
# GossipChain: Network Information Discovery on DHTs with Gossiping and Trust Chaining

Master Thesis

**Dustin De Meglio**

25 January 2021

This work is submitted to the

**Chair of IT Security**

**Brandenburg University of Technology, Germany**

**Advisors:**
Dr. rer. nat. Torsten Ziemann
Asya Mitseva, M.Sc.

**Examiners:**
Prof. Dr.-Ing. Andriy Panchenko
Prof. Dr. rer. nat. habil. Klaus Meer

# Eidesstattliche Erklärung

Der Verfasser erklärt an Eides statt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

---

Ort, Datum Unterschrift des Verfassers

# Abstract

Network information discovery protocols are integral to enabling network function of any distributed network. One particular difficulty is the spread of trustworthy and unbiased information in an efficient and verifiable way. These problems are especially of interest in the realm of anonymous networking.

Previous works and networks, such as the centralized Tor and the decentralized Octopus DHT, have issues regarding scalability. In addition, although Tor has made efforts to route traffic in a bandwidth aware manner, there has been little focus on efficient bandwidth aware routing in fully decentralized solutions. This thesis shows how to extend and modify an existing solution, known as GuardedGossip, to distribute network information in an efficient, scalable, and bandwidth aware manner.

GossipChain presents novel verifiable *trust chains*, using previous works' ideas of witness list checks and bounds checking. Additionally, it incorporates a way of route selection which accounts for node bandwidth to allow routing in a bandwidth aware way. Through further use of the trust chains, it is shown that via new checks known as the *spot check* and *lineage check*, network members can actively lower their exposure to malicious nodes.

# Acknowledgements

Firstly, I would like to thank my professors at the Brandenburg University of Technology for supporting my education, and further Prof. Panchenko for allowing me to write this thesis at the chair of IT Security. Then I would like to give particular and special thanks to my advisors Asya Mitseva and Torsten Ziemann for their consistent encouragement, reviews, advice, and support in finishing this work during a tough year, and their instrumental role in helping to ask the right questions in refining and defining the approach outlined in this thesis. My friends have my gratitude for their consistent support, whether in the form of encouragement, or more direct discussions. Particularly, Neil Chiragdin for helping me with proofreading and grammar, and Adam Rinder and Tomi Jerenko for reviewing it for technical clarity. Lastly, Levent Afsar who endured an onslaught of practice presentations.

# Contents

# 1 Introduction

In the modern era of the Internet age, communication has been commodified. No longer simply a personal or private endeavor, open lines of communication have become easily monitored, the content therein representing both literal and figurative gold mines for corporations and governments. Current events have brought this knowledge to the fore in common consciousness, both from a political perspective [1, 2, 3, 4], and a private one [5, 6]. Whether for nefarious, virtuous, or even just financial purposes, ever increasing numbers of people believe their network communications are being watched [7].

As such, today, more people than ever are seeking out and using anonymization networks while browsing the Internet. Even more than the increased paranoia after various revelations of government and private surveillance of citizens' data [8, 9], anonymized communication is seen as a defense against hackers, and pursuant to a desire to protect and advance civil liberties. This also leaves out the numerous criminal enterprises who would seek to create and maintain these networks. Essentially, any person or entity that has a vested interest in protecting their identity online has a vested interest in the technologies which can hide that identity.

Any technology which claims to provide anonymization of the initiator must provide provable obfuscation at least of the source of Internet traffic, which demands that direct communication from source to destination not occur. Instead, traffic must be relayed over various "hops" from one node in a network to another, and eventually reach the destination, without an observer being sure beyond a reasonable doubt who initially sent the traffic.

Many different proposals exist to solve this dilemma, and there are provably-anonymous networks which may be able to prevent even ardent, dedicated, state-level actors, such as Mix-based systems [10] which are high latency, or centralized low-latency systems trusted for more immediate or even government communications needs, like Tor [11]. However, each of these networks involves significant tradeoffs either in robustness (for example, a resistance to censorship), protection against various adversarial attacks, or in communications delays. As

such, it is still an openly researched problem to have a provably anonymous network resistant to attacks and censorship, which is low latency and distributed.

The desire for anonymous distributed networks is not just for the sake of research, it will have significant real world impact. The current situation of relying on centralized anonymous network topologies for near real-time communication is fast becoming untenable. Tor's unprecedented growth as the preeminent anonymization proxy in the world, will eventually cause the network to waste most of its resources determining, sharing, and maintaining state, even with recent changes in the directory protocol specification [11, 12]. Other networks crafted with similar goals to Tor, like Tarzan [13], similarly have theoretical limits.

Furthermore, to the author's knowledge, the combination of significant bandwidth consideration with anonymization networks is an open problem. Therefore, this thesis introduces GossipChain a network information discovery solution which promises to enable communication with anonymity guarantees and bandwidth consideration at least as good as Tor [11], while improving upon previous solutions' abilities to prevent unintentional routing through malicious nodes, all while not requiring total knowledge of network state. This is done largely by modifying and extending the network information discovery protocol introduced by GuardedGossip [14]. The extensions allow for gossiped information to be trusted, and introduce new active and passive malicious actor protections through *trust chains*.

# 2 Background

Digital communication networks have grown to become an important part of daily life for many people, and this growth has also brought new security concerns, and the reality of government and commercial Internet surveillance [1, 2, 3, 4]. As such, there is increased demand for anonymity protections for individuals whose communications are carried by digital communications networks. This chapter introduces the current technologies underpinning anonymous and secure communications. These topics are generally required for understanding the state-of-the-art solutions available today, and will help to motivate this thesis.

## 2.1 Problem Description

In general, a node information discovery system's purpose is to enable nodes within the network to route to their desired destinations. Discovery systems which desire to provide anonymity protections must also adhere to the following restrictions on *initiator anonymity*. Meaning that under observation by a man in the middle any initiator or target are equally plausible for a given communication. This is done through the use of other network nodes as routers (also known as relays,) to prevent direct connections between an initiator and their destination. Additionally, these route intermediaries should be chosen in an unbiased fashion. The most popular anonymity network in the world, Tor [11], does this by creating trusted central authorities who act as arbiters for information on all relays. This presents a few problems which this thesis will try to address, namely enumeration of the entire network is trivial if total network state is public, allowing for blocking of nodes by a motivated adversary, also transmission of so much network state to many nodes is not scalable, and centralization creates attractive points for hackers or adversaries and act as bottlenecks and make denial of service trivial.

An intuitive answer to the issue of centralization is to use decentralization, however they are susceptible to various attack patterns and vectors which centralized systems are not. This section will briefly additionally introduce some of the more prevalent problems which can

affect decentralized networks. Later on, this thesis will analyze the introduced solution in consideration of these problems, in order to judge its suitability. So, in simple terms, this thesis answers the question; how do we protect initiator anonymity, in a decentralized and scalable way?

## 2.2 Onion Routing

Onion routing is a method of providing anonymity to online communications first outlined in 1998 by Reed, Syverson and Goldschlag [15]. It applies multiple layers of encryption (hence the name "onion") to secure message contents. The use of multiple layers of encryption also allows for hiding the initiator as well as the recipient against assumed dishonest participants with specific capabilities. In a simplified setup, an initiator of a communication would first choose a series of nodes, i.e., computers or servers providing routing services, to route through. Starting from the last node in the series, the initiator would then recursively encrypt the data with a different key for each intermediary node. These routers then relay traffic back and forth between the initiator and destination. In practice, initial communications are encrypted using asymmetric key pairs, and used to negotiate symmetric keys for the duration of the session. These symmetric keys are used for the encryption of the user data, as symmetric cryptography is more efficient for real time communications.

When communicating a message it is forwarded to the first node in the route, who decrypts the first layer, and forwards it on to the next node, each one peeling away the subsequent layer until the last one can finally read the message. The length of the route depends upon the specific implementation, but generally is three. Since its inception, it has formed the basis of many anonymity techniques and networks, such as Tor [11], Octopus DHT [16], and others.

