APPROACHING COMMUNICATION UNOBSERVABILITY THROUGH RANDOMIZED ROUTING AND UNPREDICTABLE AGING

By

PEYMAN TAHER

Bachelor of Computer Science

Multimedia University

Cyberjaya, Selangor, Malaysia

2008

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 2010

APPROACHING COMMUNICATION UNOBSERVABILITY

THROUGH RANDOMIZED ROUTING AND

UNPREDICTABLE AGING

Thesis Approved:

Dr. M. H. Samadzadeh

Advisor

Dr. John Chandler

Dr. Nohpill Park

Dr. Mark E. Payton

Dean of the Graduate College

PREFACE

Providing anonymous communication on networks of interconnected computers is an active area of research which aims to enhance the privacy of the users of such networks. Communication unobservability is a stronger property compared to anonymity that attempts to guarantee that the adversaries will not even notice that a communication between a sender and a receiver is happening. In networks where an active global adversary may be present, it is assumed that at all times, all nodes in the network are potentially being monitored and also that at any time, any node should be treated as a possible adversary.

This thesis work introduces a set of requirements which, if followed in any system, provides enhanced anonymous communication. Furthermore, following these requirements, a new anonymous networking system was designed. This system provides communication unobservability in the presence of active global adversaries through randomized routing and unpredictable aging. Randomized routing is used to obfuscate the message routing pattern and the communication path, and unpredictable aging is used to prevent indefinite postponement of messages when routing randomly.

ACKNOWLEDGMENTS

Very special thanks go to my supportive advisor, Dr. M. H. Samadzadeh, who taught me invaluable lessons far beyond my thesis work. This thesis work, among many other accomplishments, would not have been possible without his precious insights.

My sincere appreciation also extends to Dr. John Chandler and Dr. Nohpill Park for their advice during my thesis proposal presentation and also for serving on my graduate committee.

My deep acknowledgements go to my beloved family, Nasrin, Masih, Pedram, Parisa, and my sweetheart, Nilou, for their continuous support, encouragement, and patience.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

With the tremendous increase in the use of the Internet during the past few years, there is a growing concern about the users' privacy on the net. The Internet is used by millions of people on a daily basis to check email, read news, watch videos, transfer files, etc. These activities involve network communication with other computers around the world. As long as this communication goes through the Internet which is a part of a service provided by an ISP (Internet Service Provider), it is susceptible to intervention. Every single computer that is located in the path of the communication and is used to route packets is potentially capable of monitoring, logging, and analyzing the traffic.

1.2 Problem Statement

The Internet traffic that goes through several nodes before being arrived at the destination may well contain personal and private data such as usernames, passwords, bank account information, and numerous other important potentially identifying data. This sensitive information, if accurately linked to the sender or the receiver, may introduce certain threats to either party.

Also, the way packets are routed in the Internet, ISPs could be used as a mean of censoring/filtering and/or imposing restrictions on the freedom of speech in general or monitoring the users’ activities via identifying the two end points of a communication link in particular. Thus, finding an effective and practical method of hiding the identity of nodes involved in any given communication on the Internet is increasingly important.

1.3 Assumptions and Research Objectives

Several projects have been introduced to address this issue but, due to certain technical difficulties or assumptions made by their designers, the issue of privacy and anonymous communication on the Internet is still an open problem that affects the daily lives of millions of people. In this thesis work, the problem of *Communication Unobservability in the Presence of Active Global Adversaries* was considered. In other words, it was assumed that the adversary can monitor all nodes in the network at all times, as well as take part in the communication (hence referred to as an active global adversary). The goal was to strive to provide a level of anonymity that would prevent the adversary from even noticing that any communication between a sender and a receiver is taking place, let alone the identity of the two parties involved (a.k.a. communication unobservability).

CHAPTER II

LITERATURE REVIEW

2.1 Chapter Overview

During the past three decades, several projects have been introduced to address the issue of providing private and anonymous communication. This chapter briefly describes some of the most notable such efforts as well as their advantages and disadvantages, and the main reason that these systems do not address the principal issue discussed in the current thesis work, i.e., communication unobservability.

2.2 Chaum's Mix Network

The first anonymous system, *Chaum's Mix Network*, was introduced in 1981 by David Chaum [Chaum 81]. Mix is probably the most extensively researched and implemented anonymous system [Kelly 09]. The Mix network is based on the concept of a mix which is considered trusted by all other nodes in the network [Chaum 81]. A mix basically refers to a store-and-forward device that hides the correspondences between the items in its input and those in its output. Messages arrive at a mix from various senders and the mix is responsible for hiding the order of arrivals by outputting lexicographically-ordered batches of uniformly-sized items [Chaum 81].

In a mix system, each packet is encrypted by the mix public key and consists of three parts: the address of the recipient, the message body that is encrypted by the recipient's public key, and a random string that is meant to obfuscate the encrypted message and reduce the possibility of success in case of dictionary attacks. A mix decrypts the messages upon arrival by throwing away the random string and forwarding the message body to the recipient with the given address in a way that the correspondence between the incoming message and the outgoing message is hidden.

The advantages of a mix network are the simplicity of the design and the implementation as well as a guaranteed level of communication performance due to having only a single extra layer intervening between every two communicating nodes, namely the mix itself.

The most important disadvantage of the mix system is that the mix itself has to be trusted by the other nodes in the network. By contrast, since in this thesis work an active global adversary is assumed to be present, no node in the network can be trusted. Every node is a potential adversary, including the mix. Therefore, since a mix has exact knowledge of the address of the initial sender and the ultimate receiver of a message, a mix network is not capable of providing communication anonymity in the presence of an active global adversary.

2.3 Mix-Based Anonymous Systems

Since the introduction of Chaum's mix network in 1981, several systems based on the idea of a mix have been developed. The three most widely-known such systems are briefly described below.

- A*nonymizer* is a Hyper-Text Transport Protocol (HTTP) proxy that is based on a single node mix (i.e., the Anonymizer-Server) and aims to provide sender anonymity and fast web surfing facility [Boyan 97]. The most important drawback of this system is its centralized nature. In other words, the system is as anonymous as the Anonymizer-Server node is. Furthermore, having all traffic go through a central station makes the network vulnerable to denial of service attacks [Kelly 09].
- *WebMIX or Java Anonymous Proxy* is an anonymous web surfing network proxy that is based on the notion of a *Mix Cascade* [Chaum 81] and aims to provide real-time anonymous and unobservable Internet access [Berthold et al. 2001]. A cascade mix is a series of mixes, connected in a row, such that the output from one mix is the input to another. Picking up on that theme, the input to the first mix in the cascade is the input to the mix network and the output from the last mix in the cascade is the output from the mix network [Chaum 81]. The main strength of this system is that it allows users to choose between different mix cascades [Kelly 09], but still the mix itself has to be trusted by all nodes in the network.
- *Remailers* consists of three components: 1. *Type I Remailer*, a.k.a. *Cyberpunk* [Goldberg et al. 97], which is the first widely used implementation of Mix networks [Borisov 05], 2. *Type II Remailer*, a.k.a. *Mixmaster* [Möller et al. 03], and 3. *Type III Remailer*, a.k.a. *Mixminion* [Danezis et al. 03]. The three remailers are based on the idea of a mix network that receives messages from senders, removes the message header, and uses Pretty Good Privacy (PGP) encryption [Zimmermann 95] to wrap messages and deliver them anonymously. The most

noteworthy weakness of the Remailer systems, just like any other mix-based anonymous system, is that the mix itself is aware of the location of the original sender node and the ultimate receiver node and thus, if compromised, it is capable of revealing the identity of the nodes involved in any given communication link.

2.4 The Onion Routing (Tor)

The Onion Routing is another extensively researched and mature project on anonymous communication. This system is in principle very similar to a mix cascade and provides anonymity by repeatedly encrypting the messages using the public keys of the relay nodes [Reed et al. 98] [Dingledine et al. 04]. The major issue with The Onion Routing is that the first node involved in the communication path after the original sender node is well-aware of the identity of the original sender and therefore, if compromised, the identity of the sender may be easily revealed.

In a Tor system, once a node is connected to the network, a list of other nodes involved in the network is retrieved. This list is used to create a path from the local node (the ultimate sender) to a receiver node with a number of intermediate overlay nodes (the default for Tor is three nodes). This path is stored in each packet and serves as the routing algorithm.

Packets are constructed by encrypting a message body along with the IP address of the next node, once for each node in the path. A packet is decrypted upon arrival at a node in order to determine the next node in the route.

The main disadvantage of The Onion Routing is that even a passive global adversary is easily capable of identifying the sender node via monitoring the network traffic. This is due to the fact that from a global adversary’s point of view, it is clearly visible which node has started the communication and which nodes are involved in relaying the messages from the sender node to the receiver node. Thus, The Onion Routing is not capable of providing anonymity in the presence of an active global adversary.

