simulation of the techniques of secure routing demands the simulation of the threats, however, simulation of the threats is not the goal, therefore the thesis has no aim to implement any attack rather the threat model will be designed based on the ultimate behavior of the attacks: a dropper, miss-router and colliding malicious node. In this case, therefore, the development of simulation of these three attacks is within the scope. What is more, in this thesis three different attacks posed on the lookup service has been investigated...
thoroughly and thus all conclusions, recommendations, illustrations that follows are entirely based on the Chord overlay and the attacks.

Moreover, this thesis has no aim to develop mitigation mechanisms other than those cases targeting the lookup process. However, in the process of carrying out the simulation, the assumption is that the overlay networks are healthy, except the fraction of malicious node posing a threat on secure routing.
2 Theory

Peer-to-peer overlay is a distributed collection of autonomous end-system computing devices called peers that form a set of interconnections known as an overlay, to share resources of the peers, such that peers have symmetric roles in the overlay for both message routing and resource sharing [1]. Peer-to-peer (p2p) networks have evolved enormously in the last couple of years beginning with a server based p2p network (Napster) to the advanced DHTs, for example Viceroy. Moreover, in general, based on the key features of the distributed networking technology, p2p networks are categorized into two broad categories: structured and unstructured. With respect to the fashion of communication within the p2p network, these are divided into four different types: centralized p2p, hybrid, pure and DHT [2]. As far as the recent advancement of distributed overlay networks is concerned, there has been huge development with regards to scalability, resiliency, and self-organization. Nevertheless, the problem of security and privacy is hitherto the shortcoming of the p2p network achievements.

In this section, literature reviews on the generations of p2p networks have been presented. In each of the generations, typical representatives have been presented in detail. Furthermore, major security threats on the p2p overlays have been elaborated. In the final section, related works from researchers who proposed secure routing techniques have been presented.

2.1 Unstructured Peer-to-peer Overlays

This category includes networks based on flooding or random walk routing. Most p2p applications on the Internet are based on such unstructured overlays. Small-scale phenomenon applications are typical for such technology. Early p2p networks lack some of the crucial features of recent distributed network advancements, for example scalability. In this section, the most influential and typical representative of this generation overlays, i.e. Gnutella, Napster and Freenet, are presented.
2.1.1 Gnutella

Gnutella is one of the first pure p2p file-sharing overlays and has remained one of the most popular systems to date.[1]. Gnutella uses simple flooding to carry out lookup for resources. In the event of new node join and disconnections, PING messages are used to explore the status of a node. All new arriving QUERY and PING messages are forwarded to all neighbors with the exception of the node sending the message. The procedure continues to a maximum of seven hops assuming Time-to-Live is at least one. In this case, if the message is coming again to the node which already received the message, further flooding does not take place. The earliest version of Gnutella (for example; Gnutella 0.4) did not take into consideration scalability as a performance issue. However, the recent versions (Gnutella 0.6) are built on super peer architecture to improve the scalability of the structure. The earliest version of Gnutella was based on flooding in combination with a backtracking mechanism for locating resources in the overlay.

![Figure 1](image)

Gnutella supports traditional client-server architecture, however, the most striking feature is its decentralized peer-to-peer [3] aspect.

2.1.2 Napster

Napster was a server based peer-to-peer overlay. In this case, the server is not a sort of conventional server, where content is stored and provided via the server, as in a client-server architecture, rather it was a ma-
chine where peers’ IP addresses are kept for lookup purposes. In contrast to Gnutella 0.4, which is a pure peer-to-peer network, Napster is a server based peer-to-peer overlay, which is known as a centralized peer-to-peer (see Figure 2). Napster is considered as a pioneering work in this area, which inspired all subsequent peer-to-peer networks. Napster relied on the central server for lookup, therefore, whenever a node wanted to locate a resource, it would find the IP address and port number of the target node in the central server. However, the content itself was transferred out of band using other applications, such as FTP, and the transfer was pure peer-to-peer.

![Centralized peer-to-peer architecture](image)

**Figure 2 Centralized peer-to-peer architecture**

### 2.1.3 Freenet

In contrast to similar distributed file sharing mechanisms for example Gnutella, Freenet was designed to meet security, anonymity and deniability [3], although trust and efficient search are still open issues. Each node in a Freenet keeps fixed-size routing table entries that store links to other peers. Furthermore, one of the strengths of this protocol is the fact that each node keeps information of immediate neighbors. Freenet relies on steepest-ascent hill climbing algorithm with backtracking until the
request TTL is expired. Each node tries to route to the intended destination through the closest matching routing key (peer identifier). If the first hop fails, then the resource will be forwarded through the next closest routing key in the routing table.

Freenet caches objects that have been retrieved from the previous queries in order to increase the probability of a successful response in the next query.

2.2 Structured Peer-to-peer Overlays

The challenges (for example most notably scalability) of the earliest p2p content distribution networks have now been approached through a sort of geometric routing technique using distributed hash tables (also known as DHT). Distributed Hash Tables (DHTs) are distributed systems that allow efficient lookup of identifiers and routing to corresponding nodes [4]. It is a service that maps keys in a flat identifier space onto nodes in a network of peers [5]. This category of peer-to-peer networks is known as structured peer-2-peer networks. Technically, the term *structured* refers to the fact that P2P overlay network topology is tightly controlled and content is not placed at a random peers but at specified locations that will make subsequent queries more efficient.

2.2.1 Chord

Chord keeps a logical ring of nodes, and these nodes keep neighbor pointers spaced on logarithmic interval around the ring. Each node maintains predecessor and successor links on the Chord ring (See Figure 3). Chord routing table is known as the Finger Table [1]. Chord [7] achieves high performance in the lookup and better scalability compared to Gnutella relying on just one operation which is given a key that it maps to that particular node. It adapts a somewhat different approach when it comes to traditional name and location service mapping. In the latter case, the mapping is direct between the keys and values, in which values could be an address or a document.
Moreover, Chord uses consistent hashing in order to yield high efficiency and achieve load distribution. Using the variant of consistent hashing used by Chord, if for example, in an overly network with $N$ nodes and $K$ distribution of keys, then there is high probability that each of the nodes will be responsible on $(1+\varepsilon) \frac{K}{N}$ keys. Furthermore, in an overlay network of $N$ nodes, there is a high probability that only $O \left(\frac{K}{N}\right)$ keys are moved to/from joining node/leaving node.

Resources and nodes are associated by an identifier. An identifier of a resource is known as key whereas the complement on the node side is known as ID. In the Chord overlay, the key value pair $(k, v)$ is hosted by a node whose identifier (ID) is greater or equal to $k$ (in this case the node is referred to as successor of $k$ keys). In the Chord identifier space shown in Figure 3, each of the nodes keep a unique (key, value) pair. Furthermore, in their corresponding routing table, which in the case of Chord is known a finger table, the nodes keep list of information, which points out to other nodes on the identifier circle.
Given an identifier circle with 7 nodes, the identifier space contains 3 bit and finger tables contain at most 3-1 entries. An example of a finger has been shown in Figure 3. In a node n with a finger table containing a list of entries, the i\textsuperscript{th} row points to the (n + 2\textsuperscript{i-1}) mod 2\textsuperscript{m} node. So i=2 represents the 2\textsuperscript{nd} node (N10) next node to n=8 (N8). i=3 points to N12, i=1 always points to the immediate successor.

For Chord space identifier with N nodes, lookup for a particular node results in average $O(\log N)$ routing hops. According to [9], the lookup process takes on average $\frac{1}{2} \log (N)$ steps.

**Finger table for node 1**

The i\textsuperscript{th} entry on the finger table at node n contains the identity of the first node s, which succeed n by at least $2^{i-1}$ on identifier space where 1≤i≤m and m is the number of bits in the key/node identifier space.

The first successor of a node in Chord identifier space is the immediate successor in the identifier circle.

For example: let m = 3, n=1 as shown in Figure 4
The next successor = n + 2^{i-1}

Finger table for node 1: 1 + 2^0 = 2, 1 + 2^1 = 3, 1 + 2^2 = 5

Therefore, the identifiers 2, 3, 5

The first successor of the identifier 2 is 3 as this should be the immediate successor of node 2. The successor of identifier 3 is 3 and the successor of identifier 5 is 0.

<table>
<thead>
<tr>
<th>Start</th>
<th>Interval</th>
<th>Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[2,3)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>[3,5)</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>[5,1)</td>
<td>0</td>
</tr>
</tbody>
</table>

Methods:

- Stabilize(): this function is invoked when a general node failure is encountered. This will repair successors by augmenting a live node instead of failed node.