## 2.3 Private Information Retrieval and Oblivious Transfer

The use of centralized databases/data repositories presents a unique problem regarding user privacy. An interested database owner can collate information about accesses and data retrieval of a given user. One solution to this problem is for the user to request a copy of the entire database at periodic intervals, and then only query from their local copy. However, this introduces severe overhead, and removes some of the convenience of publicly accessible

data, primarily that it allows near-instant access to the most updated, recent and verifiable data.

Private Information Retrieval (PIR) [17] was devised as a method for the user to retrieve either individual records or blocks from a database, securely and anonymously, directly from the remote database itself. Thus maintaining the benefits of a publically accessible database, and avoiding the drawbacks of having to maintain a local copy. PIR operates similarly in concept to oblivious transfer [18], that is both consider the case of a data holder $A$ who has some discrete pieces of information, one of which a user $U$ requires.

The key difference between oblivious transfer and PIR, is that an oblivious transfer is bidirectional, in that $A$ must remain oblivious to what $U$ has actually requested, and $U$ must remain oblivious to the other data $A$ can give. In PIR only the data holder must remain oblivious to what $U$ requests [17, 19]. Both methods have played roles in other solutions in the field of anonymous network information discovery. Notably in PIR-Tor [20], and ConsenSGX [21]. More information on these solutions is given in Chapter 3 which concerns related works.

There are two versions of PIR, *computational PIR* [19], leveraging problems known as hard to solve similar to the primitives involved in public-key cryptography, and *information-theoretical PIR* [17] which is based on provably (via number theory) unsolvable problems. While better, information-theoretically secure PIR is necessarily more complex both in mathematical and in physical (i.e., equipment and implementation) terms than computational PIR. For example, in practical terms, to implement information-theoretically secure PIR in a distributed way one needs at least two, and preferably more servers [17].

## 2.4 Distributed Hash Tables

A hash table is a mapping between keys and data. Typically, arbitrary data is hashed to form the key, to which some value is mapped (generally the data itself). A hash table provides a mechanism to find a value mapped to this key within a data structure. A distributed hash table implements this behavior over a network consisting of multiple hosts [22]. Since their first appearance in the 1990s, distributed hash tables have been a fairly well researched topic [23, 24, 25, 26]. When considered more simply DHTs chiefly provide a lookup service, and as such DHTs are capable of forming many different services, and fulfilling many use cases, like anonymous routing or a distributed datastore. Their distributed nature makes them

particularly scalable and fault tolerant, when considering macro-scale availability. However, as with most Peer-to-Peer (P2P) networks, they are susceptible to potential micro-scale bottlenecks among entrants in the network.

These bottlenecks can be abused, for example, consider a simple case where a node always sends the wrong value for a given key [24, 26]. The potential issues created by these bottlenecks has motivated numerous security-minded overlays to augment DHTs, a subset of which are discussed either in this chapter, or in Chapter 3. Described below are the two DHTs which form the basis of the DHT-based approaches discussed in the related works chapter. Chord is of particular importance as it also forms the underlying DHT used in this thesis.

### 2.4.1 Chord

Chord as outlined in 2001 by Stoica [24] is a DHT using a $m$ bit length identifier assigned via SHA-1 for consistent hashing. Chord nodes are arranged in a directed circular graph of at most $2^m$ nodes. The number of query hops is at most $O(\log_2(n))$, due to the use of consistent hashing and $m$-entry finger tables [24]. Due to its efficiency, it forms the base DHT of many later works ([16, 27, 28]).

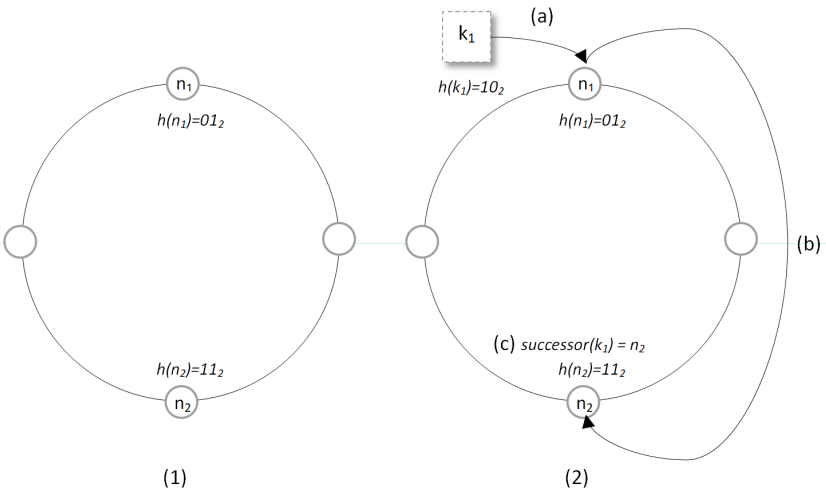


Figure 2.1: Example Chord Circle and New Key Insertion

Chord strives, and is largely successful, in attaining efficient load balancing and scalability, due to consistent hashing and uniform distribution. Consistent hashing is a specific subset

of hashing technique related to hash tables [29]. It guarantees that only a smaller subset of keys need to be remapped when a hash table is resized.

Geometrically Chord is shaped in a circle with directed edges between nodes. Each node in the network is placed in order of their identifier within the circle. The identifier is formed using a secure hash function (for example SHA-256 if implementing today), which provides a uniform distribution and a collision rate near zero, as well as the $m - bit$ identifer. The geometry provides the consistent hashing guarantees. This helps ensure that when someone joins or leaves the network, the resizing of the DHT, and the remapping of keys is done efficiently. According to the authors, this takes $O(m \log n)$ time making Chord reliably efficient [24]. This makes Chord a popular choice as an underlying Distributed Hash Table (DHT) for multiple networks and protocols [12, 16, 27, 28, 30].

On network join, each node in the network is placed in order of their identifier within the circle. To store data, a key is placed in custody of the first node within the circle whose identifier is equal to or greater than its identifier. That node is now known as the successor of key $k$, or $successor(k)$ [24], this is illustrated in Figure 2.1. Two important concepts to the Chord DHT, when considering the future security-minded overlays, are the neighbor and the fingers. Neighbors are those nodes directly successive to a node, whereas the $i^{th}$ finger of a node $n$ is defined as $successor(n + 2^{i-1})$. Nodes within Chord maintain a *finger table*, of size $m$. Therefore, fingers are nodes which are not direct neighbors, but of which a node still has direct knowledge. Through this special relationship, a node can route to its finger directly. Again, the entries in the finger table are equivalent to the number of bits in the identifier meaning a Chord DHT has the following properties if the identifier size $m = 32$, a network size of $2^{32}$ and a finger table length of 32 [24]. Furthermore, when referring to distance within the Chord circle, it is directly related to the difference in values of the identifiers.

Also of interest is the node joining (churn) and balancing phases of the network. The process for a node joining works as follows: a new node $n_2$ asks an existing node $n_1$ in the network to find its successor, similar to finding $successor(k)$ in Figure 2.1. $n_2$ begins the successor search function from its first finger. To reduce the total number of messages, it also checks for entries in its finger table which would contain no node, and functionally skips them, instead of a naive search through the whole table. The process is completely finished once the stabilization round is run. Stabilization is the process of maintaining finger tables of all nodes accounting for node joins and departures [24].

### 2.4.2 Kademlia

Kademlia uses $160-bit$ keys and an XOR operation, as opposed to a classic arithmetic operation, to provide an approximation of distance for its DHT [26]. Using XOR allows Kademlia to have efficient computations while maintaining all the necessary aspects of a distance function, that is symmetricity, identity, and the triangle inequality. Unlike Chord, where nodes can be thought of as being within a circle, for Kademlia they can be represented by a binary tree. Each node has some knowledge of nodes "near" it, meaning within the same subtree of the binary tree. Nodes searching for an identifier need only go to the closest possible identifier they are aware of, which then asks the closest they are aware of, and in this way the rest of the tree is crawled as necessary to find the requisite key. Kademlia is notable because it is fairly straightforward to analyze, as the XOR-metric forms an abelian group, and is therefore provably consistent and efficient. Like Chord, it also contacts at most $O(\log n)$ nodes per query [26].