CHAPTER III

DESIGN GUIDELINES

3.1 Chapter Overview

A study of the existing anonymous communication systems (as outlined in Chapter II) indicated that providing anonymous communication is still an open problem and there is a need for other systems to tackle the issue of privacy and anonymity especially in the presence of active global adversaries.

In this thesis work, this issue is addressed while keeping in mind that the problem stems from two fundamental concerns:

1. All nodes in the network are potentially being monitored by the (global) adversary at all times.
2. Every node is potentially an agent of the adversary and is capable of collecting information and reporting to the adversary.

In this chapter, a set of requirements that govern the underlying design of the proposed system is introduced. The contention is that any system that follows these requirements has the potential to tackle the obstacles mentioned above and provide a solution to the issue of communication unobservability in the presence of active global adversaries.

3.2. Requirement 1: Trust NO ONE

An active adversary is an adversary that is capable of joining the network, interfering with the communication, and even compromising nodes. This property of the adversary under study here may result in the second obstacle mentioned above in Section 3.1. In other words, if the adversary is an active one, then theoretically all the nodes to which the adversary has access may be compromised and used by the adversary to collect information and report back. Furthermore, a global adversary has access to the whole body of the network, thus all nodes in the network (except for the sender and the receiver) could be potentially assisting the adversary in order to discover the identity of the two ends of the communication link.

In such a case where every node is potentially an agent of the adversary, we cannot afford to trust any node in the network with the exact knowledge of the ultimate sender, the ultimate receiver, and the contents of the message except for the two ends of the communication link. Therefore, only the receiver node may be aware of the exact address of the sender node, and only the sender node may be aware of the exact address of the receiver node. This is the most important requirement that is unique to the case of having active global adversaries.

3.3 Requirement 2: Always Encrypt Uniform Sized Packets

The overarching issue to consider here is that the contents of the message must always be encrypted at the initial sender node using the public key of the ultimate receiver. This is essential because if non-encrypted contents are transferred, it is easy for any node in the middle of the communication link to have access to the details of the communication and potentially report them to the adversary. But only encrypting the message at the initial sender is not sufficient when the adversary is capable of monitoring all nodes in the network. In this case, it is vital to prevent dictionary attacks and traffic analysis by the adversary.

To further explain this issue, let’s consider this example: node A is going to send a message to node C through node B. As mentioned above, node A has to encrypt the message using node C’s public key so that the message is only readable by node C. But let’s assume that no other encryption is involved. From the adversary’s point of view, a message with unreadable contents is sent from node A to node B, and then the same message is sent from node B to node C. The issue here is that the adversary is able to detect that the very same message that was sent from node A to B, was also sent from node B to C. Therefore, simple traffic analysis techniques can be used by the adversary to easily identify the initial sender and the ultimate receiver of any given message.

In the above-mentioned example, the only way to guarantee that the adversary is not able to detect that the same message was transferred between nodes is *salting* and re-encrypting message at each and every intervening relay node. A salt is a random value that is attached to the contents of a message before encryption in order to harden it against possible dictionary attacks. In other words, in the above-mentioned example, node A has to further encrypt the message along with a salt before sending it to node B. Then, node B has to decrypt the message, remove the salt, attach a different salt to the message, encrypt it again, and send it to node C. Finally, node C has to decrypt the message, remove the salt, and decrypt the message again using its private key to access the original un-encrypted content of the message.

Now let’s furthermore consider the same example above, only this time all traffic is salted and encrypted at all nodes in the communication path. Thus, the adversary is not able to detect that the same packet with the same fingerprint has been sent from node A to B and then from node B to C. But still it can be easily detected that a message with the same size has been sent from node A to B and then from node B to C. This may lead to some usable information for the adversary in order to identify a pattern used by the routing algorithm, and moreover to disclose the identity of the sender node and the receiver node. Hence, it is always necessary to use uniformly-sized packets to further harden against traffic analysis attacks using packet sizes.

In summary, the second requirement expects all messages to be divided into uniformly-sized packets, salted, and re-encrypted at every node in the communication path.

3.4 Requirement 3: Never Reveal Patterns in Routing

Traffic analysis can always be used to glean information from patterns in communication, even when encryption is utilized. Therefore, it is important never to reveal any patterns that may allow the adversaries to use traffic analysis techniques in order to collect and deduce information from these patterns. The use of static routes, similar to the routing algorithm used in The Onion Routing [Dingledine et al. 04], is an example of a protocol that reveals important information about the identity of the sender node as well as other nodes in the network. In the case of static routes, since the global adversary is capable of monitoring all the nodes in the network, whichever node(s) that is (are) used by the routing algorithm to relay the traffic can be monitored, therefore it is easy for the adversary to glean useful information from the pattern used to route packets.

For example, in The Onion Routing, the path used to route all communications during a session is static and is determined at the initial sender node at the time of establishing the connection. Later, this path is embedded in the packets so that each relay node would be aware of the exact IP address of the next node in the path. This technique cannot be used in cases where an active adversary is present, because any of the relay nodes may have been compromised by the adversary and used to report the path to the adversary. Furthermore, since the entire path is static and all the communication between any two nodes has to go through a particular set of relay nodes, from a global adversary point of view, the communication path between the initial sender and the ultimate receiver would be noticeably and consistently busy. This property may very well reveal the identity of the two ends of any given communication link.

In summary, the routing algorithm used to address the issue of anonymous communication in the presence of active global adversaries has to somehow follow a random pattern with two important requirements: 1. Packets have to traverse through a set of random relay nodes so that traffic analysis cannot be used to determine the communication path, and 2. Ultimately, all packets have to arrive at the receiver node, no matter which relay nodes are used to route them. These two properties guarantee that, even though the communication path is not predefined statically, all packets are routed securely.

3.5 Requirement 4: Always Use Random Noise

Unobservability, which is a state that whether or not any communication exists is not distinguishable for the adversary, is a stronger property compared to anonymity. In other words, if the adversary is not even capable of recognizing whether any two nodes are involved in any sort of communication, unobservability is achieved.

In order to add the property of unobservability to an anonymous system, random noise (a.k.a. cover traffic or dummy traffic) must be included in the system [Pfitzmann and Hansen 08]. Random noise is used to keep the network busy for most of the time, so that from a global adversary's point of view, it is not distinguishable at any point in time whether any node has started a new communication with another node, or the traffic that is being monitored is dummy traffic. This property effectively increases the level of anonymity provided by an anonymous system.

It must be stated that too much dummy traffic may result in heavy network traffic which may lead to slow communication or even packet drops. Thus, it is important to generate a level of random noise that while achieving unobservability, does not dramatically deteriorate the quality of service.

CHAPTER IV

NETWORK ARCHITECTURE

4.1 Chapter Overview

A number of requirements for communication anonymity/unobservability were proposed and discussed in Chapter III. Based on those requirements, a new anonymous system to provide enhanced communication unobservability in the presence of active global adversaries was designed. This chapter contains a detailed specification of the underlying network architecture used in the proposed system. The next two chapters present the details of the sender anonymity and the receiver anonymity modules, both of which take advantage of the networking architecture discussed in this chapter.

4.2 Network Topology

The proposed system is a structured peer to peer system in which peers are grouped into clusters of nodes called *Clans* such that any node in a clan is connected to all other nodes in that clan at all times. In other words, a clan is a complete graph of collaborative nodes. A clan that consists of *p* nodes is called a Clan-p. For example, a cluster of six nodes forms a Clan-6. A node has to be a part of a clan in order to be able to communicate with other nodes in the network.

4.3 Clan ID

Each clan in the network is uniquely identifiable by a *Clan ID*. The Clan ID is a 2-byte unsigned number (0 to 65,535) that is uniquely assigned to a clan at the time it is formed. This process is described in Section 4.7 below.

4.4 Interface Node

In order for nodes in two different clans to be able to communicate, they must have an IP address representing the other clan. This IP address, as discussed in Chapters V and VI, is then used by each party as the entry point to the other clan. Thus, each clan has to have the IP address of one of the nodes in that clan to use as the address of that clan. The node used in for this purpose is called the *Interface Node* and is randomly chosen from among the nodes in a clan.

4.5 Network Metadata Catalog

The *Network Metadata Catalog* (NMC) is a set of properties unique to the network that is maintained by all nodes in the network. The NMC contains a constant representing the network maximum clan size as well as the list of all Clan IDs in the network, along with the IP address and the public key of the interface node for each clan. The node designated as the interface node for a particular clan may be different at different times. This random non-permanent designation is meant to prevent the creation of a communication bottle neck at the interface node.

4.6 Clan Metadata Catalog

The *Clan Metadata Catalog* (CMC) is a set of properties unique to each clan that has to be maintained by all nodes in the clan. In addition to the Clan ID, the CMC contains the IP addresses of all nodes in a clan along with their public keys.