- Notify():

- Fix_Fingers(): this removes failed nodes from the finger tables.

When a node wants to join the Chord identifier space, the new node first needs to choose an identifier n1. This identifier is chosen randomly through consistent hashing, and the next stage is to locate node n, which is already part of the Chord identifier space. Node n1 then queries for node n’s ID, and then n1 is able to retrieve the successor s of n. Then n1 notifies the successor of n of its presence and update of the predecessor of s to n occurs. The final stage is to build a finger table for the new node n1. To that end n1 queries the successors for n + 2^1, n + 2^2, n + 2^3, etc. iteratively.
Proper departure and general node failure differ. When a node wants to depart, it should transfer the keys to successors and notifies its successors and predecessors. However, when a node failure occurs during a look up, the querying node chooses the next best preceding node from its finger table. Only a short timeout is needed to confirm failed node, so this task does not affect the lookup process significantly. When such a failure occurs, the method Fix_Fingers() is invoked to remove the failed finger from the finger table.

One of the important features of the fully distributed networks is the way the system handles events such as new node arrival, node failures and node departure. When it comes to self-organization, Chord introduces a stabilization protocol [2]. The function stabilize() validates and carries out updates on the successor pointers. For example, node $n$ requests its successor for its predecessor $p$. If it $p$ returns $n$, then $p$ and $n$ agree that one is predecessor while the other is successor.

2.2.2 Content-Addressable Network

The Content-Addressable Network, also known as CAN introduces a multidimensional identifier space, compared to Chord, which relays on a single dimensional identifier space. The usage of d-dimensional space makes it more flexible to choose the shortest path possible to get to the intent ended resource. Therefore, CAN provide more efficient lookup. CAN features design optimizations as compared to related protocols for example Chord and Pastry [2].

The identifier space of the CAN is a d-dimensional version of the identifier space of Chord. The geometrical representation of such an address space takes d-torus with each node having its own zone (part of the d-torus). For example, for d=4 each data is assigned with coordinates $<w, x, y, z>$
Example of 2-d identifier space adopted from [11]

For a d-dimensional space and n partitioned zones, the average routing path is \((d/4)(n^{1/d})\) hops and each of the nodes maintains \(2d\) neighbors[11].

### 2.2.3 Pastry

Like Chord DHT, Pastry achieves a fully distributed structured peer-to-peer network where objects could be efficiently located and messages efficiently routed. Nevertheless, the routing in this case is based on the numeric closeness of the nodes. The fact that the physical proximity of the node is considered, besides the routing hop, yields in better efficiency [15].
Figure 5 Pastry identifier space adopted from [11]

Pastry’s node state is divided into three: The *routing table*, the *leafset* and *neighbourhoodset*. The routing table is similar to fingertable and the leafset corresponds to the successorlist. However, neighbourhoodset contains the nodes related in terms of network locality.

Network locality metric is an important factor in routing; Pastry measures scalar network proximity metric. Proximity metric could be routing hops or actual geographical locality of nodes.

Routing takes place in a two stage process.

1. Check if the destination key is within its own leaf set. If the destination key is found in its own leaf set, then query is forwarded to nearby leaf set.

2. If the key is not found in the leaf set, then the routing table is used to traverse longer distances.
2.2.4 Viceroy

Viceroy takes on a similar ring identifier space as in Chord, however, the identifier space is divided into hierarchies called levels. Therefore, each node is assigned a level in addition to ID when it joins. However, nodes are not allowed to have the same ID’s even if they are on different levels [13].

Like a node in Chord DHT, every node in the Viceroy keeps successor and predecessor nodes. In addition, each node chooses a level $l$ randomly in such a way that within n servers, each level of $\log N$ is selected with an equal probability. For a level $l$ node, two edges are added connecting it to the nodes at level $l-1$. The first edge known as down-right at a distance of $1/2^l$ and the other known as down-left at a close distance. In addition, depending on the value of $l$, an up-edge is added for a node at a level $l-1$. Finally, each node is assigned with level-ring links, see Figure 6.

The routing procedure is divided into three stages:
Secure Routing on Structured p2p Overlays
Simulating Secure Routing on Chord DHT
Mebratu Tsehayu

1. The query is forwarded to level-1 along the uplinks.

2. The query traverses along a downlink to the destination. On each level it chooses a downlink which leads to a node closer to the destination node.

3. When it reaches a node without a downlink, the query is forwarded along the ring-level and the successor lists until it finds the identifier.

2.3 Security Threats Posed on Structured P2P Overlay Network

In a fully distributed system relying on the well-known DHT lookup techniques, important features, for example, scalability, self-organization and routing efficiency, come with one particular heretofore open problem. Security and privacy vulnerabilities with peer-2-peer overlays are the challenges to combat. The fact is that the open nature core characteristics of p2p overlay make such networks vulnerable to a number of attacks. Furthermore, according to [2], these threats could arise from breaches on application layer and those on the networking layer.

Miguel Castro et al. [14] list three requirements for secure routing: (1) Secure assignment of node identifiers, (2) Secure routing table maintenance and (3) Secure message forwarding. Therefore, based on these requirements, routing attacks on structured peer-to-peer could be categorized into three broad categories: Attacks on Id Mapping, Attacks on Data Forwarding and Attacks on Routing Table Maintenance.

2.3.1 Attacks on Data Forwarding

The ability to carry a message from a source node all the way down to the destination node depends on the neighboring peers. In other words, in a fault-free structured p2p network, a packet sent from one end of the network will eventually be delivered to the destination node after going through few hops. However, when there is malicious intent and a node or a route is modified, then the fate of the packet is to be dropped or diverted. The message drop will eventually result in disruption of the
correct lookup process. In this particular case, the attacker could alter the content of the message, or alter its routing table to disrupt the operation or take advantage of a location to take control of certain routes [1]. Attacks on data forwarding could be avoided using multiple routes to send duplicated messages to the destination node. In this case, however, there is a performance trade-off as the duplicated messages could burden the system.

2.3.2 Attacks on ID Mapping

A node ID attack is a kind of overlay network attack where a malicious user gets a specific ID and frauds genuine nodes. Once the malicious node secures the intended node identification, then it is easy to disrupt correct routing.

Sybil Attack

According to [14], a Sybil attack takes place when an entity is able to secure more than one identification. The attacker forges multiple identities and subverts the healthy redundancy mechanism. A successful Sybil attacker is able to control a fraction of the entire overly. The adversary introduces a large number of corrupt entities, which are controlled by the adversary, resulting in a compromise of the security properties of the entire system, or performance degradation. When it comes to the actual sabotage carried out by these malicious entities, the attack takes place entirely by impairing the existing lookup. An adversary could either disrupt or degrade the performance of a DHT lookup service using the following two strategies:

- **Non-cooperation** [5]:
  Malicious nodes do not provide any information to other nodes.

- **Flooding** [5]:
  Malicious nodes, when prompted for a request, provide another malicious node as a reply
2.3.3 Attacks on routing table maintenance

The routing table maintenance mechanisms create routing tables for the joining nodes and maintain the tables after creation. Castro et al [14] suggests secure routing table maintenance as the second most necessary factor to achieve secure routing. It iterates that in an ideal scenario each routing table keeps on average a small fraction of entries in their routing table, which might point to malicious nodes. However, attackers could increase the fraction of bad entries during routing table maintenance.

Eclipse attack

The Eclipse attack is similar to the Sybil attack where a single malicious peer impersonates multiple identities and appears with multiple identifications. Nevertheless, an Eclipse attack is carried out by a modest number of colluding nodes which eventually separate victim nodes and set up malicious nodes as genuine neighbors. In this case, a successful attack manages to eclipse a victim node by mediating between the victim node and the neighboring nodes. In extreme cases, an Eclipse attack results in taking control of all the overlay traffic, enabling arbitrary denial of service or a censorship attack [16].

According to [17], there are two requirements for out an Eclipse attack. Firstly, attacker should choose a set of specific IDs for themselves so that the attacker can control the nodes closest to the target. Secondly, attackers must be added to other nodes’ routing tables.

In a DHT network, each peer holds a list of addresses of nodes in its finger table. The nodes in the figure table, route to the right destination. Adversaries could alter the listings within the figure table which leads to failed routings for malicious advantage. In this scenario, the adversary is actually altering the correct information.

2.4 Related Secure Routing Techniques

In the last couple of years different techniques have been proposed to secure DHT lookup. A majority of these techniques (for example, Halo and Salsa) relies on redundant lookup in order to avoid misinformation
[20]. In addition, some other techniques (for example, Myrmic) employ central authority to secure routing tables.