In order to route to a destination node $D$, an initiator node $I$ sends a `FIND_NODE` request to the nearest ID, which represents node $N$, it is aware of. $N$ responds with $k$ nodes, depending on the replication factor, it knows closer in the tree. The replication factor is a configurable parameter defining how many nodes are returned during the lookup. $I$ chooses the closest node and continues until it reaches its destination [26].

## 2.5 Gossip Protocols

Humans have long been efficient at information dissemination, and so a class of algorithms which mimic the human method of oral information transmission were created and are collectively known as *gossip protocols*. A naive gossip algorithm involves one node $A$ telling another node $B$ new information. Whenever a node hears new information, it triggers it to spread that information further. Once all nodes have been informed, transmitted, and no longer receive new and updated information the algorithm ends. In this way, information can be disseminated without the need of a centralized authority [31].

Tarzan [13] is the most notable implementation of a gossip protocols for a secure and anonymous network, using a gossip protocol to ensure the transmission of network state from limited initial knowledge. It is a P2P anonymity network with onion-style routing using sequences of routers. All members of the network are potential routers, as well as potential initiators of communication, making it a system of equal peers. As equal peers, nodes in

the network must by design know all other nodes in the network when choosing the relays through which to route traffic, and to spread this information gossip protocol is utilized. Note that significant information can be gained about a user if it does not know the whole, or nearly the whole network, or use all peers equally to route traffic [13]. More specifically, the information leaks created could create a situation in which the initiator is probabilistically determinable just by the overuse of certain routes.

Ambitiously, Tarzan incorporates the concept of plausible deniability within the network itself. Natively providing a user the ability to cover their traffic, meaning using additional dummy data in order to obfuscate characteristics about about the intentional data transmission. Without the cover traffic, these characteristics might be identifiable to the user. Therefore, Tarzan incorporates random *cover data* which the user does not consciously create, with "real data" which is the user's actual traffic. This data is then interspersed together, creating deniability about which traffic is the meaningful traffic. This cover traffic is enabled by the use of *mimics*. To enable this behavior, each node first chooses a random value $k$. The given node then creates a bi-directional relationship mimic relationship with $2k$ other nodes.

Data is sent in a time-invariant manner between mimics in either direction over the wire. Data rates as well as packet sizes are as uniform as possible in both directions, to prevent an attacker from finding the originator of a message. As mimics are used to hide the actual use of the network, they must necessarily be one of the starting nodes for the creation of routes, otherwise differentiating between cover traffic and real traffic becomes trivial [13].

Tarzan is secure, however the requirement to know the whole network, the use of mimics and cover traffic, as well as the need to transmit the network state to prevent the information leaks outlined above introduces significant delays and overhead within the network. As the network grows the amount of traffic dedicated solely to maintaining state or cover traffic grows in tandem straining the network and limiting its total capacity [12]. This makes the use of Tarzan impractical at scale.

GuardedGossip [14] attempts to use a gossip algorithm in a secure and anonymous way, while mitigating the previously seen security tradeoffs or vulnerabilities demonstrated by other networks, specifically bandwidth wasted on transmission of network state, centralization, and robustness to malicious information. GuardedGossip incorporates two specific techniques to achieve this robustness: bounds checking, originally defined in NISAN [27], and witness list checking, which performs a similar function to redundancy introduced in other solutions [12, 13, 16]. Both of these concepts will be defined in more detail below. Nodes learned

about, via gossiping, are divided into two groups, which essentially boil down to verified and unverified. Verified nodes have passed both security checks, unverified have not.

The first of the checks is the aforementioned bounds checking. GuardedGossip is an overlay of a Chord DHT, and therefore inherits the concept of fingers. In simple terms, bounds checking is a measure of the plausibility of a finger table. To accomplish this, a node determines an approximation of the number of nodes in the DHT, $n$. $n$ is approximated by calculating the distance between the given identifiers in its finger table and what the optimal identifiers would be if the underlying DHT was full (i.e., $2^m$ nodes, as defined in the Chord section). This allows an approximate calculation of node density $d = 2^m/n$. NISAN [27] additionally calls for a tolerance factor, here $t; t > 0$. A received finger table is considered plausible if its node density $d' < td$. GuardedGossip defines its tolerance value to be $t = \sqrt{1/f}; f := 0.2$. The second check is the witness list check. The basic idea introduced here is to determine if a node has ever come into contact with a more correct finger for a node than it is reporting. To that end, if witness list checking is performed on a finger table from a node $B$, by a node $A$, then $A$ first calculates the ideal fingers of $B$, as if the DHT was fully populated. Next, it checks its witness list, which is a collection of node identifiers it has witnessed either during stabilization or gossiping, and determines if one of these nodes is a more correct finger for $B$ than what it is reporting its fingers to be. That is, does the witness list of $A$ contain a node with an identifier closer in distance to the ideal finger than what $B$ says? If so, then with a probability of $p = 1/2$ the finger table is discarded, or a liveness check is performed on the more correct finger, to see if $B$ was lying. By doing this probabilistically, doubt is cast on the state of information held by $A$ [14].

After these checks, verified nodes are called *guarded nodes* and can be used for routing as well as be further propagation. Each node maintains a *guarded list* of nodes having passed this verification. A further change from the naive gossiping algorithm is that gossip is actively sought, instead of passively obtained. Additionally, the gossip relationship is created in such a way that nodes only accept or transmit gossip to those within its finger table [14].

A complete overview is then: when a node wishes to receive gossiped information from another node it requests that node's whole finger table, on which it performs bounds checking. Nodes within that finger table are candidates for further use, either for gossiping or circuit creation. In either case, random nodes are chosen from these candidates, and the witness list check is performed on this subset, as outlined above, and if appropriate nodes are placed on the guarded list. Afterwards, circuit creation is done from a random node within the guarded node list, and further hops through to the destination are accessed via tunneling.

GuardedGossip was shown to be very successful with minimal overhead against both passive and active attacks, and proves the viability of using gossip protocols to enhance the security of DHT based anonymous communication methods [14]. The approach proposed in this thesis is based on this algorithm, using both bounds checking and witness list checking, and adding the concept of a trust chain or *lineage*.

## 2.6 Attacker Model

When defining whether a system is secure or not, it is important to understand the assumed environment in which it will operate. Typically, one can assume any variety of attackers who may interfere with a network, i.e., governments, hobbyist hackers, revenge seekers, etc. Each may seek to encourage false operation of the network for any number of reasons. As such, it is not considered possible to defend against all attackers with any potential capability at all times. This section lists the specific attacker model assumed for this thesis.

First, it helps to define a few terms related to attackers and how they operate. Of critical importance are *passive capabilities* and *active capabilities*. Passive capabilities mean an attacker has the ability to monitor some portion of the network without manipulating traffic specifically. For example, as the router for all traffic, assuming no other forms of encryption are used, an Internet service provider can view the Internet traffic of a specific user without significant effort. Active capabilities imply an attack who can manipulate traffic or perform directed injections, modifications, or deletions of data in order to attempt to influence network operation. Additionally, attackers can be *internal, external, local* or *global* which equate to inside or outside the network, and with a small or large view of network participants, respectively.

For this thesis, the attacker model is assumed to be similar to that of other related work [12, 14, 16, 27, 32, 33] in the field i.e. $f = 0.2$, which is a fairly aggressive fraction. This paper assumes an attacker with passive and active capabilities who is internal and local, and can act arbitrarily in relation to the protocol. That is, the attacker can create messages at whim, save data which otherwise would be ethereal, and generally act maliciously. However we do not assume a global attacker with the ability to have total oversight of entrance and exit within the network. It is not assumed the attacker has the ability to break the basic assumptions on which the system is built.