4.7 Parent Clan and Spawning a New Clan

One of the properties of a network is a constant representing the maximum clan size, which is the maximum number of nodes allowed in a single clan in the network, and is assigned by the network administrator at the creation time of the first node. Consider the situation that at the time of establishing a connection when a new node attaches to a clan that contains the maximum number of allowed nodes. In such a case, the hosting clan, called the *Parent Clan*, is divided into two new clans, each containing half of the nodes in the original parent clan. This process is called *Spawning*.

When a clan is spawned from its parent clan, a new unique Clan ID is assigned to it by the parent clan and then the NMC and the CMC for the two clans are updated to reflect the changes. Finally, these changes are propagated to all other clans in the network which would subsequently be reflected in all the nodes in the network.

4.8 Bridge Node and Connection Establishment

The *Connection Establishment* process is the set of steps that are required to be completed before a new node becomes a member of the network. In order for a new node to join the network, it has to have the IP address of at least one of the nodes in the network. This node is called the *Bridge Node*. The Bridge Node is responsible for introducing new nodes to the network. The initial node to form a network is the only exception to this rule, in that it is allowed to introduce itself to the network. Once the IP address of the Bridge Node is provided, the prospective node starts the hand-shaking process as explained below. At the end of this process, the prospective node becomes a member of the network and only then is the node allowed to communicate with other nodes in the network.

4.9 Handshaking

Handshaking is the process of introducing a new node to the network and transferring the necessary information to the prospective node so that it would be able to communicate with other nodes in the network. This process is executed immediately after that the prospective node successfully connects to a Bridge Node for the first time. Handshaking consists of the following three steps.

1. Download the CMC.
2. Download the NMC.
3. Establish a connection to all other nodes in the clan.

After handshaking, the new node is considered a member of the network and is able to communicate with any other node in the network. Note that the spawning process (if required) is the Bridge Node's responsibility and does not require any additional action from the prospective node.

4.10 Design Rationale

Principal idea behind this design is that by clustering nodes into clans, nodes would be able to obfuscate the incoming and outgoing traffic. This can be done by randomly distributing the traffic among several nodes in the clan in such a way that, from the adversary's point of view, the best guess to determine the initial sender of a message or the ultimate receiver of a message would be the entire group of nodes in a particular clan. In other words, if all the nodes in a clan are partly responsible for obfuscating the outgoing traffic from the entire clan or, in an analogous sense, if all the nodes in a clan are partly responsible for obfuscating the incoming traffic, then the adversary would not be able to distinguish between the nodes that simply relay the traffic on the one hand, and the particular nodes that are actually sending or receiving the message on the other hand.

This property is achieved by relaying the other nodes' traffic (i.e., nodes other than the original sender) in such a way that none of the nodes in the network are readily aware of the identity of the original sender. This process is described in more details in Chapter V.

4.11 Message Format

All communication among nodes in the proposed protocol are packaged in a special messaging format. This format, in the spirit of the requirements mentioned in Chapter III, was specifically designed for this system to reduce the level of information that an adversary may deduce from traffic analysis of messages in the network. Thus, in order to hide the correspondence between the messages arriving at a node and those departing from a node, which otherwise would be revealed via traffic analysis of the packets' sizes, all packets are of a uniform size of 1452 bytes.

The packet size of 1452 bytes is the *maximum segment size (MSS)* for optimum communication on TCP and IPv4 over IEEE 802.3 networks which also fits in Ethernet v2 networks [Postel and Reynolds 88]. It is calculated by subtracting the TCP header size (i.e., 20) and IPv4 header size (i.e., 20) from the *maximum transmission unit (MTU)* used in the IEEE 802.3 networks (i.e., 1492). The reason IEEE 802.3 MTU was selected instead of Ethernet v2 MTU is that the former is smaller than the latter (i.e., 1500), which means that packets generated in this system will not cause fragmentation either on IEEE 802 networks or Ethernet v2 networks, hence resulting in better performance on both networks.

A message, as illustrated in Figure 1, is comprised of the following four segments:

1. *Target Clan ID*: The 2-byte ID of the clan of which the receiver node is a member.
2. *Priority*: A 2-byte value used for *aging* to prevent the indefinite postponement of forwarding a message. The mechanism used to assign and update this value is described in Chapter V.
3. *Signature*: A 2-byte encrypted value used by the routing algorithm as a flag to determine whether or not the message is intended for the receiver (for more details see Chapter V).
4. *Payload*: The actual message body encrypted by the receiver's public key and padded to fill up the allocated space.

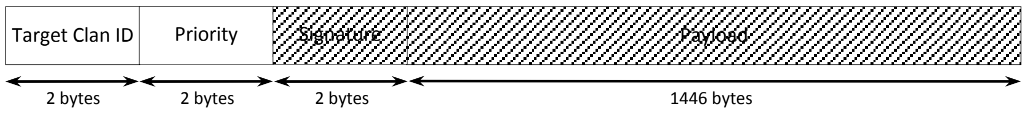

Figure 1: Message Format

CHAPTER V

SENDER ANONYMITY

5.1 Chapter Overview

The principal idea for the sender anonymity module of the proposed system is that, if nodes obfuscate their outgoing traffic in such a way that an adversary is not able to identify whether a node itself is the initial sender or the packet actually originated from some other node in the network, then the adversary will not be able to accurately identify the initial sender. In other words, the protocol requires all nodes in a clan (for a definition see Section 4.2) to voluntarily relay the other nodes' traffic randomly. As a result, the initial sender node would not be identifiable by the adversary.

5.2 Message Construction

The first step to start a communication is to create the message. Message construction takes place at the sender node and requires the sender node to have the IP address, the public key, and the clan ID of the ultimate receiver node. The message is then constructed by assigning the receiver's clan ID to the *Target Clan ID* field, assigning the encrypted message body (padded to fill up the allocated space) to the *Payload* field, assigning a random value between 1 and 65,535 (as described in more details below in Section 5.3) to the *Priority* field, and assigning the message signature (as described in Chapter VI) to the *Signature* field.

5.3 Unpredictable Aging Algorithm

An important aspect of the proposed system is the randomized method of routing packets (as discussed in Section 5.4). Because of this random process, it is theoretically possible for a packet to fall in a loop and never reach its destination. Therefore, a method of aging has to be deployed to prevent the indefinite postponement of the routing of the messages. For this purpose, a priority value is added to each message to represent how urgently a message has to be routed compared to other messages in the waiting list.

Calculating the priority of a message is one of the crucial steps in hiding the identity of the initial sender. This is due to the fact that if the first node in a communication path after the original sender of a message (i.e., the first node in the sequence of nodes that are used to relay the message) is an agent of the adversary, it would be possible for that node to identify the original sender by analyzing the priority assigned to the message, unless an unpredictable sequence of numbers is used to assign priorities.

In other words, when aging is used to prevent indefinite postponement, as a message goes through the first node after the original sender, it might be possible to determine the source of the message by analyzing the priority assigned to the message. Thus, in the proposed system, an aging algorithm was introduced to address this issue. To initialize the priority of a message, several methods can be used. One method would be assigning a specific number as the initial priority for all messages. In this case, if a message is sent to the next node before going through the aging process even once, it would be detectable by the first node after the sender. Another method would be assigning a random number as a message's initial priority. Now, unless the aging algorithm generates a virtually untraceable sequence of numbers, it would be easy to spot the sender in cases that either the maximum or the minimum random value is chosen as the initial priority of a message.

It is proposed that a suitable approach to tackle this issue is by designing an aging algorithm that produces a virtually untraceable sequence of numbers. In other words, given a priority value, one must not be able to confidently determine the previous priority value before the last aging iteration. It also goes without saying that, in order for the priorities to have meaningful values that can be used to prioritize the messages, the resulting sequence of number has to be convergent to a certain number.

Furthermore, even if the priority is initialized to a random number and the aging algorithm only predictably and intuitively decreases/increases the priority value, then there must be a maximum/minimum number, which in case this maximum/minimum value is used by the original sender, again the first node in the path would be able to identify the original sender. Therefore, the priority value not only has to be initialized to a random number, but also the aging process must at some points increase this value and at other points decrease this value, keeping in mind that, in order to prevent indefinite postponement, the priority values have to converge.

For this purpose, the Collatz Conjecture was used to generate untraceable iteration of numbers [Lagarias 85] using the 3x+1 function as defined below:

$$T : \mathrm{N} \to \mathrm{N}$$

$$T(n) = \begin{cases} (3T(n-1)+1)/2 & if\ T(n-1) \equiv 1 \pmod 2 \\ T(n-1)/2 & if\ T(n-1) \equiv 0 \pmod 2 \end{cases}$$

It has been demonstrated that the numbers generated by the 3x+1 function in the range of 2 bytes, which is the range under consideration for the priority field of the proposed message format, always converge to 1 [Leavens and Vermeulen 92]. In other words, the recurrence relation above produces sequences of numbers in such a way that a sequence always ends in 1. Furthermore, it is not possible to confidently determine the value of a_{n-1} for all values a_n (e.g., assuming that $a_n = 20$, then a_{n-1} can be either 40 or 13). This property is illustrated using the Collatz graph [Lagarias 85] in Figure 2 below.