Miguel [19] Castro et al. proposes a secure routing primitive in order to carry out secured routing. A secure routing primitive includes (1) Secure assignment of node identifications (2) Secure routing table maintenance (3) Secure message forwarding. Nevertheless, in general, the strategy followed to secure routing depends on the performance requirement of the application or the overlay. Therefore, the approach in this case is a tradeoff between performance required and security to be achieved.

2.4.1 Redundant Routing

Castro et al. [14] proposes certified IDs and constrained routing tables to achieve better performance in combating malicious activities targeting node ID assignments and routing table maintenance respectively. Furthermore, when it comes to secure routing primitives, secure node ID assignment and secure routing table maintenance are not enough in order to make sure the routing is secure; secure message forwarding also needs to be done. One of the mechanisms to ensure proper message forwarding has been to use diverse routes to reach the replica roots. By using diverse routes, each of the replica roots is able to receive at least one copy of the message. In this case, the cost of the redundant routing is an important factor which matters for the overall cost of the approach.

Redundant routing in Chord [14]: If node S sends m message to the key of node R and the routing failure is positive, then the following steps are taken.

1. S sends m to all the successors and predecessors of R via different routes (via its own successors and predecessors). This causes the messages to use diverse paths.

2. Any correct node that receives one of the messages and has R’s root in its neighbor set returns its nodeID certificate and the nonce, signed with its private key, to S.
3. In a set N, S collects the $\frac{1}{2} + 1$ nodeID certificates numerically closest to x on the left, and the $\frac{1}{2} + 1$ closest to R on the right. Only certificates with valid signed nonces are added to N and they are first marked pending.

4. After timeout or after all R replies are received, S sends a list with the node IDs in N to each node marked pending in N and marks the nodes as done.

5. Any correct node that receives this list forwards S’s original message to the nodes in its neighbor set that are not in the list, or it sends a confirmation to S if there are no such nodes. This may cause steps 2 to 4 to be repeated.

6. Once S has received a confirmation from each of the nodes in N, or step 4 has been executed three times, it computes the set of replica roots for R from N.

2.4.2 Octopus

The main goal of Octopus is to provide secure lookups and thus avoid malicious nodes targeting the lookup process [18]. Octopus does not provide a solution which avoids all malicious activities, e.g. a Sybil attack, but the following malicious activities have been considered:

*Lookup Bias Attack:* the last queried node could replace its successors with a malicious node, which means the malicious node is responsible for the lookup results.

*Lookup Misdirection Attack:* Malicious nodes could replace honest finger tables with malicious nodes; the anonymity of a node could be compromised. Malicious nodes collect information about the target lookup node.

*Finger Pollution Attack:* Each node could be misleading adding malicious nodes in its finger table when updating the finger table. Eventually, polluted finger tables contribute to biased lookup results and misdirection [18].
Octopus relies on a mechanism to sort out malicious nodes on the routing table and then to punish these nodes. The process of sorting out malicious nodes runs independently of the lookup and in secret [18].

2.4.3 Halo

Halo (High-Assurance LOcate for Distributed Hash Tables) proposes disjoint redundant search in order to efficiently and securely lookup target resources when facing an attack. The underlying idea is: “We make the observation that the target of a locate operation exists in several routing tables of nodes distributed in the DHT”. Therefore, the algorithm does not carry out redundant searches on every node instead it uses specific nodes known as knuckles.

The locate operation for key $k$ computes the knuckles keys $k_1$ and $k_2$ (see Figure 7). Node $v$ initiates two locate operations for key $k_1$ and $k_2$ beginning with $finger1$ and $finger2$. Eventually, the locate operation finds the nodes ($Knuckle1$ and $Knuckle2$), which contain information about the target resource.
3 Methodology

In this section, the method adopted to carry out the thesis is explained in detail. Research strategies, development tools and efforts applied to reach at the conclusion are presented. Furthermore this section aims to approach the concrete and verifiable goals outlined in Chapter 1 scientifically and conventionally. Therefore, to achieve the intended goals, the following steps, presented in sub sections have been taken in order.

3.1 Secure Lookup Techniques

In the literature review, a good knowledge base is the anticipated gain, therefore; attacks and threats posed on the structured peer-to-peer (p2p) are elaborated. In this case, attacks compromising healthy routing has been given due emphasis. In the corresponding section, Sybil, node ID, Eclipse, DDOS, unsolicited messages attacks, and routing table poisoning attacks are included. Moreover, in the same corresponding section, structured peer-to-peer (p2p) networks are explained. In this case, Gnutella, Napster, Gia, Chord, CAN, Tapestry, Pastry, P-grid, butterfly and Viceroy are presented. In addition, the same section provides knowledge on the simulator for Chord Overlay network that has been implemented for the experimentation later on.

3.1.1 Candidate Lookup Attacks for the Evaluation

The attacks aimed at hindering the normal routing on a structured p2p overlay does not just follow simple or the same techniques, but use varying techniques depending on the actual intention of the malicious activity. The malicious techniques range from feeding incorrect information in finger tables (e.g. Routing Table Poising) to colluding to disrupt communication between nodes (e.g. Eclipse attack). Therefore, in this case three harmful attacks from each of the four general categories (i.e. interruption, interception, modification and fabrication) of malicious strategies by effects on victims have been made. In this case, interruption, modification and fabrication have been represented by the specific tactic of attack. However, interception, which happens when
Unauthorized accesses takes place, is just the first step of all three candidate attacks in this section, so we found it is redundant to include unauthorized access. Hence, attacks on Data Forwarding, Routing Table Poisoning and colluding sub-ring attacks are the candidates for examining the secure routing techniques to examine.

3.2 Proposed Secure Routing

The first and concrete step for prevention has been to detect the existence of malicious nodes in the system. This has been achieved by defining system invariance and verifying them. The verification of the system invariance helps to detect the malicious node, and the second approach is observation of a lookup progress, allowing the querying node to observe the progress of the lookup progress. This will give a controlling mechanism to the routing query; a simple example of these mechanisms is that the distance of the successor should get closer. This means, the distance of the lookup has to become closer to the seceding node at every step. In order to carry out the stated detection technique, it is mandatory that there is a back and forth communication between the querying node and the node that is responsible for the next lookup. Hence, the iterative routing mechanism is used.

After the detection of the malicious node there has to be a mechanism for finding an alternative way so that it can deliver the query to the targeted node. There also has to be a means of teaching the healthy nodes about the infected node so that they will omit the malicious node during their lookup process. To achieve this, the querying node has to back track to the healthy nodes and ask the next closest node to give information about the infected node so that the requested node mark the infected node as a void and sends the next closest node to the requester. This process will continue until it finds the non-faulty node from the list.

3.3 Mechanism for Implementing Proposed Secure Routing

In this thesis, given the project time and because the work is carried out by a one person, it would not be feasible to start-up all the inputs of the actual research work from scratch. In addition, as this is a master’s
thesis the central aim is not to just develop a tool, but to employ implementation outputs for the research work that follows. Therefore, considering these important points, implementation of the secure routing solutions, has been simulated in the developed Chord Simulator. Moreover, new libraries have been added on the top of the existing package. The development of simulator uses existing Chord simulators as the knowledge source. The additions include security threats (the attacks mentioned in section 3.1) and secure routing solutions. The details of the effort applied to implement and simulate a secure routing solution have been explained in section 3.6.

3.4 Analysis of the Proposed Secure Routing Solution

The aim in this section is to evaluate the performance gain of the implemented solution. In more precise words, the performance gain in this case refers to the security and anonymity aspect of the Chord DHT Lookup. Given the current Chord Simulator and the security compromises implemented, the strategy in this case has been to investigate correct routing. So to this end, the number of successful and failed lookups will be compared with the number of successful and failed lookups a scenario when the security routing solution is applied. Furthermore, the analysis includes all three different general categories of attacks (drop, tamper and inject). Therefore, the statistical analysis has been presented with, for example, when a dropper is involved; a lookup message is tapered with; and injection of malicious messages is carried out on lookup. The results in the corresponding section of analysis have been represented by informative illustrations and tables.

3.5 Theoretical Analysis of the thesis

In this master’s thesis, the aim is to increase the security of lookup techniques, compared to the current Chord (default chord). Moreover, the aim is also to have a simulator designed to simulate the three kinds of threats; one threat from each of the main categories of attack strategies (i.e. interruption, interception and modifications). These are shortlisted in the first chapter in the overall aim section. Therefore, at the end of the research work, the mitigation techniques have been learnt
and comparisons illustrated using graphs. In the comparison and evaluation, performance gain (i.e. lookup success rate) amid the attacks on the lookup is the parameter to be analyzed.
4 Design

In this chapter, implementation of the algorithm of the proposed solution is presented in detail, with illustrations and figures. The approach in the design process is discussed thoroughly. In addition, the components (functions) of the algorithm are explained briefly. The diagram in Figure 8 generally illustrates the overall tasks carried out in this section. The assumptions that have been made in the proposed solution and the threats we intend to defend against are discussed.