## 2.7 Common Attacks on Network Information Discovery Solutions

Next, a collection of the common attacks mentioned in this thesis against anonymity networks will be introduced and defined. These attacks are generally all performed by internal local adversaries, as described in Section 2.6. Understanding these attacks is required to understand the motivation behind the defenses of GossipChain, as well as the background work on which it is based, and the related work.

### 2.7.1 Range Estimation Attack

The range estimation attack [16, 34] is a method of breaking initiator anonymity via taking advantage of the directed nature of an underlying DHT like Chord. It works by setting "estimation bounds" as a circuit is created to at least have a probabilistic idea of a target node. That is, in the beginning the upper and lower bound of estimation are the preceding and successive nodes of the initiator. If a malicious node is queried later, it knows necessarily the target exists beyond its position in the DHT, and if there are other malicious nodes it controls which were not queried, it knows another bound. In the end, the attacker can determine a small subset of nodes which can be the target, partially (and in some cases [16] totally) breaking target anonymity [34].

### 2.7.2 Eclipse Attack

An eclipse attack [35, 36] involves forming a community of nodes which collude in order to break potential protections introduced into the network. This community then, in its most damaging incarnation, topologically surrounds honest nodes and feeds them false information in order to control future routing or surveil traffic. For a more concrete example of this attack in practice, see the section on ShadowWalker in Chapter 3, and the paper by Schuchard et al. [36].

### 2.7.3 Bridging and Fingerprinting Attacks

In [37, 38] Danezis et al. outline the concept of the fingerprinting and bridging attacks against anonymous networks. The basic idea is to leverage the limited knowledge of each node against it in order to break the initiator anonymity assumptions or protections. If a

node has only a limited view of the network from which it performs routing, its anonymity can be broken. Consider the following case: a node $A$ knows five total nodes, $N_i$ for $i \in [0, 4]$ and assume a network topology where $A$ is the only node who knows some subset of those nodes in combination, $N_1, N_2$ and $N_3$. If $A$ routes through these three nodes, it becomes trivial to determine $A$ as the initiator for a node with passive observation skills over one of the intermediate nodes [38]. Similarly, by using knowledge of network topology and the fact that nodes have limited total view of the network at any one time, and make assumptions from communication "relationships". This means, if $A$ communicates to $C$ through $B$, and an attacker has view of the link between $C$ and $B$, and knows only $A$ knows both $B$ and $C$, the attacker can assume the communication initiated from or at least contained $A$. [38].

### 2.7.4 Sybil Attack

The Sybil attack [39] outlined originally in 2002, occurs when an attacker can cheaply create nodes (usually virtually) within the network to encourage a large malicious fraction. If this happens, all other attacks listed in this section become easier. In general, all anonymity networks are susceptible to the Sybil attack, without outside mechanisms to control entrance into the network. As such, this thesis also does not attempt in and of itself to solve this problem.

### 2.7.5 Denial of Service

Although denial of service, meaning partial or total disruption of communication within the network, is an outcome rather than an attack, it is prevalent, and should be discussed. Typically, denial of service attacks are used to make the network not viable for real-time or low-latency communication. Therefore, care must be taken to ensure that a network's behavior does not allow, or minimizes the risk of this occurring.

# 3 Related Work

This section contains a review of the state of the art within the field. Approaches can be divided largely into two lanes: centralized and P2P. As will be seen, centralized solutions are based mostly around Tor [11]. The decentralized methods take many forms; some of which may incorporate some aspect of centralization, although considered here for our purposes as decentralized. Where there exists a central authority in a P2P network, it generally exists to provide or revoke certificates, as in [16].

Client-Server systems have the benefit of their trusted authorities to maintain and coordinate information which can or will be used for routing. However, these centralized mechanisms are attractive targets for attack. Decentralized (P2P) systems operate in a wholly untrusted environment, where any actor can be potentially malicious, even those generating or sharing routing information. While from a security and anonymity standpoint, this is challenging, P2P networks theoretically offer many benefits over their centralized counterparts in terms of scalability. Solving this tension has encouraged a lot of research into making P2P networks suitable for anonymity purposes, [12, 13, 16, 27, 40].

## 3.1 Client-Server-based Approaches

The most commonly used anonymous communication network with over two million users is Tor [11]. This network uses a limited number of nodes acting as relays (or routers) to assist users in forming circuits and enabling onion routing [11, 12]. A circuit is a series of nodes through which traffic is onion routed [11].

The network is formed by nodes fulfilling different specific roles to enable anonymous communication. In general, circuits are formed of three nodes, not counting the traffic initiator or destination. The first role is that of the *entry guard node*, which are computers which another uses to enter the Tor network. They are used statically for a period of around 2-3 months. The next node through which traffic is routed is the *middle relay* and lastly there is the *exit node* which forwards the traffic to its final destination. Collectively these

three are all known as *relay nodes* or *routers*. Finally, of critical importance are the *directory authorities* running the directory protocol, and who feed a network consensus to *directory caches*. The purpose and content of the consensus is described below.

The directory authorities run the directory protocol, and are small in number and unique among the Tor network. Considering a threat model of Tor, they occupy a space in which they are considered "semi-trusted". Other nodes in the network are considered untrusted. Directory authorities are also run and spread among multiple organizations to aid in network robustness, and to divide trust. They are trusted in that they have the ability to sign information on network state, which is the list of relay descriptors, which are the IP address, public key fingerprint, and other pertinent information about the relay; they also are arbiters of network state to directory caches.

In the initial specification of the Tor network, each directory authority reported on network state *as they saw it*. This was improved to a consensus model in version two, where a node in the network would request state from any/all directory nodes, and form their own consensus. While an improvement over the first version, it caused nodes to have variable views of the network status at any time. As of version three of the specification[1], shown in Figure 3.1, network state information is aggregated among all directory authorities and a consensus on network state is formed and agreed upon via majority vote periodically. These directory authorities are long-term in the network, and a list of them is shipped with the Tor source code [11].

Each node entering the network requires full knowledge of all currently available relays, which is requested from a directory cache. The directory cache then provides the consensus it received from the directory authorities which was signed with a tightly kept long term key. All subsequent requests for updates result in that entering node only receiving updates to outdated information, or new information it does not already have. This is an improvement from previous versions of the directory specification in which the whole consensus was forwarded [11].

However, even with these performance improvements, as the network grows, an increasing amount of traffic is dedicated to the transmission of this information. As this clogs network capacity, it presents a scalability problem [12]. The new directory specification was only a temporary fix. Additionally, this bottleneck can create a security problem, if a dedicated adversary is capable of gaining control over these dedicated relays. Furthermore, the necessary

[1] https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt

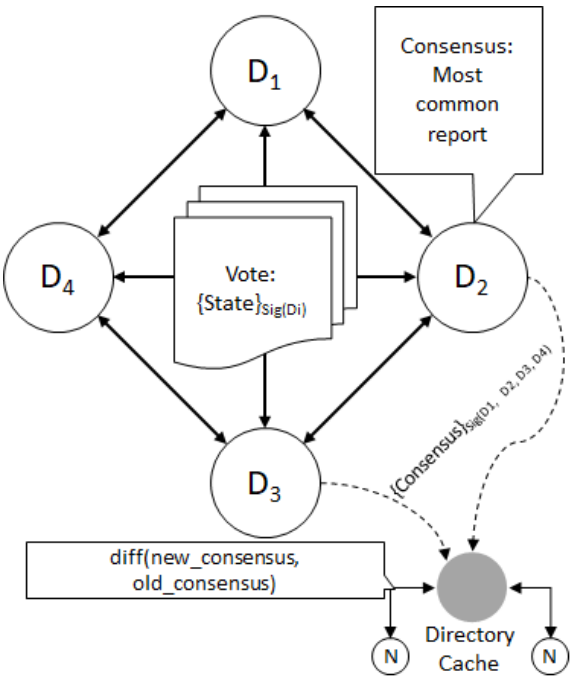


Figure 3.1: Tor dirspec-v3 overview

knowledge of all network state makes Tor not a viable option for censorship prevention, as when a motivated actor (say a network operator) joins the network, they can easily obtain a full network consensus, and use that information to block all routing nodes, should they have the capability.