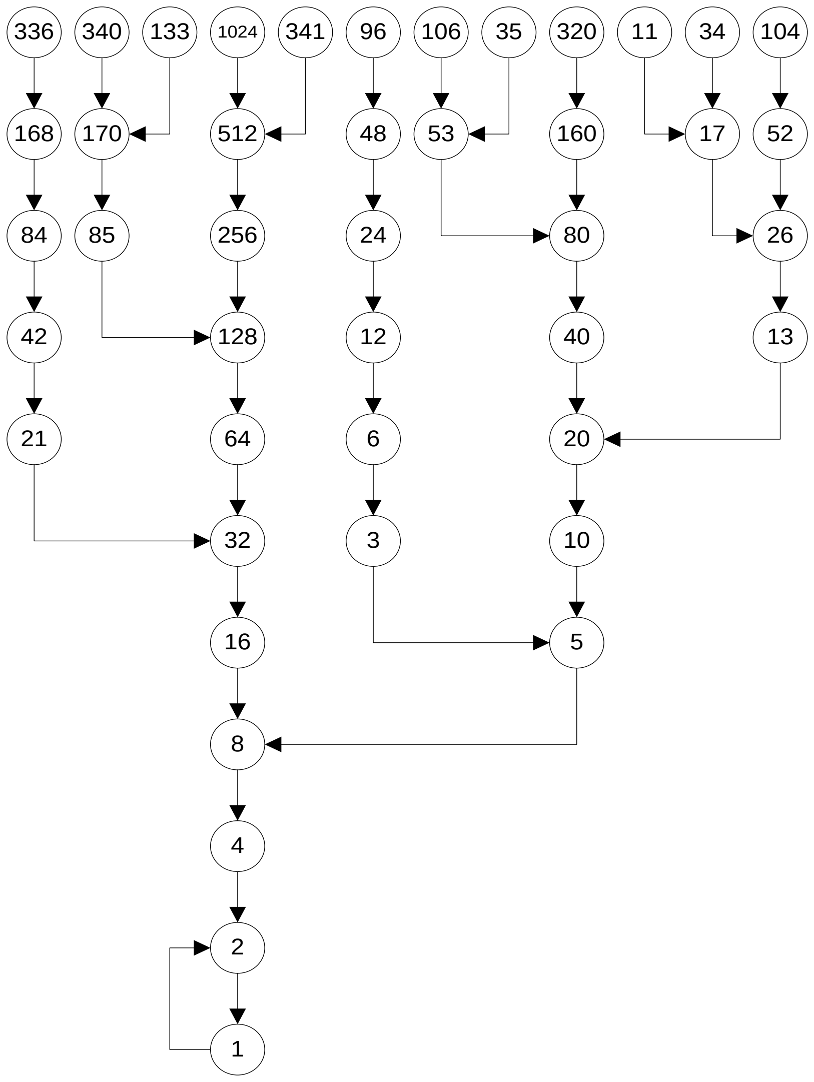

Figure 2: Colltaz Graph

As shown above, even though function T(n) is a convergent function, it is not monotonic. This property is suitable for anonymous aging because, as was mentioned earlier, the combination of a monotonic function and a fixed maximum/minimum value may result in a predictable priority value which can be used by the adversary to identify the original sender node. Thus, the following aging mechanism is proposed using the Colltaz's conjecture:

Let $P^{(n)}$ represent the priority of a message after *n* iterations. Therefore, $P^{(0)}$ will be the initial value assigned as the priority of a message.

$$P^{(0)} = 1 + (rand \quad \mathrm{mod} \quad 65535) \qquad [1]$$

Now having $P^{(n-1)}$, the value of $P^{(n)}$ is calculated using one of the following equations.

If $P^{(n-1)}$ is even:

$$P^{(n)} = P^{(n-1)}/2 \qquad [2]$$

If $P^{(n-1)}$ is odd and $(3P^{(n-1)} + 1)/2 < 65535$:

$$P^{(n)} = (3P^{(n-1)} + 1)/2 \qquad [3]$$

If $P^{(n-1)}$ is odd and $(3P^{(n-1)} + 1)/2 \geq 65535$:

$$P^{(n)} = 1 \qquad [4]$$

Equation [4] is introduced to prevent an *overflow* error when the generated number does not fit in 2 bytes and also to avoid possible loops in the sequence of numbers.

To sum up, at the time of creation of a message, a random number is used to assign a priority to it as in Equation [1] and subsequently at each iteration a new value for the priority is calculated using one of the equations [2], [3], or [4].

5.4 Routing Algorithm

The routing algorithm is the most important component of the sender anonymity part of the proposed system. The routing algorithm is in fact responsible for transferring messages from a sender to the receiver's clan while making every attempt to hide the identity of the sender node. The rest of the process i.e., after the routing algorithm delivers the message to the receiver's clan, which is to transfer the message to the ultimate receiver, is the job of the receiver module that is discussed in Chapter VI.

In the proposed system, each node in the network maintains a priority queue (called Outbox) of the outgoing messages, with the message priorities being stored in the priority fields of the messages, with the higher values signifying a lower priority. So the priority value of 0 represents the highest priority and the priority value of 65,535 represents the lowest priority.

The Outbox priority queue is used to hold the outgoing messages. Thus, in order to send a message, a sender node has to construct a message object and insert it into the local Outbox. After this point, the routing algorithm will take the message and route it throughout the network until it reaches its destination clan.

The routing algorithm was implemented as a separate thread (or process), called the engine, which has to be running on all nodes in the network at all times. The routing algorithm is based on the idea that each node has to help the other nodes in the same clan to hide their identity from the global adversary. This is done in a randomized fashion in such a way that it is not easily detectable which node in a clan is the original sender of a message. Thus, it could be argued that the closest that the adversary can get to identifying the sender is finding the clan of which the original sender is a member.

The proposed routing algorithm is comprised of two distinct actions. One action is picking the message with highest priority in the Outbox (the head of the queue) and sending it to the interface node of the receiver's clan as specified in the message's Target field. The other action is fetching a message from a random node in the sender's clan and inserting that message in the local Outbox queue. The routing algorithm decides which one of the above-mentioned actions to take based on the *Anonymity Level Indicator* (*p*). This number represents the probability of choosing the first action over the second action. For example, let $p = 0.35$ then 35% of the time the first action is taken and 65% of the time the second action is taken. In other words, in this clan, 35% of the time a node would send a message to the destination, and 65% of the time a node would help relay the other nodes' traffic.

Note that the anonymity level indicator has to be $0 < p < 1$, otherwise, either no messages will be sent to their destinations or all messages will be sent to their destinations directly from the original sender. Thus, it could be argued that the lower the value of p, the higher the anonymity provided by the system. Put differently, the higher the number of times that a node decides to forward a message to the ultimate receiver, the lower the level of anonymity would be that the system provides.

The routing algorithm can be best described in pseudo code as follows.

```
Randomized-Anonymous-Routing

while (true) {
    r ← rand() % 100
    if (r ≤ p*100) {
        # Send Local Traffic
        message ← Outbox.head
        receiver ← Interface Node of the target clan
        send message to the receiver
    } else {
        # Relay Other Traffic
        pick a node in the clan
        message ← remote Outbox.head
        push message to local Outbox
    }
    Age-Queue()
    Flush-Queue()
}
```

Age-Queue

```
# Update the Priorities of All Messages
for i ← 0 to Outbox.size {
    message ← Outbox.head
    pop the head of the Outbox
    pr ← message.priority
    if (pr is even) {
        pr = pr/2
    } else if ((3pr + 1)/2 < 65535) {
        pr = (3pr + 1)/2
    } else {
        pr = 1
    }
    message.priority ← pr
    push message to local Outbox
}
```

Flush-Queue

```
# Immediately Forward Messages with Priority 1
do {
    message ← Outbox.head
    if (message.priority = 1) {
        receiver ← Interface Node of the target clan
        send message to the receiver
        pop the head of the Outbox
    }
} while (message.priority = 1)
```

It is worthy of mention that the communications between two nodes in the same clan is exactly the same as communications between two nodes in two different clans, except that in this case, a message would always go through at least one relay node, which plays the role of the interface node for the target clan.

In summary, to initiate a message, the sender node simply creates the message and pushes it into the local Outbox. And, to route the messages in the Outbox, each node has to randomly decide whether to send the highest priority message to the respective receiver’s clan or to request a new message from another randomly selected node to be relayed by that node. Finally, all messages with the highest possible priority are forwarded to their respective receiver’s clans, hence no node would wander around indefinitely. It must be stated that the unpredictable nature of the proposed aging algorithm may result in having several nodes with the highest possible priority at the same time in a node’s local Outbox priority queue. Thus, in order to prevent starvation, it is important to immediately forward all these messages to their respective receiver’s clans.