![Diagram showing the overall tasks carried out in this section](Image)

**Figure 8** The overall tasks carried out in this section.

### 4.1 The Approach in Proposed Solution – Secured Node

The solution proposed in this section is based on the idea that is suggested in paper [21]. In the paper, Robert Morris et al suggested several design principle for securing distributed hash table, out of these suggestions –defining system invariant and verifying them has been used as a basis to achieve the goals mentioned in section 1.3.
In this thesis, the solution has been approached by utilizing the basic Chord matrix property ($|A-f| \leq |A-B| \&\& |B-f| \leq |A-B|$), where A and B are two successive peers, and f is a pointer reference (finger pointer) that ends up among the two consecutive nodes [19]. During the lookup, faulty nodes/malicious nodes are avoided through evaluation of each of the hops by comparing the distance between the finger pointer and the ID of its successor from the average distance between peer A and B.

Therefore, the calculated average distance between consecutive nodes is used to evaluate the counting hop during the lookup process. The computed average distances are extracted from locally known statistical data that are gained from the finger table entry of the forwarding hop in the Chord ring. Based on this information, we examine the distance of the given hop from the finger pointer. If the distance of comparison is in the range expected, then the hop is considered to be healthy and we use this hop to progress the lookup to the next phase. Nevertheless, if the given hop is faulty we mark the node as a faulty node and go back to the node that provided the hop and appeal the next closest node in the finger table entry. This procedure is repeated until a valid routing path to the destination is found or all the entries in the finger table have been visited. This process is called *backtracking*.

The ability to observe and control the flow of the route during the routing process is achieved through iterative routing. In the iterative routing method, the requesting node contacts each hop in the routing path and requests the next closest node that leads to the destination. However, in the recursive routing, the lookup performing node sends the request into the closest preceding node from its finger table, and then the request will be routed from one hop to another towards the destination. Even though the recursive routing is faster than iterative routing, it does not provide a controlling mechanism for the requesting node over the routing process as in the iterative routing.
4.1.1 Hop Verification

According to [11], all nodes participating in the Chord DHT form a finger table containing \( m \) entries, where \( m \) is the bit length of the identifier. Each entry in the finger table contains a pointer value, \( f_i \), which is obtained from \( (id + 2^i) \mod 2^m - 1 \). It is known that the finger pointer in the \( i^{th} \) row in the finger table points to a space that is in between the entries in the finger table and \( i^{th} \) level precursor node. To achieve the hop verification technique we have adopted and modified the technique that is proposed in [19]. In the proposed system identification of hop is carried out one at a time for the entire route. However, in our approach the verification is carried out for each hop encountered during the routing process. To do this we have modified the original Chord finger table by adding the list of successor and predecessor identifier nodes. This is done for each entry in the \( i^{th} \) level of the finger table, we then collect statistical data from each entry node in the finger table and calculate an average distance between successive node ID’s. We use this average numerical difference and compare its distance between pointer reference value and entrance node identifier. Based on the result gained from the comparison, we validate the hop that we encounter during a lookup process until it reaches its destination. The following flowchart shows the hop verification technique.

![Hop Verification Flowchart](image)

Figure 9 Hop Verification Flowchart
The illustration in Figure 9 shows the algorithm has three inputs: level_index (the level of the index used for the finger table entry), firstNode (ID of the node which provides the required hop), and lastNode (ID of node at last hop). The finger pointer of level_index contains the first node that secedes \((\text{id} + 2^i) \mod (2^m - 1)\); the distance has the value of the numerical difference of firstNode and lastNode. The STDEV parameter is a standard deviation of the distance samples scaled by a system parameter sdMode providing a mechanism for balancing of false negatives and positives by determining the amount of the standard deviation that the distance of finger pointer over the average distance is allowed to be. The scaling effect of the standard deviation is discussed in chapter 6.4. The average distance is a method that provides the average distance computed from the distance samples that collected from each entry in the finger table. By summing up the average distance together with the standard deviation parameter, we get the acceptable distance limit. Having the sum value, we compare the actual distance and the expected distance. If the distance is in between the expected distance, then valid is returned, indicating success of the verification. But if it happens to be outside the range of the expected distance, then false is returned indicating the node is faulty.

It has been already mentioned that the original Chord finger table entries have been slightly modified so that each entry contains the ID of the predecessor and the identifier of the list of successor nodes.

<table>
<thead>
<tr>
<th>Table 1 Modified finger table format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
</tbody>
</table>

These extra entries enable us to obtain a distance sample by the time of the finger table entry is updated, so that we can calculate the average distance of consecutive node identifiers. As discussed in [19], the distance of identifiers of nodes in the chord ring is distributed exponentially. As in exponential distribution property [23], the difference of the
mean and the average of consecutive values are very close. This property has been used to avoid large distance samples. This happens due to the fact that the attacking nodes provide fake a predecessor and successor to the attacked node. The result in the distance obtained from the faulty nodes is greater than it really is. We make use of a pruning mechanism to cut out the samples that are likely from the faulty nodes. The following chart in Figure 10 shows the algorithm used to calculate the average of the distance from samples that we have collected from finger table entries.

![Diagram](image)

**Figure 10**  The average numerical difference

### 4.1.2 Backtracking Mechanisms

In the Chord lookup service, the source nodes search the destination from its finger table, and if it is not in the finger table it forwards the query to the closest preceding node (or predecessor- the predecessor node is the nearest node to the destination identifier among all entries of linked neighboring nodes [entries from the finger table]). This process will be repeated until the query reaches the destination. After going through these steps, the verification of the preceding node passing the query to the destination node is carried out. If it fails to pass the verification, we go back to the previous node that provides the hop and ask for
the next closest preceding node. This process will be followed thorough-
ly until it finds the healthy hop or has visited all entries in the finger
table. This verification mechanism is known as backtracking. To achieve
the backtracking mechanism we have employed and made our en-
hancement in the find_successor() function.

During this process, all nodes that are determined to be flawed or that
have used up hops will be put away in a transitory "black list" which is
created for each lookup request. So, we will never utilize this node again
in the lookup request. Next, we go back to the finger table and seek the
next closest preceding node. Depending on the verification output, the
process goes forward and backward until it obtains a valid hop from the
routing entries. Moreover, the lookup process has to guarantee to
sidestep the nodes on the route to the destination in case all possible
immediate preceding nodes happen to be faulty. The greatest concern in
this a case is that nodes further from the destination are more prone to
have out of date successor data. In this case, we need to utilize
bypassing as a last option. Bypassing is achieved by searching the
preceding node for the address of the node sought after in its finger
table. When find we check the validity and make sure the key sought
after is in between the destination node being sought and its
predecessor, and if the node satisfies this condition we bypass the
intermediating nodes and reach the destination. The following figure
illustrates the backtracking bypassing scenario.
As we see in from the Figure in section A, the backtracking algorithm of the queering node coloured yellow validates the next hop that fails to pass the validation, then it falls back and appeal for the next closest preceding node in the finger table entry and validates it. In this case the hop that is provided is a non-faulty node coloured grey, the queering node then uses this hop to route around the faulty node and, as is expressed in the figure, the reaming hops are valid so the query progresses untill it reaches the destination coloured green.

In Figure11 in section B, a bypassing scenario is shown where the immediate preceding nodes are faulty nodes. The queering node coloured yellow sends the request to the next node which advances the query and the destination node sin in it figure table, in this case the query needs to be sent to the immediately preceding node but all nodes preceding the destination happens to be faulty, so it searches a preceding node which passes the verification and contains the reference to the destination then the hop goes around the intermediate hops coloured red and reaches the destination coloured green.

4.1.2.1 Modified closest preceding node

The modified method first checks weather the ID falls between the hopId and its successor. If the check returns true and the level is $m-1$, where $m$ is bit length in identifier space, then hopId is the destination node so we return the successor of the hopId as destination. However if the level is
less than m-1, then we are not looking for the destination, instead we are looking for the next closest node, so it returns null and it continues searching for a legitimate hop. The next step is to find whether the ID falls in the range between finger table entry and its predecessor, then we will add it to *bypassNode*. Nevertheless, if we are unable to find other valid preceding nodes, then we return it as a destination. In all other cases, we search for the hop until we reach the specified level and return entry from our finger table. The pseudo code in Figure 12 illustrates the modified algorithm and the default one.