Various attempts have been made to improve the shortcomings of Tor have been made, but greater attention will be given here to two state of the art methods, PIR-Tor [20], and ConsenSGX [21]. Both relying on different oblivious transfer techniques in order to improve or maintain the privacy of the system while allowing for lesser data transfer dedicated to network state material.

PIR is a method to retrieve a block of data, from most likely a database, without the owning server having knowledge of which data was requested. PIR-Tor by Mittal et al. [20] is a modification of Tor, where instead of sending whole encrypted databases to peers, it uses various forms of PIR in different portions of the Tor architecture. Namely, *information theoretic PIR* for guard relays; by which is meant guaranteed Privacy without regard for the computational power of the servers or machines involved the query, and *computational PIR* for directory caches which means using techniques thought to be reasonably secure, and

exceedingly hard to break in reasonable time.

Additionally, the ability to request specific records from the database measurably reduces the load of network state information transfer. The use of PIR also ensures minimal loss in user anonymity.

As in normal Tor, any relay can be a directory cache and still holds potentially global network view, and clients connect to these servers to form circuits. For the guard relays, meaning those first servers in the circuit chain, information theoretical security helps prevent potential collusion, and minimizes information leaks. And in the information theoretical PIR mode, all three guard nodes must be directory caches. It should be noted here that there is significant trust, perhaps undeserved, placed upon the guard nodes within the Tor network. The use of information theoretical PIR can actually help mitigate potential issues that come along with this trust. Unlike Torsk[41] which adds some aspect of P2P networking to minimize network load, and necessary information required for network function, PIR-Tor's model is still centralized, as in normal Tor. In short, PIR allows clients to request from the more centralized servers only the information they need, lessening the amount of traffic used in the network to maintain state, while keeping or improving current security standards [20].

As it inherits current security standards of Tor, there are still potential problems left unaddressed. First, any directory cache, and even any motivated user, is able to obtain a global network view. For a state actor interested in censorship, it would be trivial to perform a denial of service against specific users in this setup, assuming they control a directory cache. Also, the performance expectation of the recommended computational PIR system is not high, and the information theoretical PIR requires a larger trust surface, which can be impractical [20, 21]. Additionally, there are issues with, for example, the use of Snader-Borisov [42] critera for node selection weighted by bandwidth and/or throughput capability, is not enough to prevent a motivated (or wealthy) malicious actor to unduly influence the Tor network (i.e. by purchasing many servers with high bandwidth, thus encouraging that their servers are selected).

As opposed to the oblivious transfer technique used by PIR-Tor, which in practice can prove impractical, ConsenSGX uses Oblivious Random Access Memory (ORAM) as well as the Security Guard Extensions (SGX) [43] technology from Intel (or other similar extensions by other vendors, collectively known as Trusted Execution Environments (TEEs)) to improve the transmission of Tor network state [21]. ORAM, a technology which allows access to data from a server or similar construct in an oblivious manner, is an attractive methodology for

this purpose, for the same reasons that PIR is, providing similar capabilities and guarantees as ORAM [20, 21].

Sasy and Goldberg [21] define a newer query paradigm for nodes to obtain some or all of the network consensus. The network consensus held by the directory servers is changed into two functions which enable the ORAM process, and the ORAM access sections are held in a tree-like data structure. To obtain new descriptors a user will use a parameters list to choose a node in a bandwidth aware way. By bandwidth aware, we have only a sorted list by order of bandwidth from best to worst [21]. The authors state the bandwidth ratings are either self reported as in current Tor [11] or via Snader-Borisov, like in PIR-Tor [20, 21, 42].

All secure operations are done within one of the requisite TEEs of the directory caches. Once user receives the public key of the SGX of the directory cache, they send an encrypted request for specific records it determined via the parameters they downloaded earlier [21]. Since this is via an oblivous manner, the cache does not know the exact record requested. When guard nodes are updated, a full network consensus is used, since current guidance from Tor does not recommend updating the guard nodes for a few months [11].

Again with ConsenSGX there are still times in which the full network consensus is downloaded, and therefore the full network state. Additionally, there is no prevention of enumeration of the whole network state. Then, as admitted Sasy and Goldberg recognize the security trade-off that exists by relying on the TEEs [21]. Recently, there have been successful attacks on the Intel SGX outlined in [44, 45, 46], which negates the trust assumption based on the TEEs while any of these vulnerabilities are viable.

In 2020 Komlo et al. [47] describes *Walking Onions* as a method to improve the scalability of the Tor network. Walking Onions differentiates relays and *clients* (those using the network, but not acting in any routing capacity) of the network in a more specific way than original Tor. No longer do all nodes need a complete consensus, but only those acting as relays, which allows for a constant-size client overhead. In the paper they define two different algorithms, *Telescoping Walking Onions* and *Single-Pass Walking Onions*, which is the more innovative of the two. Both require some $i$ which cannot be influenced by a third party, in telescoping walking onions $i$ is chosen directly by the client at each step. For the Single-pass version, $i$ values are derived from a client controlled random choice, and the choices are verifiably random. Each relay descriptor is modified to include a range of numbers, corresponding to its weighted chance to be a relay. The weighted chance is the range of possible $i$ values to which it answers. One additional modification is the inclusion of an authentication tag created by the directory authorities [47].

To extend a circuit using the telescoping method, a client must provide either $i$ or information to derive $i$ along with the public half of an ephemeral key pair $g^x$. The client is then returned the appropriate relay corresponding to that $i$. the client can then check that the relay indeed is responsible for that $i - range$ and continue the process. The single-pass method involves an additional key pair, but saves in overall message complexity due to requiring only the single-pass [47]. The intuition for the single-pass procedure is that the use of a verifiable-random function (VRF) will allow a client to know that some value $i$ was chosen uniformly from a distribution without choosing it itself. Instead, it produces a set of key pairs, one for the VRF inputs, and one for the normal path construction procedures, and instead of verifying the relay record itself as above, it verifies the VRF proof [47]. Even though this solution introduces a far more scalable solution than prototypical Tor, these algorithms still rely on a centralized infrastructure, namely the directory authorities, and therefore still suffer from the one of the ills of centralized solutions–that is the centralized component is an attractive target for attack and has supreme information and power over the network in the event of its compromise.

## 3.2 Structured Peer-to-peer-based Approaches

Decentralized systems operate in an untrusted environment, with many malicious actors acting at any one time. While considered to be more scalable than their centralized counterparts (naively it can be considered that network capacity increases directly with network size,) special care must be paid to ensure robustness in the face of the added adversity. Moreover, without centralized coordinating servers, there must be a way to transmit network state to new nodes. Not considering protection of the initiator or destination for a moment, there have been multiple influential works on secure networks, notably [24, 25, 26].

But even those networks which consider anonymity must protect against many attacks. These include *fingerprinting attacks* which are a class of attacks which allow an adversary to probabilistically or deterministically determine the initiator and/or destination of a communication [37] as well as the *bridging attack* in which an adversary uses the lack of knowledge of other nodes to determine information about their identities i.e. if a node only knows one other node, etc [38]. Further more specialized attacks also exist, such as the *eclipse attack* [36] which quickly stated involves compromising some consensus mechanism through controlling enough nodes to collude against it, *selective or total denial of service* [48], a *Sybil attack* where an adversary takes advantage of the ability to cheaply creates nodes to flood a

network with malicious entities [39], or even a *range estimation attack* [16] which is explained later below. This is in addition to potential problems also faced by the centralized networks, such as censure by public authorities, and potential bottlenecks from overused nodes.