CHAPTER VI

RECEIVER ANONYMITY

6.1 Chapter Overview

The issue of receiver anonymity is a more challenging problem compared to the issue of sender anonymity. This is singly due to the fact that every packet in the network has to somehow arrive at a receiver node. This cannot be obfuscated as easily as sending packets. Also, perhaps for this very reason, not much work has been reported in the literature on this issue, and in fact most of what is reported is based on a single idea of multicasting messages instead of transferring them right to the ultimate receiver [Waters et al. 03].

The principal idea for the receiver anonymity module of the proposed system is as Follows. The traffic is routed to several nodes instead of just to the ultimate receiver node, only the intended receiver is able to find out whether a message was targeted at that node or another node, and is able to actually decrypt the message and access the un-encrypted contents of the message. So, a group of potential receivers may be held responsible for any given message, thus achieving receiver anonymity. In this chapter, the proposed design's receiver anonymity is discussed based on the above scheme.

6.2 Message Signature

As discussed in Chapter IV, each message in the proposed system carries an extra two bytes of *Signature*. This value is used by the nodes to decide whether or not a message is meant for them. Note that the messages do not contain any information regarding the actual IP address or anything else that would reveal the actual identity of the receiver node.

The message signature is a 16-bit value that is generated at the sender node. Then the signature is attached to the beginning of the actual message body and is encrypted by the receiver's public key. Once a node receives a message, it has to decrypt the message body using its own private key. This is when the signature value can be used to tell whether or not the message was actually meant for a specific node.

The signature value is comprised of two separate bytes. The first byte (the most significant 8 bits) is called SIG0 and the second byte (the least significant 8 bits) is called SIG1. SIG0 is calculated by XORing the first 8 bits of the message body with 8 ones (11111111), and SIG1 is calculate by XORing SIG0 with the second 8 bits of the message body. Let's consider both the message body and the signature as a string of zeros and ones called MSG and SIG. Now the following formula generates the message signature for message MSG:

$$\begin{array}{ll} SIG[0] = MSG[0] \oplus 1 & SIG[4] = MSG[4] \oplus 1 \\ SIG[1] = MSG[1] \oplus 1 & SIG[5] = MSG[5] \oplus 1 \\ SIG[2] = MSG[2] \oplus 1 & SIG[6] = MSG[6] \oplus 1 \\ SIG[3] = MSG[3] \oplus 1 & SIG[7] = MSG[7] \oplus 1 \end{array}$$

$$SIG[8] = MSG[8] \oplus SIG[0] \qquad SIG[12] = MSG[12] \oplus SIG[4]$$
$$SIG[9] = MSG[9] \oplus SIG[1] \qquad SIG[13] = MSG[13] \oplus SIG[5]$$
$$SIG[10] = MSG[10] \oplus SIG[2] \qquad SIG[14] = MSG[14] \oplus SIG[6]$$
$$SIG[11] = MSG[11] \oplus SIG[3] \qquad SIG[15] = MSG[15] \oplus SIG[7]$$

When the message signature is ready, it is attached to the message body and the entire message is then encrypted by the receiver’s public key. Now, once a node receives a message, it has to decrypt the message body using the local private key and do the reverse operation to check whether or not the message was meant for that node. This process is discussed in details in the next section.

6.3 Routing Algorithm

As described in Section 5.4, messages are routed from one of the nodes in the sender’s clan to one of the nodes in the receiver’s clan called the interface node. Once the interface node receives a message, it is responsible for delivering it to the ultimate receiver. But, this is not a straightforward task, because the only information that any node at this point has regarding this message is that the ultimate receiver node is one node among all the nodes in the receiver’s clan. Even the ultimate receiver node itself is not aware of the final piece of the puzzle until it decrypts the first four bytes of the message body. The first four bytes contain the message signature and the first two bytes of the message body that have been used by the sender node to calculate the message signature.

The proposed routing is not a complete and sophisticated algorithm. Once the interface node of the receiver’s clan receives a message, it has to extract the target clan ID and the priority from the message, and forward the message to all the nodes in the clan. Note that the list of nodes in a clan is already provided to each node in each node's CMC. In other words, once a message enters a clan, it is broadcasted among all the nodes in that clan. Now all nodes have a copy of the message, but the message is intended for only one of them.

The process of identifying the receiver of a message takes place at each node at the time of message arrival. Once the message is completely transferred to a node, the node has to use its own private key and decrypt at least the first four bytes of the message. Let's call the binary string consisting of the first two decrypted bytes of the message SIG, and the second two bytes of the message MSG. A node in the receiver's clan uses the following formula to check if it is in fact the ultimate receiver.

If the first byte of SIG equals to the second byte of SIG XOR the second byte of MSG, and the result of XORing the first byte of MSG with the first byte of SIG equals eight ones, then the message has indeed arrived at the intended receiver, otherwise the node can simply drop the message.

For example, suppose the first two bytes of the message body is:

```
MSG := 11001100 10101100
```

Thus, the message signature would be:

```
SIG0 = 11001100 XOR 11111111 = 00110011

SIG1 = SIG0 XOR 10101100 = 10011111
```

Therefore:

```
SIG = 00110011 10011111
```

Now if a node receives a message the first four bytes of which after decrypting look like `00110011 10011111 11001100 10101100`, then that node is the intended ultimate receiver, because:

```
BYTE1 = BYTE2 XOR BYTE4
```

and

```
BYTE1 = BYTE3 XOR 11111111
```

If this condition does not hold for a node, then the message is simply dropped.

CHAPTER VII

SIMULATION RESULTS

7.1 Chapter Overview

In order to evaluate and test the proposed design, a simulation program was implemented. In this program, all clients maintain a thread-safe queue as the Outbox and also an *Engine* program which is the actual implementation of the routing algorithm. The engine runs on a separate thread and is responsible for routing the messages in the local Outbox. Furthermore, a central module responsible for initializing clients, assigning parameters, and collecting and reporting statistics was implemented. In this chapter, the statistics generated from running the implemented simulation program are discussed.

7.2 Sender Anonymity Simulation

As mentioned previously, the receiver anonymity module follows an uncomplicated and straightforward routing algorithm (a variant of multicasting) that would not require simulation to demonstrate its workability. Hence, this chapter mostly focuses on the sender anonymity module and routing algorithm.

In this experiment, the simulation program generated 100,000 messages for each of the 99 different anonymity level indicator values (0.01 to 0.99), and used the proposed routing algorithm to route them from the sender node to the receiver node. The graph below depicts the average number of nodes traversed by a message before arriving at the receiver clan in four different situations characterized by clan sizes of 5, 10, 20, and 50 nodes.

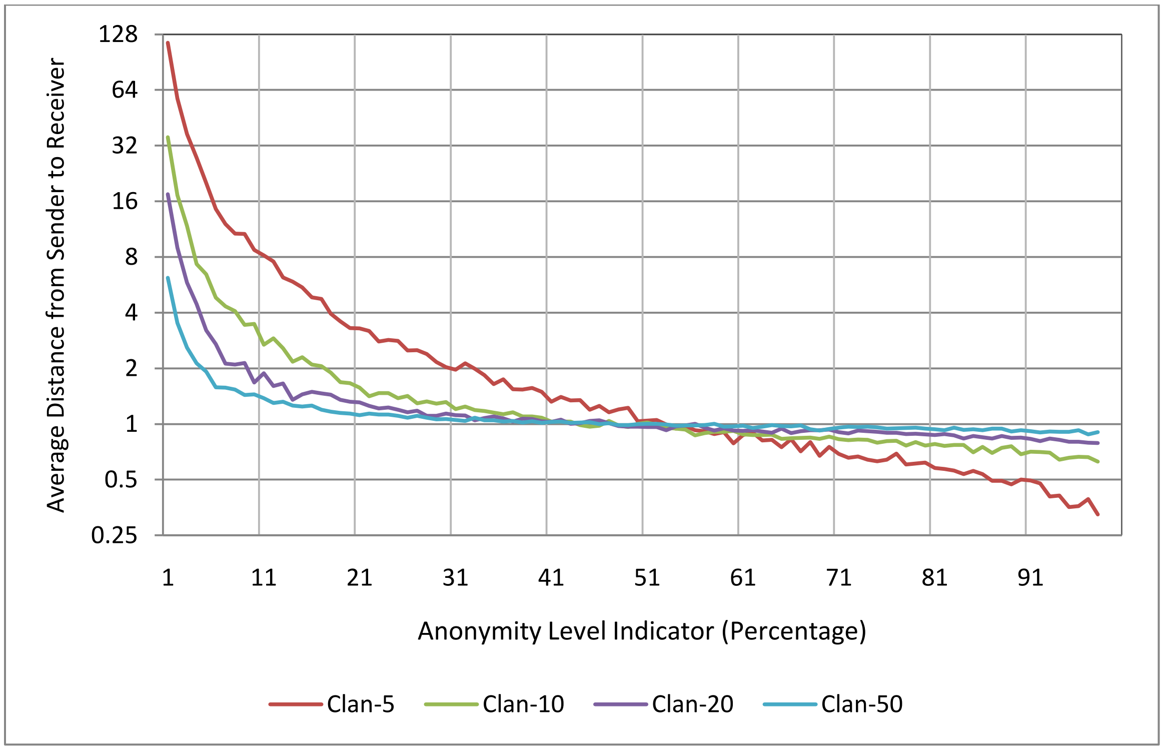

Figure 3: Average Sender/Receiver Distance vs. Anonymity Level Indicator

As the graph shows, the lower the anonymity level indicator (for a definition see Section 5.4), the higher the number of nodes a message would traverse before getting to the receiver. This is a direct result of the fact that, when the anonymity level indicator is low, in line with the design of the proposed system, most of the nodes in the system would prefer to pass the local traffic to the other nodes instead of forwarding the traffic to the receiver node.