![Fig 12 modified pseudo code for the closest preceding node]

### 4.1.2.2 Updated Find Successor Method

The *find_successor()* algorithm is modified in such a way that it makes use of verification and closest preceding node algorithm. By doing so, the modified *find_successor()* algorithm will be able to provide the correct successor of interest. In the modified algorithm it accepts parameter *Id* representing the identifier sought after and it keeps the detailed information of all hops that are used during the lookup process, including *nodeId*, *fingerTable*, and the *levelIndex*. Moreover, we have a list called *blackList* which stores the nodes that fail the verification (a list of malicious nodes identified during the verification stage and the nodes that are failed to provide a next hop from their routing table).
The routing list will be loaded and looped through until the routing node is in the list. For every loop, the first node is taken from the list and the value of levelIndex is set. The closest_preceding() algorithm provides the closest node from the given finger table. The next step is to verify whether the node that has been retrieved is valid or invalid. To do this we make use of the hop_verifier() algorithm, and we make sure the given hop is not in the routing list. If it fails to pass the verification test then we save the current hop in the maliciousList and we backtrace to the next nearby node.

If the node fulfills the condition then the next task would be to check if the given node is in the maliciousList. We just push the node, the levelindex will be one step closer to the top, i.e. the value of levelindex will decrease and look for the next closest preceding node from the finger table. However, if it is not in the maliciousList, then we check either if the given node is the successor to the Identifier. If the node is, then this means the node is the destination and returns it. Nevertheless, in cases where all route lists are visited and a correct destination could not be retrieved, Null is returned, which indicates the routing has failed. Figure 13 shows the pseudo code for the modified successor function.

---

Fig 13 Modified pseudo code for finding successor
4.2 Joining Nodes

A couple of improvements should be made so that new nodes can join the Chord network effectively in the vicinity of malicious nodes. Therefore, we modified the secure joining process in such a way that the joining node needs to know a collection of uncompromised bootstrap nodes, which exist in the Chord ring to safely go along with it. In the event that the joining node attempts to join with faulty bootstrap nodes, they can be adjusted into any place in the Chord ring where they wish to be placed. These nodes must be figured out, and this is the main situation where we require nodes to trust each other but there should be no other conditions that trust except here.

We have avoided using a single bootstrap node, instead a list of bootstrap nodes is used. This is because of the fact that even with powerful security instruments set up, some inaccurate lookups may occur in the vicinity of attackers. Assuming a specific destination to populate the introductory finger table, it requests these bootstrap nodes to perform a lookup of the finger pointer identifiers. Then the node with the identifier that is closest to the finger table pointer will be utilized.

4.3 Finger Table Update

A situation to be taken into consideration is when a finger table is updated and receives a faulty upgrade for a finger table entry. To avoid this slight modification to the original finger table is made, updating the process in the following manner we can prevent the uncompromised node from having an incorrect finger table during fix-finger().

The following explains the finger table update.

- During finger table update, if the node is similar to the entries of the finger table from the old, the node will acknowledge it and update its finger table.
- If we find a finger entry located at a distance from the older entry, then it needs to verify the old entry and the successor list nodes. This is to ascertain that all the preceding node identifiers
are out of the system. It acknowledges the update if none of these nodes are available in the system, otherwise we throw out it and utilize the nearby succeeding node, which we have information about the previous finger table before.

4.4 Assumptions and Threat Models

In order to evaluate the proposed mitigation techniques, well established knowledge based on the attacks to be mitigated is crucial. It also necessitates a need to state assumptions that should be used to accomplish the proposed mechanism. Therefore, this subsection provides information about the kinds of attacks to be mitigated and the assumptions taken.

The proposed defense mechanism is designed based on the following assumptions and considerations:

- The nodes in the system cannot choose their identifiers, rather we assume it will be provided by a certification authority, or it will be obtained from the hashing of the IP and port number.

- If the minority nodes in the system are the attacking nodes this that we are not going to provide a mechanism for other attacks other than attacks, that are compromising the lookup success.

- There are N live nodes with node identifiers are distributed over the identifier space in a Chord ring

- The nodes truest no one except in the bootstrapping node during the joining process

For the evaluation of the presented mitigation technique the following threats that impair the lookup service have been implemented. Evaluations of the security against these threats have also been performed.
• **Lookup request dropper:** these attacks receive the request and they simply drop the request. This is the simplest type of attacks and our system is expected to recover from such attacks only by using the backtracking method. In this type of model we have considered the healthy nodes; they drop the lookup request for several reasons, and we have provided a mechanism to distinguish the malicious dropper nodes from the healthy nodes that drop the request.

• **Misrouting lookup query:** These kinds of malicious nodes acts as the cooperating healthy node because they receive and forward each incoming lookup routing request, but they forward it to other random nodes. These attacks are very difficult to detect because it looks like everything is working as it should.

• **Sub-ring attack:** A number of malicious nodes collude together to form a sub ring and forward the lookup request to each other to end the lookup request at a faulty node. The colliding malicious nodes use a correct a finger table and another faulty finger table which contains a list of a successor malicious node which is close enough from the correct finger table entries. These faulty nodes are very difficult to detect since they accept the lookup request as a healthy node and show progress to the destination by providing a next node, which is close enough from the correct closest preceding. Whenever the faulty node receives a lookup request it forwards the request to the other faulty node using the faulty finger table, this process continues until the routing query ends at a faulty destination.
5 Implementation

In this chapter a brief discussion regarding to the unmodified Chord and the modified Chord protocol implementation is presented. The implementation of three types of threats is briefly explained. Finally, the implementation of the Chord simulator is briefly discussed.

Figure 13 illustrates the default Chord protocol and modified class together with threat models that are modelled to represent the most effective lines of routing overlay attacks.

- **MainChord:**
  It refers to the existing Chord Protocol that implements the basic chord features and functionalities.

- **DropperNode:**
  Dropper nodes are nodes which drop all requests they receive.

- **RandomNode:**
  Random nodes are the kinds of nodes that play part in malicious activities by forwarding incoming requests to other random nodes or other infected nodes.
• **ColludingNode:**
  Faulty nodes form a group and collude together by forming a sub-ring and routing the lookup among each other which means that the request ends up at a malicious node.

The threat models are designed in the same manner for the ModifiedChord. LookupSource is used for nodes to build their finger table quickly; it is also used by the malicious nodes to build their modified finger table.

Java is used to develop all the simulations and other implementations related to this project. Therefore, although there are a number of open and free development environments, we have chosen to use Netbeans 7.1 IDE for its ease and familiarity in the previous courses. In the processes of implementations of MainChord and the proposed solution, there are additions to the basic features mentioned in the protocol itself. Hence, this requires points that clearly differentiate the behaviour of nodes that is going to participate in both case, and how they differ one from another.

### 5.1 The Existing Chord

The existing chord DHT protocol is implemented in MainChord class as in [22] suggested. The basic functionalities and components that are expected from Chord lookup service are covered for completeness of the protocol including the recursive lookup service. Although these two kinds of lookups are implemented, we used an iterative lookup mechanism rather than recursive mechanism as discussed in the design chapter. So in the iterative lookup we have made use of a local class for pairing the next hop with a Boolean representing whether the hop is a final hop.
or not. The iterative lookup is the only way that our implementation differs from the default Chord protocol. The MainChord class uses its default constructor to create the first Chord node in the network and it uses a parameterized constructor to create the joining node to the existing network through the given bootstrap node. Node identifiers are represented by 160 bits. A tick method is provided for each node and it is invoked periodically by the simulator whenever the nodes require to update their finger tables, stabilize it is successor information and check weather its predecessor has failed or not.

![Chord Class Diagram](image)

**Figure 13, an overview of default chord class**

The figure shows all of a `MainChord` class in the list of methods that is implemented in that class. This class also contains a method that is not required by the protocol, but we needed to have the utility methods to gather important information about this node for diagnostics purposes. Moreover, we created a node event listener list to collect lookup statistics from nodes.
5.2 The Modified Chord

The proposed solution for securing the overlay routing service is implemented on top of the modified Chord. The modified chord is created by extending the default Chord protocol and adding the required functionalities that are listed in the design chapter. Modified Chord class uses three basic subclasses that help to create a struct type for grouping different pieces of information together. The extended finger table subclass groups together multiple pieces of information about finger table entries including the identifier of the node predecessor and list of successor identifiers for each entry in the finger table. The router state stores pieces of information required during the routing lookup process and the lookup performing node will store the object of router state class in its stack and use it to route around the faulty node during the backtracking process. Chord hop is a simple struct type sub class that allows the next_hop() method to return multiple pieces of data. Two constructors are used to create a modified Chord node, the first constructor creates a new node with a random identifier and it has statistical parameters whereas the second constructor is used to create a new Chord node joining the network that contains the list of bootstrap nodes with a random identifier and which have a statistical parameters.