In their paper, Castro et al. define requirements and common attacks on peer-to-peer networks. In it they define primitives and risks for *secure routing*, *secure node ID assignment*, *table maintenance*, *secure message forwarding*, and *self-certifying data*. They also outline a series of attacks which should be considered, and by which all networks should be judged. Critically, it lays out that it is not enough to assume nodes follow the protocol correctly, and some level of malfeasance should be assumed. It leaves the following topics unaddressed: *recursive queries*, as they are very hard if not impossible to do anonymously, and *information leakage* which is an exceedingly broad topic [49].

Mittal and Borisov [50] set about implementing these defenses, and demonstrated the insufficiency of the secure message forwarding primitive outlined by Castro et al. The limitations of Distributed Hash Tables (DHTs) with these defenses are related to information leaks introduced during routing. Similar information leaks were taken advantage of by Wang et al. [16] against NISAN [27], with the range estimation attack.

Salsa is a DHT-based project to create a scalable anonymous network by design. As is typical for anonymity solutions, Salsa introduces misdirection via routing requests via intermediary nodes. Novel for Salsa is the ability to have knowledge of only a few nodes, while creating routes from the total view of all nodes [40]. the ID space, which is based off of user IP addresses, of Salsa is divided into a binary tree. Each node has total knowledge of its local area in the tree, and limited knowledge of other areas in the tree [40]. To secure the lookup, similar to the process in NISAN [27], Salsa uses a bounds check to help prevent lookup bias. Lookups are redundant, and recursive, and done through random nodes within their local area of the tree. Because of the knowledge of a few nodes in the global neighborhood, nodes are able to communicate throughout the network, with only limited knowledge themselves. The circuit is then formed via the redundant lookups at multiple levels, until finally one route is selected. Unlike Octopus [16], there is no method to remove or inform others of offending nodes.

However, Mittal pointed out Salsa's susceptibility to various attack vectors [51]. First and damningly information leaks introduced by the redundant lookups allow an attacker to discover the initiator of a lookup, if a local node is compromised. Furthermore, Borisov [48] noted the ability to cause significant issues if, even at a malicious rate of $f = 0.2$, malicious actors are capable of performing selective denial of service on within the network, to ensure

that as many routes as possible are through malicious nodes. Additionally Borisov outlines a possible public key modification attack when an entire stage of the requested nodes in the tree are malicious. Naturally, these can work synergistically by denying any route selectively they cannot perform the modification attack, whenever possible.

NISAN [27] is a network information distribution system meant to mitigate issues with redundant routing networks, due to a lack of scalability resulting a reduced number of paths. The goal is to provide a robust searching algorithm which preserves anonymity while providing better reliability, via gauging the plausibility of finger tables, and additionally using techniques to protect the identity of the destination. NISAN is based on a Chord-like DHT.

The main idea behind the functionality to hide the destination is to request entire tables from intermediate nodes. This hides the target node. This is done to prevent fingerprinting and bridging attacks. Enabling this involves two main approaches considered novel to NISAN; *aggregated greedy search* including an *iterative Chord lookup* which involves correlation of information from multiple sources and whole finger table requests to prevent misinformation, and *bounds checking*, which compares returned information about a following node or destination to some defined distance bound to determine if the information is plausible [27]. The aggregated greedy search occurs as follows: given a requester $v$, a lookup target $x$, and a network-wide Neighbor limit $\alpha$, $v$ requests the entire finger tables of its fingers, and from the returned fingers attempts to find $x$ from the $\alpha$ closest nodes it knows, according the the underlying DHT's distance metric. the lookup loops until for one round the $\alpha$-list of closest neighbors doesn't change, the search ends [27]. To accomplish bounds checking, NISAN compares the ideal finger identifiers for a node versus the reported fingers in a finger table. If all the identifiers are near to the ideal identifiers (within a threshold) the finger table is considered plausible. This helps defend against overly biased information dissemination by malicious actors [27]. This technique is described in more detail in the section describing GuardedGossip 2.5.

While NISAN makes significant strides in improving the security posture of a network, while maintaining anonymity of the destination and robustness to malicious nodes (particularly with bounds checking,) it is not wholly successful, falling to a documented range estimation attack [16]. A range estimation attack, which is possible mostly due to the directed nature of Chord, occurs when an attacker understands that in a directed-connection environment, like Chord, there are limits in which all nodes operate. A querying node $Q$ will never query a

node after its destination. As $Q$ proceeds with its query, if it crosses some malicious node $M$, $M$ knows necessarily that $Q$ exists somewhere before it on the chord circle.

Myrmic [32] is a DHT operating and behaving much like Chord (and is in fact built atop Chord), with a distinct difference and addition in order to handle potential malicious or adversarial behavior. Namely, it adds a *root verification protocol*. The root verification protocol is enabled via the use of *nCerts*, which contain current information about a node and its surrounding neighbors. If the network is unchanged, this indicates freshness. This means there is a certificate authority, who is responsible for issuing and revoking certificates as well as updating nodes when a certificate is revoked. All certificates are signed by this certificate authority, and the certificates contain information of the node's surrounding neighborhood, who act as witnesses. That is, each witness should also contain information about that node, and therefore can be used as verification. Using the root verification protocol, a requester is therefore able to verify that a responder is the true root of the requested key, as they have a fresh certificate which can be verified[32]. However, Wang et al. recognized that on node churn all certificates must be reissued leading to a lot of state information transmission in high-churn networks. Myrmic is one of the two basis networks which are the crux behind Torsk [41].

Torsk [41] introduces a Tor-compatible replacement for the directory service. This solution uses a Kademelia (Kad) DHT [26] with Myrmic [32] characteristics. A Myrmic certificate authority issues *nCerts* as in Myrmic, i.e. containing information about the node itself, and its neighbors (the *nList*). Additionally, each node maintains another certificate authority-signed list of random nodes (the *rList*), who are chosen by the certificate authority. Torsk also uses a concept of buddies. Buddies are chosen via a random walk, where a node $Q$ chooses uniformly random from its known nodes some node $P$, and requests its known nodes. If all certificates and routes are verifiable by the certificate authority, $Q$ performs an onion route through $P$ to the next uniformly chosen node, and continues until it reaches a predetermined path length. The last node is added as a buddy. When routing, $Q$ again randomly chooses a node from its known nodes, $P'$, and $P'$ asks one of its buddies for the desired identifier. This hides the relationship between $Q$ and $P'$. The lookup for the identifier is done within the DHT for its root node, which is the next hop in the circuit [41].

However, the buddy selection process allows for a denial of service attack [34]. This is critical, because an attacker can influence route selection via selectively denying service to routes it cannot control. There are two phases, first is to overload the neighbors of the Kademlia lookup target. This will deny the route. Next, is to take advantage of the buddy selection

process. If an attacker can simply cause one step of the random walk to fail, the entire walk is started over. Therefore, any time a walk crosses a malicious node it can provide an invalid certificate, causing the random walk to start anew. This significantly hampers the circuit building process, and limits the total amount of available circuits to those with malicious nodes [34].

Whereas other systems do not necessarily define the process by which a node is credentialed, there are a subset of structured topologies which rely on Distributed Key Generation (DKG) alongside the concept of threshold cryptography. Given a set of $n$ nodes and a threshold parameter $t$, $DKG(n, t)$ enables the requisite nodes to create a secret *collectively*, without any individual node knowing the whole secret, and without a central authority. Now, in order to sign or decrypt a message at least $t$ nodes must collude using their shares of the key to sign or decipher the contents [52]. These principles are critical for defining what are known as quorum-based approaches, which address potential security vulnerabilities via collective action.

Young et al. define two main and novel algorithms for robust communication with the presence of potentially malicious actors on top of a DHT with a quorum-based topology, with a malicious fraction $f \simeq 0.33$ [28]. Notably introduced into the DHT is the use of threshold signatures, and distributed key generation to assist in performing secure cryptographic operations in the absence of a trusted central authority to handle certification and Public Key Infrastructure (PKI). This is enabled, semi-inherently, through the use of the quorum topology which allows nodes to act as a group, where in non-quorum based overlays each node must act independently. In a quorum topology, nodes are layed out into groups, $Q_i; i \in [1...n]$, and nodes within one group are able to communicate freely as "neighbors". Two quorums can communicate, wholly and freely if they share an edge similar to two singular nodes in a non-quorum based solution [28].