The above graph also suggests that for low anonymity level indicators (below 50%), the greater the number of nodes in a clan, the lower the distance traveled by the message. This can be justified by the observation that, when the number of nodes in a clan is increased and there is not much traffic in the network, messages stay longer in each Outbox. In other words, while monitoring a single message, when the number of nodes in a clan increases, that particular message would stay longer at each Outbox in the path from the sender to the receiver, hence the higher chance of being forwarded to the receiver as opposed to being sent to another node in the clan.

In another experiment, 100,000 packets were sent from a single node to some random receiver (in a different clan) and the number of communications between each node in the clan with nodes outside of that clan was calculated. This data can be used to determine whether the adversary is capable of detecting a pattern in the network from the number of communications in a clan, perhaps resulting in identifying the sender node. The graph below shows the percentage of communications between each node in a clan of size 5 with nodes outside the clan, keeping in mind that all traffic is originated from a single node. The ratio of total communication is the calculated by dividing the number of communications between each node with other nodes outside of that clan by the number of communications between two nodes in one clan.

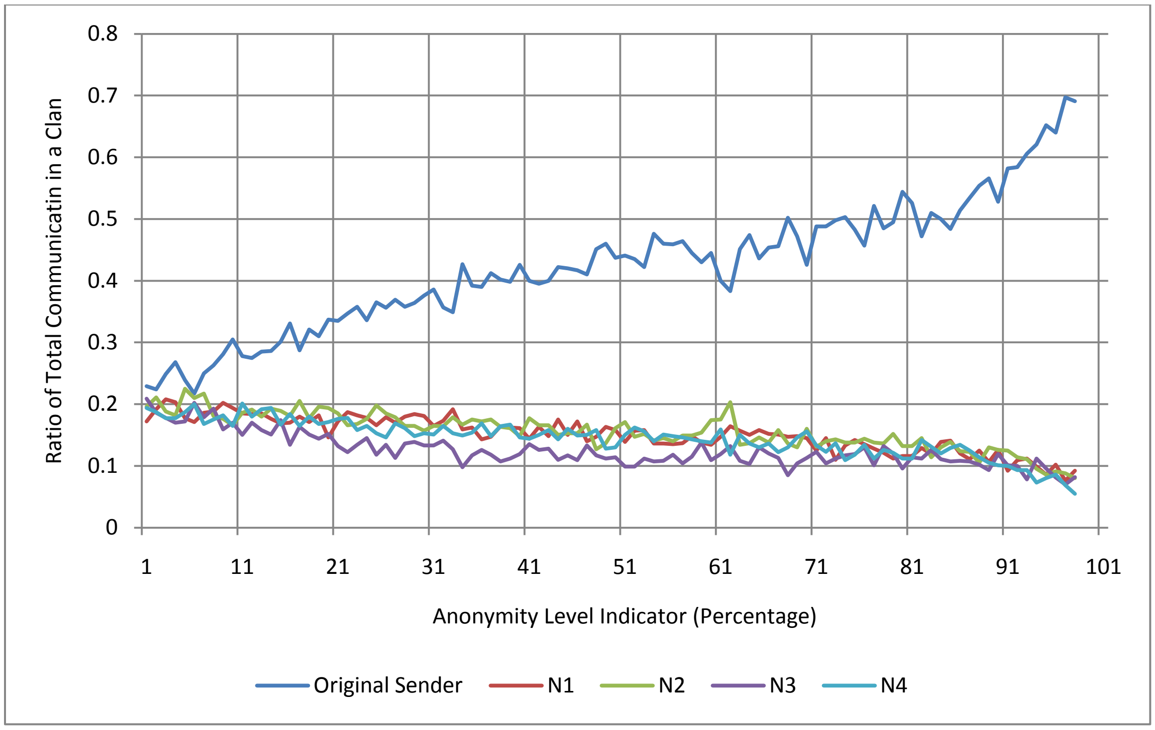

Figure 4: Ratio of Total Communication in a Clan-5 vs. Anonymity Level Indicator

It can be inferred from the above graph that, when the anonymity level indicator is increased, when only one node in the network is the originator of all packets in the network, it is more likely for the adversary to identify the sender node using traffic analysis. Thus, for the system to work as expected, it is best to adjust the anonymity level indicator according to the behavior of all the nodes in the network. In other words, when there are a few particular nodes in the network that initiate most of the traffic, the anonymity level indicator must be reduced to increase the level of anonymity provided by the system. On the other hand, in a situation where all nodes are equally contributing to the network traffic, the anonymity level indicator can be increased without the fear of revealing some patterns to the adversary.

CHAPTER VIII

SUMMARY AND FUTURE WORK

8.1 Summary

In this thesis work, a new network protocol was introduced that provides enhanced communication unobservability in the presence of active global adversaries. The proposed system is comprised of two modules, namely the *Sender Anonymity* module and the *Receiver Anonymity* module. The combination of these two provides communication anonymity. Moreover, random noise was added to the system to provide some level of unobservability. It was also argued that, in order to hide any sort of pattern that would otherwise be revealed to an eavesdropper, an unpredictable aging algorithm must be used to assign priorities to messages. This aging algorithm was designed using the Collatz conjecture and was introduced in Chapter V.

Furthermore, a new network architecture based on groups of collaborative nodes called Clans was designed. The point was also advanced that, if all the nodes in a clan try to obfuscate the outgoing traffic from that clan, then from the adversary's perspective, it is not identifiable which node in that clan is in fact the actual sender node.

Moreover, a receiver anonymity module based on multicasting messages to all nodes in a clan, was designed and added to the system. When the receiver anonymity module is combined with the sender anonymity module, the system as a whole provides enhanced communication anonymity. Also, a mechanism for hiding the identity of the ultimate receiver node in the message body was introduced.

Finally, a simulation program to evaluate and test the proposed design was implemented and a couple of plots depicting the data collected from running the simulation along with a detailed discussion of each of the graphs were presented.

8.2 Future Work

A more comprehensive simulation of the system can be designed and implemented, for example, the receiver anonymity module can be simulated. Also, investigating the optimum maximum and minimum clan size as well as a mechanism for merging clans may improve the system’s performance. Moreover, improvements on the receiver anonymity module to design a new algorithm that would not broadcast all messages, yet does not reveal any pattern to the adversary, can be invaluable.

REFERENCES

[Berthold et al. 2001] Oliver Berthold, Hannes Federrath, and Stefan Köpsell, "Web MIXes: A System for Anonymous and Unobservable Internet Access", *Proceedings of the International Workshop on Designing Enhancing Technologies: Design Issues in Anonymity and Unobservability*, pp. 115-129, Berkeley, CA, July 2001.

[Borisov 05] Nikita Borisov, *Anonymous Routing in Structured Peer-to-Peer Overlays*, Ph.D. Thesis, Computer Science Division, Electrical Engineering and Computer Science Department, University of California, Berkeley, CA, Spring 2005.

[Boyan 97] Justin A. Boyan, "The Anonymizer: Protecting User Privacy on the Web", *Computer-Mediated Communication Magazine*, Vol. 4, No. 9, pp. 7-13, September 1997.

[Chaum 81] David Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", *Communications of the ACM*, Vol. 24, No. 2, pp. 84-88, February 1981.

[Clarke et al. 00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hang, "Freenet: A Distributed Anonymous Information Storage and Retrieval System", *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pp. 46-66, Berkeley, CA, June 2000.

[Danezis et al. 03] George Danezis, Roger Dingledine, and Nick Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol", *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pp. 2-15, Berkeley, CA, May 2003.

[Dingledine et al. 04] Roger Dingledine, Nick Mathewson, and Paul Syverson, "Tor: The Second-Generation Onion Router", *Proceedings of the 13th USENIX Security Symposium*, pp. 303-320, San Diego, CA, August 2004.

[Freedman and Morris 02] Michael J. Freedman and Robert Morris, "Tarzan: A Peer-to-Peer Anonymizing Network Layer", *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pp. 193-206, Washington DC, November 2002.