In the modified Chord, we have overridden different methods from the default Chord implementation including the join method so that the joining nodes can generate their finger table entries and use the one which is close to the finger pointer. We also did an override for quick converge method to build the extended finger table and we made use of the finger table data to generate the average distance and standard deviation of nodes. Next, we use a pruning method to prune our sample data set to avoid the high samples that are likely to come from the malicious node, in case the node contains faulty node in its finger table. The fix fingers method refreshes extended finger table entries whenever its service is needed.
The *find successor* method implements the basic goal of our defense solution, here the method uses the *nextHop()* method to get the closest preceding node and validates the given hop using the *verifyHop()* method if the hop is non-faulty node. If the destination is not in the current querying extended finger table then it asks for the entire finger table from the given hop, then loops it through until it finds a valid closest preceding node. In the case of failure of all the extended finger tables, the providing hop will be stored in the black list and we backtrack to the node that provided this hop, and then ask the next closest preceding node. This method goes back and forth until it obtains the destination or until it reaches the maximum amount of attempts. The *nextHop()* method serves as a mechanism for providing the next closest preceding node based on the index level indicating the number of backtracking attempts. Moreover, this method provides us with a means to find a healthy preceding node, which has a destination node in its finger table for the bypassing mechanism when the immediate preceding nodes of the destination node happen to be faulty.

The diagram in Figure 15 illustrates the implementation and construction of a modified Chord class. The default Chord serves as a super class letting the modified Chord use non-overridden methods as they are defined by the re-user, whereas the other classes shown in the figure are used as a subclass for the modified Chord protocol defining and grouping a different set of information for secured nodes.
5.3 Threat Models

To test efficiency and carry out a performance evaluation of the proposed solutions, the following three routing attacks are implemented, as discussed in the design chapter:

The DropperNode class is used to create a dropper node which drops all incoming lookup requests. The find successor finds the successor if the request is initiated from its own node, otherwise it returns null, preventing other traffic from passing through it.
The `RandomNode` class represents a threat that forwards incoming requests properly while using a faulty finger table to forward the requests to other random malicious nodes. This class maintains a separate finger table, using out-of-band lookup sources class where all the entries are faulty nodes in the system. This finger table is used to forward all incoming request appearing as if they are cooperating nodes and while preventing the requests from ever reaching the destination.

The quick converge method is used to perform a quick converge like other node do and also to build other faulty finger tables.

The `CollidingNode` class represents a sub-ring (colluding) malicious version. This class maintains a faulty node that builds a group and colludes together by forming a sub-ring in the system. The `findNextHop()` method is used to find the closest preceding node if the request comes from itself otherwise it finds the faulty node succeeding the request from the bad finger table and routes the lookup among each other, aiming for the request to end up at the malicious node.

The `isMalicious()` method indicates whether the node is faulty or not, the method is used for testing purposes during the simulation. Moreover, the random and sub-ring node use this method when they form their bad finger table whenever a quick converge method is called.

The points mentioned in this subsection clearly differentiate the default Chord (i.e. existing Chord) and the modified Chord. However, these differences do not call for a need to create different malicious activities, but only separate instances of the same attacks are applied for both. Therefore, the same kind and number of malicious nodes for the modified Chord protocol are created as for the default Chord protocol.
Figure 15 Various threat modes

The diagram in Figure 16 illustrates the classes and their methods of threat for both modified and default Chord protocols. The LookupSource class is used in both implementations as a means of out-of-band source to create a bad finger table for randomly routing nodes and colliding nodes.

5.4 Simulator

In order to perform scientific investigation on the performance gains of the proposed algorithm due to enhancements carried out in the success of the lookup process in a structured overlay network, a simulator is designed and implemented in such a way that it provides the required environment to carry out testing, comparison and evaluation of the proposed algorithm.
The illustration in Figure 17 shows the different classes that we have used to set up the simulator and run the experiments on both protocols. Moreover, the diagram shows the flow and structure within the simulator and mechanism that we have used to run the experiment.

The implemented simulator makes use of the following classes and interfaces:

*The ChordManger* class is a core class that manages the system. It sets up the network and allows different types of nodes to be added to the network and from that Chord network we run the test, during the process it uses state keeper for storing different information for the experiment purpose.

*The ChordRing* class keeps tracks of the nodes in the network and provides the necessary information and functionalities for the ChordManger.
The StateKeeper class implements a NodeEventListener interface and stores statistical information during the experiment. The NodeEventListener is an interface type class that listens for events that are happening from each Chord node and updates statistical variables.

The LookupRecorder class used is to keep information regarding all lookups and will be used when the lookup process is presented in GUI.

The ChordGUI is assumed to be is the graphical representation of the Chord DHT service and planed it displays the chord routing process graphically. We use this for simulation demonstration and debugging purposes, but the actual testing and evaluation is done by writing a batch of test files. The Cpanel class is responsible for the visual representation of a Chord ring and the ChordGUI class is responsible for presenting the ring. The actual implementation of the ChordGUI is not yet fully developed at this stage, but all definition is presented to complete the implementation.
6 Results

In this chapter the results from the evaluation of the proposed solution are illustrated and elaborated. We evaluated the performance of the Chord protocol and proposed system against the most effective lines of routing overlay attacks especially the kind of attack that drops all the lookup traffic as it passes through it, the attack that forwards the request to other malicious nodes randomly preventing the lookup request from ever reaching the target node, and, finally, malicious nodes creating a sub-ring of malicious nodes and forwarding all lookup requests by using the ring of nodes which forwards the requested lookup to end in a faulty node. Moreover, the effects of the network size on the success rate of the system have also been taken into consideration during evaluation and discussed in the corresponding section.

In the evaluation process, the measurement of the success rate of the lookup has been calculated as in the following equation:

\[
Average\ hops = \frac{\text{Lookup Success (Sought)}}{\text{Total (Lookup Request)}}
\]

In addition, the standard deviation and pruning parameters against the success rate of lookup requests in the system have been calculated.

6.1 Dropped lookups

Dropped lookups are by far the most common and attack requiring less knowledge in peer-to-peer DHT routing service. The attacker makes use of these kinds of attacks as an instrument for denial-of-services threats. Whenever the lookup is not progressing the default Chord tries to deal with this type of problems by running stabilize and check predecessor methods. The faulty nodes respond to these methods properly to show that they are not failed nodes and thus deny the service that they
are expected to do so. Therefore, the proposed solution is designed to
deal with this type of threats and failed nodes that are not recovered.
We are not required to run hop validation at this stage, instead the
iterative routing and backtracking mechanism is used to deal with such
attacks. Moreover, in order to make sure the capability of the mecha-
nisms figuring out the faults, the hop verification algorithm has been
disabled.

To carry out the evaluation and achieve the intended goals, a system
consisting of 1,000 nodes have been step up. In the system a dropper
node of droppers = \(\sum_{i=0}^{n} i = i + 20\) where \(i\) is the number of dropper
nodes in the system and \(n\) is the maximum number of nodes we expect
to have (-500 nodes). In this experiment, the reason why we expect to
have 500 nodes is that we want to make sure we travel halfway through
all of the existing nodes. In addition, for each iteration we have created
100 different networks, converged each network in a stable state and run
1,000 random lookups, in total 100,000 random lookups are initiated
from a non-faulty node. The following graph (Figure 8) illustrates the
result obtained from the running experiments.
Figure 17  Success rates under the influence of dropper nodes

The graph in Figure 18 shows the proposed lookup mechanism performed very well compared to the default Chord lookup mechanism. Even in the worst case scenario the proposed solution has managed to avoid these threats and travel through the link route around lookup. Up to 95% of the time the routing avoids threats which lie in the path to the destination. Whereas a similar experiment in the default Chord protocol showed that around 94% of the request is not successful. Therefore, based on these experimental figures the approach in the proposed solution improves the default Chord routing mechanism by 89%, for 50% dropper nodes. Furthermore, 16% and 30% of dropper nodes the success rate for the default Chord is 44% and 20%, however, the proposed approach yields 100% and 99% avoidance of dropper nodes. In general based on these comparisons, dropper nodes are better controlled in the proposed solution as compared to the default Chord. Nevertheless, all figures in the above graph have been gathered at the expense of hop counts, which are illustrated in Figure 19.
Figure 18 Average hop count used to secure dropper nodes in the system