Specifically the problems solved regarding a DHT as outlined by Young et al. are: (1) key generation and maintenance, and (2) robustness to spamming attacks. Which are solved via the introduction of the aforementioned distribute key generation, and a prove-and-verify system and protocol. This last piece is enabled by two new protocols $RCP-I$ and $RCP-II$, which provide robust communication with low overhead [28].

However, unlike some other protocols mentioned here, i.e. Octopus DHT, ShadowWalker, etc., it does not address initiator anonymity and/or query privacy [16, 28]. Also of concern is that even if a method of preserving anonymity is naively added, there it is potentially susceptible to a range estimation attack.

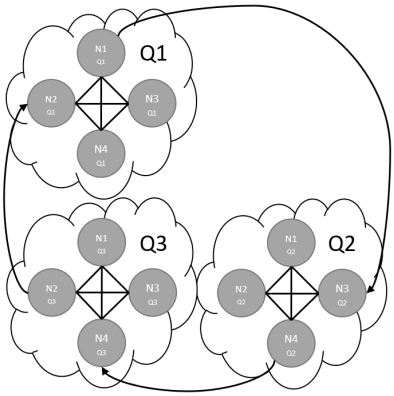


Figure 3.2: Simplified overview of a Quorum-based DHT

Backes et al. define a method for using an oblivious transfer protocol to add query privacy to DHTs, more specifically the quorum-based DHT as defined by Young et al., mentioned above. The basic idea of an oblivious transfer is that a requester can request a specific piece of information, without the information arbiter knowing which information was requested, and without the requester reading more information than is required. This allows for greater privacy than requesting an entire neighbor table from another node [53].

Created here is a novel index-based oblivious transfer protocol, added to $RCP - I$ and $RCP - II$ [53]. The main goal is to limit data revelation at any one time, so as to prevent a malicious actor from spamming and receiving too much information about the network as a whole, while maintaining query privacy to another node or server. Here Backes et al. use a public oblivious transfer algorithm [54], to enable adding oblivious transfer to various aspects of the query process in DHTs [53].

The protocols given by Backes et al. provide then robustness to a Byzantine adversary, spam prevention, and preserve anonymity or initiator and destination. As stated above, this solution would still be susceptible to the range estimation attack.

## 3.3 Random Walk-based Approaches

A fully P2P interpretation of Chaum's 1981 mix systems [10, 55], was introduced by Rennhard and Plattner [33]. MorphMix realizes a system in which all nodes act as available routers. Allowing all users to operate as routers solves the potential bottleneck of having a limited set

of routers in the original mix system, or even in Tor [11]. MorphMix relies on random walks and witness nodes to secure the connection process. When setting up a circuit, each node only selects the next step of the tunnel, due to this global network knowledge is not required. The witness nodes assist in a collusion detection mechanism, which was later determined to not function well enough to preserve anonymity by Tabriz and Borisov [56].

ShadowWalker [12] provides a low-latency anonymous communication network. In short, ShadowWalker provides so-called *shadows* which form mirrors of a given node. A node and its shadows form a *neighborhood*. ShadowWalker incorporates three distinct steps (1) adding redundancy into the network topology, (2) circuit building, and (3) a secure lookup protocol. (1) is done via the concept of shadows; who additionally sign routing tables to provide verification, and prevent routing table manipulation. (2) and (3) add the shadows into circuit building and lookups, to provide continuous verification, as well as detection of manipulation, with the goal of preventing circuits with malicious nodes. When a lookup is performed in circuit generation, the shadows serve to verify the node's response. So long as at least one node within this neighborhood isn't compromised, the network should be robust to malicious actors attempting to influence the system. ShadowWalker is built on top of *Chord* [24]. Specifically, it adds greater redundancy to Chord via the shadow relationship.

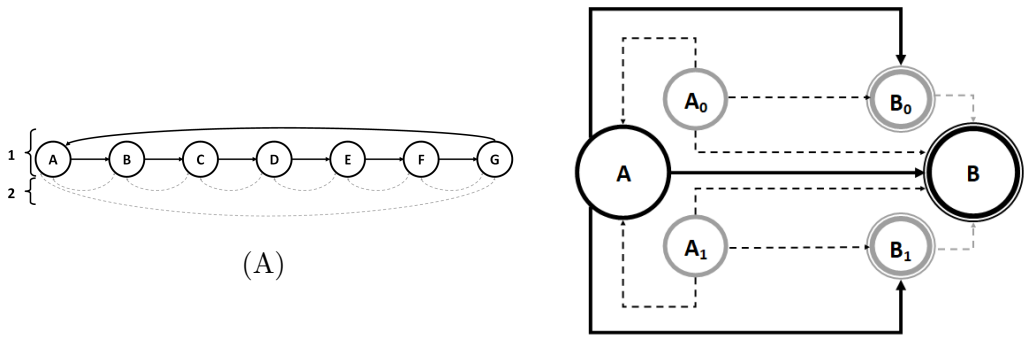


Figure 3.3: (A) Simplified view of a Chord circle with Shadow relationships, (B) Detailed view of two connected neighbors in ShadowWalker

Shadows are determined the by the successor relationship defined in Chord. Nodes have successors and predecessors, and contain some knowledge of each. The graph is directed, and cyclic. However, Significant limitations and attacks exist in and for ShadowWalker.

To more formally state the relationship of shadows and nodes; for a predefined redundancy

parameter $r$, the shadows form a set of roughly $r/2$ predecessors and successors, shown in Figure 3.3 (A), for $r = 2$. Given a node $A$, neighbor $B$, and $r = 2$, ShadowWalker defines the following properties to build redundancy into the network topology, shown in a detailed view in Figure 3.3 (B):

- $A$ has connected shadows $A_0$ and $A_1$, and $B$ has $B_0$ and $B_1$
- $A$ holds information about (i.e. is connected to) shadows $B_0$ and $B_1$
- For $i \in 0, 1$, $A_i$ is also connected to $B$, $B_0$, and $B_1$.

Additionally involved with redundancy is how the protocol can handle network churn. These lookups are used to update its information for which nodes it is a shadow, and then recursively performs the secure lookup on those nodes' neighbors. By doing so, in the event of a node failure among its neighbors, it can later connect to another node in the network. This allows ShadowWalker to efficiently add or remove nodes from the network, while maintaining the redundant topology.

ShadowWalker has significant issues, brought to light by Schucard et al. [36], who showed that it is susceptible to an eclipse attack, and a denial of service attack through the shadow relationship, and how it handles malicious shadows. Additionally, ShadowWalker makes no effort to prevent full network knowledge, as with all the protocols listed here.