[Goldberg et al. 97] Ian Goldberg, David Wagner, and Eric Brewer, “Privacy-Enhancing Technologies for the Internet”, *Proceedings of the 42nd IEEE Spring COMPCON (Computer Conference)*, pp. 103-109, San Jose, CA, February 1997.

[Kelly 09] Douglas Kelly, *A Taxonomy for and Analysis of Anonymous Communications Networks*, Ph.D. Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, School of Engineering and Management, OH, March 2009.

[Lagarias 85] Jeffery C. Lagarias, “The 3x+1 Problem and Its Generalizations”, *The American Mathematical Monthly*, Vol. 92, No. 1, pp. 3-23, January 1985.

[Leavens and Vermeulen 92] Gary T. Leavens and Mike Vermeulen, “3x+1 Search Programs”, *Computers and Mathematics with Applications*, Vol. 24, No. 11, pp. 79-99, December 1992.

[Möller et al. 03] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman, “Mixmaster Protocol – Version 2”, *Internet Engineering Task Force Internet Draft*, http://www.abditum.com/mixmaster-spec.txt, date created: July 2003, date accessed: February 2010.

[Pfitzmann and Hansen 08] Andreas Pfitzmann and Marit Hansen, *Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management – A Consolidated Proposal for Terminology*, Draft, http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, Version 0.32, date created: February 2008, date accessed: February 2010.

[Postel and Reynolds 88] J. Postel and J. Reynolds, “A Standard for Transmission of IP Datagrams over IEEE 802 Networks”, RFC-1042, *Information Science Institution*, http://tools.ietf.org/html/rfc1042, date created: February 1988, date accessed: February 2010.

[Reed et al. 98] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag, “Anonymous Connections and Onion Routing”, *IEEE Journal of Selected Areas in Communications*, Vol. 16, No. 4, pp. 482-494, February 1998.

[Waters et al. 03] Brent R. Waters, Edward W. Felten, and Amit Sahai, “Receiver Anonymity via Incomparable Public Keys”, *Proceedings of the 2003 ACM Conference on Computer Communications Security*, pp. 112-121, Washington DC, October 2003.

[Zimmermann 95] Philip R. Zimmermann, *The Official PGP User’s Guide*, The MIT Press, Cambridge, MA, May 1995.

APPPENDICES

APPPENDIX A

GLOSSARY

Active Adversary	A visible adversary that is capable of altering messages traversing a network [Kelly 09].
Aging	The process of increasing the priority of messages while they are traversing through a network.
Anonymity	The state in which a specific subject is not identifiable within a set of subjects [Pfitzmann and Hansen 08].
Anonymity Level Indicator	A number representing the probability of choosing to forward a message to the target clan over relaying the traffic of the other nodes in the clan.
Bridge Node	A node that is used by a prospective node to establish connection to a network.
Clan-*p*	A group of a maximum of *p* collaborative nodes in which each node is connected to all other nodes in that clan.
Clan ID	A 2-bytes unique value used to identify clans. It is assigned to each clan by the parent clan.
Clan Metadata Catalog (CMC)	A set of properties unique to each clan, containing the clan ID and the IP addresses of all the nodes in that clan along with their public keys.
Communication Anonymity	The state in which a message cannot be linked to any sender–receiver pair and no message is linkable to a particular sender–receiver pair [Kelly 09].
Communication Unobservability	The state in which it is not detectable whether anything is sent out of a set of “could–be” senders to a set of “could–be” receivers [Kelly 09].

Global Adversary	An omnipresent adversary that has full access to the entire network of nodes and links [Kelly 09].
Hand-shaking	The process of introducing a new node to the network and transferring the required information to the prospective node.
Interface Node	A node in a clan that is used by another node from outside the clan to communicate with that clan.
Network Metadata Catalog (NMC)	A set of properties unique to the network, containing the maximum clan size, the list of all clans in the network, and the IP address of an interface node for each clan along with the public key of the interface nodes.
Outbox	A priority queue maintained by each node in the network that is used to store and sort messages based on their priorities (age).
Parent Clan	A clan that spawns a new clan after joining a new node that results in that clan having more nodes than the maximum allowed clan size.
Random Noise	Untargeted traffic which causes the network to appear busy at all times or most of the time (a.k.a. cover traffic or dummy traffic).
Salting	The process of adding random values to the contents of a message before encryption in order to harden against dictionary attacks.
Unobservability	The state in which an adversary is unable to observe anything regarding a communication, i.e., the adversary is unable to determine the very existence of a communication.