The graph in Figure 19 illustrates the average hop counts for the modified solution. The results in the average hop count for a modified system is doubled when the dropping nodes become 25% and it is tripled in 40% and it is four times doubled when the dropping nodes control 50% of the nodes in the system. However, the success rate achieved in each case was 99%, or, in other words, the lookup requests have been successfully delivered to their destination as opposed to the default Chord where the success rate is less than 30%, showing a total difference of 71% success rate and 4.8 average hop counts. Surprisingly, in the proposed solution, 95% success rate has been achieved with 50% of the malicious nodes and the cost for hops is approximately four times the average hop resources. However, at 50%, dropper nodes’ default Chord is only expected to achieve a 6% success rate. Lastly, the same figure proves that the proposed system takes more average hops to outperform the default Chord protocol.
6.2 Random lookup routes

In the previous evaluation, hop verification has been avoided and the results showed that the proposed solution outperforms the default Chord without running hop verification. However, in this section, when it comes to random lookup routes hop verification has been considered. In the random lookup routes, malicious nodes forward all incoming lookup requests to other faulty nodes using their bad finger table entries, which makes it seems as if the lookup progress is running properly but in reality the request goes through these faulty nodes preventing them from ever reaching their destination. Hence, the hop verification and routing mechanism have to be used.

In the performance test and evaluation, a network consisting of 1,000 nodes have been setup on a simulator. A dropper node of \( \Sigma_{i=0}^{n} i = i + 20 \), where \( i \) is the number of dropper nodes in the system and \( n \) is the maximum number dropper node expected (~500). In this experiment, the reason 500 nodes are expected is that we want to make sure we travel halfway through all of the existing nodes. In addition, for each iteration, we created 100 different networks, converged each network in a stable state and run 1,000 random lookups, in total 100,000 random lookups are initiated from a non-faulty node. The maximum hop limit is set to 100, determining how many hops it expected to lookup to route around faulty nodes (we can consider it as a time to leave set). In the verification algorithm we have setup the standard deviation parameter and pruning parameters to 1.3 and 1.0 respectively. The following graph in Figure 20 illustrates the results obtained from the experiments explained in this section.
The graph in Figure 20 shows the results obtained from the experiments explained in this section. In the graph the test reveals that with the consideration of the verification technique on the lookup process the proposed solution yields a high success rate. The rates of lookup success of the proposed solution are 99% when there are no faulty nodes, while the default protocol is doing 100%. The reason behind 1% less performance is the standard deviation parameters used in calculation of hop verification. In addition, as the numbers of faulty nodes increase more and more incorrect success results are experienced due to the fact that attackers manage to provide a faulty node in the acceptable area. Nonetheless, in the lookup consisting of 50% faulty nodes, the proposed solution has a 91% success rate with 6% incorrect results only. When it comes to success rate of the default protocol, the lookup request fails whenever a faulty node occurs on the route. Most importantly, the success rate of lookup requests decreases as the number of dropper nodes increases. In the worst case scenario the default chord protocol fails again more than 90% of the time.
The diagram above (Figure 21) shows the result obtained from the average hop count. It increases as the numbers of malicious nodes increased in the system, but compared to the previous result it is reduced the average hop count. The standard deviation parameter created significant effect on average hop count.

6.3 Sub-ring lookup routes

In the previous sections, hop verification and backtracking mechanisms have been thoroughly evaluated on the default Chord and the proposed solution, the results have shown that the proposed mechanism outperforms the default Chord when faced with these two kinds of threats, in the third place sub-ring lookup routes have been bought up for performance evaluation and comparison.

Therefore, in this section, performance evaluation has been carried out on what is known as a colliding sub-ring attack. A sub-ring attack poses a complicated and challenging task for the verification. These types of malicious nodes create bad finger tables and use them to provide a
faulty hop which is close to the actual node that is responsible for forwarding the lookup query.

The performance of the proposed system is evaluated through the experiments by setting up the network as for dropper case consisting of 1,000 nodes. A dropper node of \( \sum_{i=0}^{n} i = i + 20 \) where \( i \) is the number of dropper nodes in the system and \( n \) is the maximum number dropper node expected (~500). In this experiment, the reason 500 nodes are expected is that we want to make sure we travel halfway through all of the existing nodes. In addition, for each iteration we created 100 different networks, converged each network in a stable state and ran 1,000 random lookups, in total 100,000 random lookups are initiated from a non-faulty node. The maximum hop limit is set to 100, determining how many hops it expected to lookup to route around faulty nodes (we can consider it as a time to leave set). In the verification algorithm we have setup the standard deviation parameter and pruning parameters to 1.3 and 1.0 respectively. The following graph in Figure 22 illustrates the results obtained from the experiments explained in this section.
Interestingly, the results obtained from the simulation are not quite similar to the previous scenarios. The main reason is this being the behaviour of the attack. Sub-ring attacks are difficult to detect since the attacking nodes use malicious hops from its faulty finger table that are close to the correct entry. However, considering the difficulties of detecting such attacks, the proposed solution provides a better result compared to the default Chord protocol. For example in Figure 22, for 26% of faulty nodes, the success rate for the default Chord is 45%, whereas for the proposed solution the success rate is 78%. Furthermore, it is important to consider that as the amount of faulty nodes increases, the default Chord’s success rate increases. The increase in the success rate happens because the destination for that lookup request happens to be faulty node, in this case even if the colluding nodes tries to fool the protocol it will end up at its destination.

The hop count expected when there are no faulty nodes has almost doubled compared to the default Chord (see Figure 23). The reason being the fact that the standard deviation parameter used affects the verification function to produce more false positives. For the other cases, the hop count increased as the number of faulty nodes increased.
6.4 STDEV and Pruning Parameter Effects

In the proposed solution, the STDEV parameters used in the verification algorithm are used to control the acceptable distance to the above the threshold. We used a STDEV parameter to provide a means to trade-off security against performance during hop verification. The values of this parameter are added together with average distance of finger table entries to provide acceptable distance during verification. The effects of the values of standard deviation are expressed in terms of how much false negative and false positive results are obtained in the verification process. In other words, higher values of the parameter results when increasing false positives allows faulty nodes to pass the verification, which in return decreases the probability of getting the correct destination. However, a lower value for the parameter increases false positives resulting in increase of average hop count.

During the calculation of distance samples it would be expected to derive distances from the malicious nodes that are greater than they really are because the malicious nodes can provide successor and predecessor node identifiers that are not consecutive to its own identifier. In such cases, pruning prevention mechanisms are used to cut out the
samples that are likely to be from faulty nodes. This is done by cutting the edges of our distance sample until the average distance is close enough to the standard deviation of those distance samples. The pruning parameter has also been used to scale the average distance. In this case the effect of setting the parameter to a low value leads the calculated average distance to be too low, which forces the verification to generate more false negatives. Likewise, if the parameter is set to a high value then the calculated average distance will be high, which forces the verification to generate more false positives. The more false negatives in the results the more failure of the verification. The more false positives in the result, the more incorrect lookups.

To test the effects of these parameters in the verification algorithm the following experiments are run, varying the experiment variables (see Table 3).

**Table 2 Experiment setup for scaled standard deviation and pruning parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Range</th>
<th>steps</th>
<th>System nodes</th>
<th>Sub-ring attacking nodes</th>
<th>Pruning</th>
<th>STDV</th>
<th>Network created</th>
<th>Random lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>STDEV</td>
<td>0 - 10</td>
<td>0.1</td>
<td>1000</td>
<td>25%</td>
<td>1.0</td>
<td>-</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>Pruning</td>
<td>0.5- 2.0</td>
<td>0.1</td>
<td>1000</td>
<td>25%</td>
<td>-</td>
<td>1.75</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

The graph in Figure 24 illustrates the results gained from standard deviation effects and the graph in Figure 25 shows the effects of pruning parameters.
The illustration in the figure above shows a high standard deviation results when the success rate of lookup decreases and the incorrect lookup is increased. A high standard deviation value parameter results in a higher number of false positives in the verifier algorithm, which basically means an increase of the faulty nodes to be flagged as valid nodes, thus routed incorrectly and ending up at faulty nodes. On average, 81% success rate is obtained between 0.8 – 1.7, the success rate of the lookup process, while the average incorrect lookup for these sets is 15.6% and average failures is 3.3%.
The smaller the pruning parameter the higher the failed lookup and the smaller the lookup rate. A very small number of pruning values means the average distance value tends to be low, resulting in more false negatives, which results in valid nodes to be set to be invalid. As we increase the variable, the average distance becomes high, resulting in fewer false positives and more true negatives. When the parameter is set to 0.9, the lookup success is 80%. However, it seems a sound argument to take a pruning parameter of 1.0 when a small number of nodes are compromised.
7 Discussion

The experimental results clearly suggest that the modified solution (proposed solution) outperforms the default Chord regarding the three kinds of threats examined in the experiment. In the evaluation, the simulation environment and all the tools have been developed and thoroughly tested with previously existing inputs and expected outputs. Even though the simulation environment and the tools could be a source of biased output, all the necessary precautions and testing on the simulator have been carried out. Moreover, the simulation has been carried out for two slightly different overlays (i.e. default Chord and modified Chord) and this also means the magnitude of error propagation on just one of the platforms alone is hardly possible.