APPPENDIX B

SOURCE CODE

This appendix contains the complete source code of the simulation program.

```
/**
 * File: Main.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.istdout.anonsim.engine.Engine;

public class Main {

	private static Log log = LogFactory.getLog(Main.class);

	private Engine engine = null;

	public static void main(String[] args) {

		log.debug("*** PROGRAM EXECUTION BEGINS ***");

		Main app = new Main();
		app.init();
		app.run();
	}

	private void init() {
		engine = Engine.getInstance();
	}

	private void run() {
		engine.run();
	}
}
```

```
/**
 * File: Constants.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim.utils;

public class Constants {

        // path to the configuration file
        public static final String PATH_CONFIG_FILE =
                  "conf/anonsim.cfg.xml";

        // total number of clients in the network
        public static final String TAGS_NUMBER_OF_CLIENTS =
                  "number-of-clients";

        // maximum clan size allowed
        public static final String TAGS_FAMILY_SIZE =
                  "clan-size";

        // value of the anonymity level indicator
        public static final String TAGS_ANONYMITY_LEVEL =
                  "anonymity-level";

        // number of iterations the experiment has to be executed
        public static final String TAGS_ITERATIONS =
                  "iterations";

        // sleep time used to slow down the simulation so that
        // it is more understandable
        public static final String TAGS_SLEEP =
                  "sleep";

}
```

```java
/**
 * File: ConfigLoader.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim.utils;

import org.apache.commons.configuration.ConfigurationException;
import org.apache.commons.configuration.XMLConfiguration;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class ConfigLoader {

            // singleton instance of the class
        private static ConfigLoader instance = null;

        private Log log = LogFactory.getLog(ConfigLoader.class);

        private XMLConfiguration config = null;

        private ConfigLoader() {
                config = new XMLConfiguration();

                log.debug("Loading configuration file at: "
                        + Constants.PATH_CONFIG_FILE);

                try {
                        config.load(Constants.PATH_CONFIG_FILE);
                } catch (ConfigurationException e) {
                        log.fatal("Unable to load configuration file: "
                                    + Constants.PATH_CONFIG_FILE);
                        System.exit(-1);
                }
        }

        public static ConfigLoader getInstance() {
                if (instance == null) {
                        instance = new ConfigLoader();
                }
                return instance;
        }

        public XMLConfiguration getConfig() {
                return config;
        }

        public String getRequiredString(String key) {
                String res = null;
                try {
                        res = config.getString(key);
                } catch (Exception e) {
                        log.warn("Unable to load configuration for: "
                              + key);
                } finally {
                        if (res == null) {
                                log.fatal(
                              "Missing required configuration key: \""
                                + key + "\"");
                                System.exit(-1);
                        }
                }
                return res;
        }

        public String[] getRequiredStringArray(String key) {
                String[] res = null;
                try {
                        res = config.getStringArray(key);
                } catch (Exception e) {
                        log.warn("Unable to load configuration for: "
                               + key);
                } finally {
                        if (res == null || res.length == 0) {
                                log.fatal(
                              "Missing required configuration key: \""
                                          + key + "\"");
                                System.exit(-1);
                        }
                }
                return res;
        }
}
```

```java
/**
 * File: Message.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim.client;

public class Message {

	private String body = "";
	private Integer receiverId = null;
	private int nodeCounter = 0;

	public String getMessageBody() {
		return body;
	}

	public void setMessageBody(String body) {
		this.body = body;
	}

	public Integer getReceiverId() {
		return receiverId;
	}

	public void setReceiverId(Integer receiverId) {
		this.receiverId = receiverId;
	}

	public void incNodeCounter() {
		nodeCounter++;
	}

	public int getNodeCounter() {
		return nodeCounter;
	}
}
```

```java
/**
 * File: Client.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim.client;

import java.util.ArrayList;
import java.util.Queue;
import java.util.Random;
import java.util.concurrent.ConcurrentLinkedQueue;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.istdout.anonsim.engine.Engine;
import org.istdout.anonsim.utils.ConfigLoader;
import org.istdout.anonsim.utils.Constants;

public class Client implements Runnable {

	private static Log log = LogFactory.getLog(Client.class);

	private Integer id = null;
	private Queue<Message> outbox =
		new ConcurrentLinkedQueue<Message>();
	private ArrayList<Client> clan =
		new ArrayList<Client>();
	private double anonind = 0.0;
	private int traffic = 0;

	private static final int ANONYMITY_LEVEL;

	static {
		ConfigLoader cfg = ConfigLoader.getInstance();
		ANONYMITY_LEVEL = Integer.parseInt(cfg.getRequiredString(
				Constants.TAGS_ANONYMITY_LEVEL));
	}

	private String prefix = null;

	@SuppressWarnings("unused")
	private Client() {
	}

	public double getAnonind() {
		return anonind;
	}

	public void setAnonind(double anonind) {
		this.anonind = anonind;
	}

	public Client(Integer id, double anonind) {
		this.id = id;
		this.anonind = anonind;
		prefix = "[" + getId() + "] ";
```

```
}

public Integer getId() {
	return id;
}

public void setClan(final ArrayList<Client> clan) {
	this.clan = clan;
}

public ArrayList<Client> getClan() {
	return clan;
}

public Message requestOutgoing() {
	synchronized (outbox) {
		if (! outbox.isEmpty()) {
			Message msg = outbox.poll();
			log.debug(prefix + "PASS");
			return msg;
		}
	}
	return null;
}

public void send(Message msg) {
	synchronized (outbox) {
		outbox.add(msg);
	}
}

public int getTraffic() {
	return traffic;
}

public void resetTraffic() {
	traffic = 0;
}

private boolean doISend() {
	int percent = ANONYMITY_LEVEL;
	Random rand = new Random();
	int num = rand.nextInt(100);
	if (num < percent) {
		return true;
	} else {
		return false;
	}
}

public void run() {
	Random rand = new Random();

	while (true) {
		boolean iSend = doISend();
		if (iSend) {
			synchronized (outbox) {
```

```
                    if (! outbox.isEmpty()) {
                        Message msg = outbox.poll();

                        Engine engine =
                            Engine.getInstance();
                        engine.reportSent(this, msg);

                        traffic++;
                    }
                }
            } else {
                int max = getClan().size();
                Client cl = null;
                do {
                    int index = rand.nextInt(max);
                    cl = getClan().get(index);
                } while (cl.getId() == this.getId());

                Message msg = cl.requestOutgoing();
                if (msg != null) {
                    synchronized (outbox) {
                        msg.incNodeCounter();
                        outbox.add(msg);
                    }
                }
            }
        }
    }
}
```

```java
/**
 * File: Engine.java
 * Author: Peyman Taher
 * Date: February 02, 2010
 */

package org.istdout.anonsim.engine;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Iterator;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.istdout.anonsim.client.Client;
import org.istdout.anonsim.client.Message;
import org.istdout.anonsim.utils.ConfigLoader;
import org.istdout.anonsim.utils.Constants;

public class Engine {

	private static final int MAX_CLAN_SIZE;
	private static final int ITERATIONS;

	static {
		ConfigLoader cfg = ConfigLoader.getInstance();

		// maximum number of nodes in each clan
		MAX_CLAN_SIZE = Integer.parseInt(cfg.getRequiredString(
				Constants.TAGS_FAMILY_SIZE));

		// iterations to calculate average distance
		ITERATIONS = Integer.parseInt(cfg.getRequiredString(
				Constants.TAGS_ITERATIONS));
	}

	// singleton instance of the class
	private static Engine instance = null;

	private final String body = "DUMMY_MESSAGE";

	private Log log = LogFactory.getLog(Engine.class);
	private ArrayList<Client> clients = new ArrayList<Client>();

	public static Engine getInstance() {
		if (instance == null) {
			instance = new Engine();
		}
		return instance;
	}

	// singleton class
	private Engine() {
		init();
	}

	private void init() {
        log.debug("Initializing engine...");

        log.debug("Populating " + MAX_CLAN_SIZE + " clients...");
        for (int i = 0; i < MAX_CLAN_SIZE; ++i) {
              Client client = new Client(i, 1);
              clients.add(client);
        }
        log.debug("Done with populating clients.");

        log.debug("Notifying clients of other nodes in the clan.");
        Iterator<Client> iter = clients.iterator();
        while (iter.hasNext()) {
              Client cl = iter.next();
              cl.setClan(clients);
        }
        log.debug("Done with notification.");

        log.debug("Starting clients...");
        iter = clients.iterator();
        while (iter.hasNext()) {
              Client c = iter.next();
              Thread t = new Thread(c);
              t.start();
        }
        log.debug("Done with starting clients.");
        log.debug("Done with initializing engine.");
  }

  public ArrayList<Client> getClients() {
        return clients;
  }

  private int totalDistance = 0;
  private boolean received = false;
  public void reportSent(Client sender, Message msg) {
        log.info("Reporting message sent:");
        log.info("Sender: " + sender.getId());
        log.info("Nodes traversed: " + msg.getNodeCounter());
        log.info("... and we're done.");
        totalDistance += msg.getNodeCounter();
        received = true;
  }

  public void run() {
        try {
              for (double p = 0.01; p <= 1.0; p += 0.01) {
                    Iterator<Client> iter = clients.iterator();
                    while (iter.hasNext()) {
                          iter.next().setAnonind(p);
                    }

                    for (int i = 0; i < ITERATIONS; ++i) {

                          Message out = new Message();
                          out.setReceiverId(0);
                          out.setMessageBody(body);
                    log.debug("Message: " + "|TARGET|"
                            + out.getMessageBody()
                            + "|");

                    Client sender = clients.get(0);
                    sender.send(out);
                    while (! received) {
                        Thread.sleep(1);
                    }
                    received = false;
                }

                double avgd =
                    (double) totalDistance / ITERATIONS;
                DecimalFormat df = new DecimalFormat("#.##");
                System.out.println(df.format(p)
                    + " \t\t\t\t" + avgd);

                iter = clients.iterator();
                while (iter.hasNext()) {
                    Client cl = iter.next();
                    System.out.print(cl.getTraffic() + "\t");
                    cl.resetTraffic();
                }
                System.out.println();

                totalDistance = 0;
            }

            System.exit(0);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```xml
<!--
      File: anonsim.cfg.xml
      Author: Peyman Taher
      Date: February 02, 2010
 -->

<anosim>

      <number-of-clients>150</number-of-clients>
      <clan-size>15</clan-size>
      <anonymity-level>50</anonymity-level>
      <iterations>1000</iterations>
      <sleep>1</sleep>

</anosim>
```

```
#
# File: log4j.properties
# Author: Peyman Taher
# Date: February 02, 2010
#

# the root logger's level is set to "error"
# change to lower levels to see more details
log4j.rootLogger=error, consoleApp

log4j.appender.consoleApp=org.apache.log4j.ConsoleAppender
log4j.appender.consoleApp.layout=org.apache.log4j.PatternLayout
log4j.appender.consoleApp.layout.ConversionPattern=%d %-5p [%F:%L] %m%n

log4j.appender.fileApp=org.apache.log4j.RollingFileAppender
log4j.appender.fileApp.File=anonsim.log
log4j.appender.fileApp.MaxFileSize=5MB
log4j.appender.fileApp.MaxBackupIndex=1
log4j.appender.fileApp.layout=org.apache.log4j.PatternLayout
log4j.appender.fileApp.layout.ConversionPattern=%-5p %d [%F:%L] %m%n
```

VITA

Peyman Taher

Candidate for the Degree of

Master of Science

Thesis: APPROACHING COMMUNICATION UNOBSERVABILITY THROUGH RANDOMIZED ROUTING AND UNPREDICTABLE AGING

Major Field: Computer Science

Education:

Completed the requirements for the Master of Science Degree in Computer Science at The Computer Science Department of Oklahoma State University, Stillwater, Oklahoma in July 2010.

Completed the requirements for the Bachelor of Science Degree in Computer Science at The Information Technology Department of Multimedia University, Cyberjaya, Selangor, Malaysia in 2008.

Experience:

Worked as a Software Engineer and Network Administrator in several companies during 2003-2005 in Iran and in 2008 in Malaysia.

Name: Peyman Taher Date of Degree: July 2010

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: APPROACHING COMMUNICATION UNOBSERVABILITY THROUGH RANDOMIZED ROUTING AND UNPREDICTABLE AGING

Pages in Study: 58 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: Providing anonymous communication on networks of interconnected computers is an active area of research which aims to enhance the privacy of the users of such networks. Communication unobservability is a stronger property compared to anonymity that attempts to guarantee that the adversaries will not even notice that a communication between a sender and a receiver is happening. In networks where an active global adversary may be present, it is assumed that at all times, all nodes in the network are potentially being monitored and also that at any time, any node should be treated as a possible adversary.

Findings and Conclusions: This thesis work introduces a set of requirements which, if followed in any system, provides enhanced anonymous communication. Furthermore, following these requirements, a new anonymous networking system was designed. This system provides communication unobservability in the presence of active global adversaries through randomized routing and unpredictable aging. Randomized routing is used to obfuscate the message routing pattern and the communication path, and unpredictable aging is used to prevent indefinite postponement of messages when routing randomly.

ADVISER’S APPROVAL: Dr. M. H. Samadzadeh