7.1 Results

Under each of the general categories of threats on secured Lookups; dropped lookups, random lookups and sub-ring lookups have been thoroughly tested for interruption, interception, as well as modification and fabrication respectively on the simulation by introducing malicious nodes and groups of nodes. In each of these tests, all the necessary precautions and mandatory steps have been taken. In the first of two tests the evaluation result showed a huge performance gain. However, in the third case, even though the performance gain was an improvement to the default Chord, compared to the previous kinds of threat it showed less performance gain.

# Dropped Lookups

In this simulation the results are collected by setting up a network of 1,000 nodes and eventually increasing the number of dropper nodes to 500. For each iteration, 100 networks are created and each network covers 1,000 lookups and in total 100,000 lookups are run. The dropper
nodes are not made to pass through hop verification. The introduction of iteration routing and a backtracking mechanism in the modified solution handles the task of hop verification.

# Random Lookup Route

In this simulation the same network as in the previous scenario has been used. However, since the iteration routing and the backtracking cannot handle the hop verification, a hop verification algorithm has been considered. In the case of a dropper node, the node drops the entire request, however, the malicious node in this case forwards all requests coming to a faulty-node. So this seems as though the lookup is being carried out in a healthy manner but in actuality the routing is taking a wrong path which eventually turns into no result to the requester node.

# Sub-ring Lookup Route

As in the previous two cases, the network is setup with 1,000 nodes, up to 500 malicious nodes and 100,000 lookups. Sub-ring attacks are more difficult to detect than dropper and random lookups. In this case too, as in the random lookup it necessitates running a hop validation algorithm. Moreover, the values of the pruning parameter significantly impacts on the results of the simulation. The standard deviation for this parameter is taken from its highest success rate (~ 80% success rate, 15.6% average incorrect lookups and 3.3% average failure rate).

Simulation Parameters

The values of standard deviation are expressed in terms of how much false negative and false positive results are expected. On average an 81% success rate obtained from a standard deviation of around 1.3 have been chosen for the last two simulations. Increasing the value of the standard deviation above 1.3 results in more false positives and thus more faulty-nodes and, similarly, decreasing the value to a smaller number generates more false negatives.
The values of the pruning parameter are correlated with the success rate in the lookup process. Smaller values of the pruning parameter generates more false negatives which basically means valid nodes are flagged as invalid nodes. The higher the value of this variable the more false positives. At a pruning value of 0.9 the highest success rate is recorded (80% success rate). It is makes sense to give a little more room for performance when a small amount of faulty nodes are presented in the system as a result. We consider 1.0 and 1.75 for pruning and standard deviation parameter’s respectively.

7.2 Ethical Deliberations

The research work carried out in this master’s thesis gives better insight in to how to handle the three kinds of attacks on peer-to-peer overlays. The tasks carried out clearly showed that introduction of backtracking and verified iterative routing could improve the security of Chord Overlay network to the higher level. Therefore, the results presented in this thesis prove the practical gain of using the techniques, i.e. adding backtracking and verified iterative routing when appropriate. By far, these are the result of this master’s thesis most useful to outside consumers.

When it comes to the regulations, rules and copyrighted materials, all materials, people and company copyright rule as well as the rule of the owning institution, are treated accordingly. This paper has not been prepared for commercial purposes however the institution (i.e. Mid Sweden University) is entitled to give it away or to modify it.
8 Conclusions

In this section, the achievements are analyzed with regard to the goals outlined in the first chapter. It has been the main aim of this thesis to investigate the lookup attacks on peer-to-peer networks. In the same chapter the scope of the tasks to be carried out have been delimited to three kinds of attacks and simulation design.

In general the overall aim of the project has been successfully achieved. Each of the pieces of the overall aim of the project have been answered in separate sections based on the goals. The following subsections explain the achievements under each goal.

8.1 Achievements and Gimmicks

# Goal One: Investigate the four secure routing techniques on structured peer-to-peer (p2p) overlay networks.

The literature review has been the basis for knowledge gain in this specific area. The literature review discusses the technology of the peer-to-peer network, the security issues and existing secure lookup mechanisms. Therefore, the goal in this section has been achieved through literature review.

# Goal Two: Propose secure routing mechanism for structured p2p overlays, Chord in particular.

For the purpose of evaluation and simulation, we chose one of the well know DHTs (Chord overlay). Comparison and performance evaluation have been carried out for the existing Chord (i.e default Chord) and the modified Chord (i.e the proposed solution). The general approach in the design and implementation of the proposed solution has been presented
in the methodology chapter. In addition, the modifications and manipulations in each section of modified chord have been thoroughly explained in the design and implementation chapter.

The proposed solution introduced two techniques to increase performance gain in the looking up process among the types of threats. The first technique is known as verified iterative routing and the second technique is known as backtracking.

# Goal Three: Simulate the proposed secure routing techniques.

The proposed secure routing techniques which are verify iterative routing and backtracking are simulated, and the corresponding results are illustrated in the results section. The simulator does not display graphical output. However, evaluation is done by writing a test case of batch processing in Java and running the batches of tests. Results of the tests are saved in a file. Results of the simulation (i.e. numerical results) have been presented in the results section.

#Goal Four: Analyze the performance gains of the proposed techniques on the lookup service.

The results chapter presented the performance gains and the analysis of the proposed solution. Simulation results clearly proved the performance gains of the proposed solution as compared to the default Chord. These simulations are carried out for only for the three kinds of attacks mentioned in the same chapter (i.e. dropped lookup, random lookup route and colluding sub-ring attack). The figures suggested a huge improvement the performance gain over the default Chord especially in the first two cases.
8.2 Contribution and Impact

The contribution of this master’s thesis could be seen from two different angles. The first angle would be the view it from the perspective of the users who have already setup their system on default Chord. The second angle would be the view of the users who employ different types of peer-to-peer technology. The third angle would be a view from the technologies that dig for mechanisms help to carry out secure lookups.

From the view of the users who have already setup their system on a default Chord, the research carried out in this thesis and the corresponding results provide an opportunity to improve their system making it more secure than their existing system. The profit of making their system more secure depends on what business that system is used for. However, in general terms, performance gains introduced brings about in economic and other gains.

For users who employ different kinds of peer-to-peer overlay the impacts of the results from this thesis could not be underestimated. These users could learn the mitigation mechanisms and the techniques to make their peer-to-peer overlays more secure.

Finally, the technologies interested in digging deep into the mechanisms of securing lookups, this report provides as a reference and stepping stone to further study these techniques and other lookup threats not covered here.

8.3 Future Work

Considering the time schedule, the material resources and human resources allocated for this master’s thesis, the scope has only been made to fit within these limitations yet keeping the expected standard of the report. However, when it comes to a real-life scenario and the types of attacks, more research work is required to make lookups as secure as possible. In this thesis work, the constraints are so severe that emulations are out of the question. Therefore, the first step in future work would be to run the same test on actual hardware (test-bed). The second step would be to add other lookup threats. Last but not least there is
quite an interesting future work to consider on the SensibleThingsPlatform with regards to the kinds of attacks explained in this master’s thesis. There is also a possible future project on security aspects of such platforms intended for distributed architectures of sensory information. Therefore, these three points are worth considering for future work by others.
References


Secure Routing on Structured p2p Overlays - Simulating Secure Routing on Chord DHT
Mebratu Tsehayu


[17] IEEE PERFORMANCE COMPUTING AND COMMUNICATION, NOV. 2011, Ren Z., Jianyu Z., Yu C., Nanhao Q.,
Secure Routing on Structured p2p Overlays - Simulating Secure Routing on Chord DHT
Mebratu Tsehayu 2015-01-30

Bingshuang L., Yuan Z., “Making Eclipse Attacks Computationally Infeasible in Large-Scale DHT”, Beijing Key Laboratory of Internet Security Technology, Peking University