Considered state of the art for anonymized communication is the Octopus DHT [16]. Octopus addresses shortcomings in its predecessors, most notably NISAN, Torsk, and ShadowWalker. The novel features added to Octopus DHT can be summarized as follows:

**Anonymous paths and relays:** bounds checking and active malicious node identification

**Secret neighbor surveillance:** actively check validity of predecessors' neighbor tables

**Secret finger surveillance:** use anonymous lookups to check validity of finger tables

**Certificate authority:** collects and collates reports, issues certificates, and removes malicious nodes

Octopus is based on Chord, with the additional qualification that nodes hold information on predecessors and successors. The first phase of the routing process is seen in Figure 3.4 (B), the anonymous paths are set up via random walk process, with two distinct phases. The first uses onion routing in an extending circuit, and the second a deterministic seed-influenced random function [16]. Next, of particular note here are the secret neighbor surveillance and

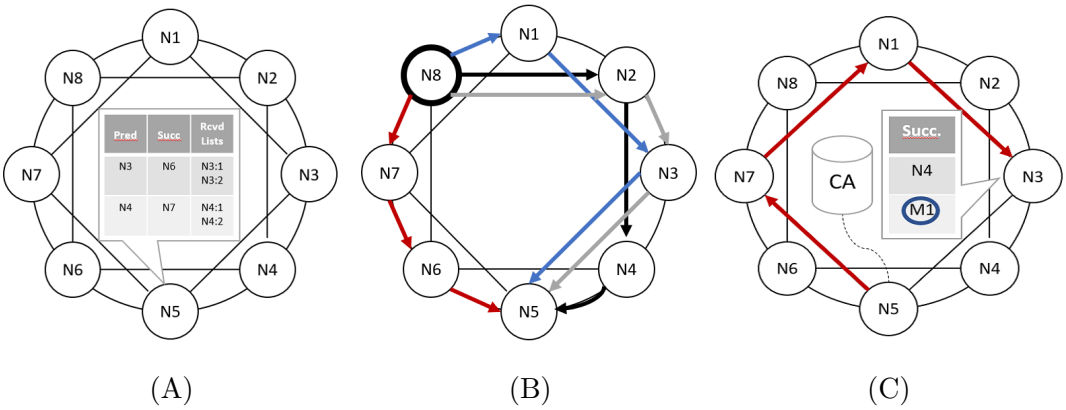


Figure 3.4: (A) A brief overview of an Octopus DHT overlay, (B) Routing via multiple pathways, and (C) Secret Neighbor Surveillance

secret finger surveillance. Secret neighbor surveillance is illustrated in Figure 3.4 (A). Shortly summarized, a node $A$ makes an anonymous query to a predecessor node $B$ and requests its successor table. $A$ then checks to ensure it is listed in the table correctly. This works, because a node's predecessor and successor lists are the same length, therefore if $B$ is in $A$'s predecessor list, $A$ must be in $B$'s successor list. If $A$ detects manipulation, it reports $B$ to the Certificate Authority (CA). In order to perform secret finger surveillance, a node $A$ selects a random finger $F$ of another node $B$ (which it knows because it received the finger table at an earlier time). $A$ requests $F$'s predecessor list, and requests one of these predecessors. It then checks if this predecessor has a truer or more ideal finger than $F$. If it does, $B$ is reported to the CA [16]. When $A$ makes a report, the CA asks for proof from $A$ (i.e. the signed routing tables), and if it is sufficient, kicks $B$ from the network, this process is shown in Figure 3.4 (C).

These novel features of Octopus are suitable for preventing various attacks against DHTs. Octopus sufficiently addresses preventing manipulation and flooding of finger tables, and removing malicious nodes. However, its techniques are not free, and it adds measurable overhead [16]. This leads to the open question of how one can obtain the active and passive defenses of Octopus, in a more scalable manner.

# 4 Approach

For a basic understanding of the network setup, GossipChain builds upon a Chord DHT while borrowing ideas from previous works, namely GuardedGossip. It uses a gossip protocol to spread chains of information of witnessed nodes. These chains are created and sent only if a set of checks are passed, which introduces trust. The main goal of this thesis is to solve the issue of network scalability in an extensible and secure way, i.e. mitigate the risks involved via active and passive attacks commonly used against network information discovery systems, while also accounting for the concepts of limited network overhead and load-balancing. The underlying DHT is Chord.

GossipChain introduces a chain $C$ which is a collection of records $r_i$ with $i \in \{0, 1, ..., n\}$ where $n$ is the chain length. Each record $r_i$ contains the information indicated in Figure 4.1. Using the hash of the previous record, it forms a linked list-style chain.

| Record $r_i$ | | | |
|---|---|---|---|
| $hash(r_{i-1})$ | node identifiers | bandwidth | node address |
| Total: 112 $bits$ | | | |

Figure 4.1: Singular record within a chain

The chains spread are dubbed *GossipChains*, and are a form of trust chaining. Each record is formed by the node requesting gossip information. The active and passive defenses used are *bounds checking* and *witness list checking*, borrowed heavily from the GuardedGossip paper by Panchenko et al. [14]. The checks are performed on each node as it sends gossip information. Novelly introduced here is the concept of *lineages*, and relatedly, *spot checking*. Lineages are a history which can be tracked to ensure all nodes in a chain responded honestly according to the protocol. To this end, in addition to the trust chains, there is a doubly used bandwidth metric both to limit the life of chains, and therefore network overhead, but also to allow a high-bandwidth node also to create a longer living chain.

The system relies on some critical assumptions, without which the system does not function. It is assumed there is a method to exchange public keys, which are required for the use

of digital signatures. GossipChain does not require a full CA, or any centralized online component. For the writing of this thesis all keys were exchanged out-of-band. It also requires a so-called *bandwidth oracle*, to grant the bandwidth scores. Next, this work does not address the Sybil attack (only cursorily), and so it is assumed there is a way to limit node join in the network (for the purposes of this thesis, this was IP address and a publicly available random component.) There are additional assumptions related to the malicious environment in which the network will operate, which are detailed in the following section.

## 4.1 Attacker Model

As stated in Chapter 2, this thesis assumes an attacker fraction $f = 0.2$, similar to related works in the field. Further assumed is an attacker with passive and active capabilities, who can act arbitrarily in relation to the protocol. That is, the attacker can create messages at whim, save data which otherwise would be ethereal, and passively monitor all traffic coming across nodes it controls. However we do not assume an attacker with the ability to have total oversight of entrance and exit within the network. It is not assumed the attacker has the ability to break the assumptions on which the system is built, and mentioned above. Also, the attacker is not performing a Sybil attack.

## 4.2 Open Problems

Chapter 3 introduced a series of related works each of which addresses secure and anonymous communication. However, each solution was found to be lacking in some critical way. Most notable was the necessity in the state of the art (i.e. Octopus) for a centralized authority, or with Tor's lack of scalability. Therefore this section will introduce the various mechanisms GossipChain used in attempting to solve these problems, through the use of trust chains to distribute network state, and providing the necessary checks to secure communications without the need for a centralized authority to actively kick nodes from the network, or authorize their ability to transmit information. This will be done through the bounds check, witness list check, lineage check, and spot check.

## 4.3 Fundamentals

GossipChain was not made in a vacuum, it is built upon previous and historical solutions, which have paved the way by creating various techniques that are proven to efficiently solve various problems. GossipChain uses an established DHT in order to provide lookups, techniques proven in NISAN to secure them, and hardware-based node identifier assignment to prevent malicious actors from influencing their network position.

### 4.3.1 Node Identifier Assignment

It is critical that a node not be able to define its place in the network by controlling its identifier. The ability to determine its place in the network would allow it to collect passive information from arbitrary places within the network, and encourage information leakage. As a solution to this problem, in GossipChain node identifiers are assigned via a deterministic random, seeded by the combination of both a hardware identifier (e.g., IP address), and some publicly available source, which for the simulation is a hash of a blockchain taken at the time of network creation.

When practically used (i.e. not in simulation), other potential sources besides hardware IDs are possible, so long as they are verifiable. The assumption here is that IP addresses are not inherently cheap, and therefore an attacker is not capable of just trying again to get a suitable place within the network. This is important, because without this assumption GossipChain becomes susceptible to attacks considered outside the scope of this thesis. Most notably, from an attacker with a significant number of malicious nodes who all have some form of finger relationship, thus having the ability to create valid chains. Therefore, an attacker who has the ability to cheaply obtain many IP addresses is outside the scope of this thesis.

### 4.3.2 Underlying DHT

As this thesis is heavily influenced by GuardedGossip, it was chosen to follow in its footsteps and to use a DHT to providing the necessary structure over which to build an overlay and perform lookups. In Chapter 2, Chord was introduced and explained as a background of the thesis, and is also the underlying DHT. Like GuardedGossip, the network is expected to use $32 - bit$ identifiers, with a maximum network size of $2^{32}$ possible participants. This is not a theoretical limitation, but was the chosen value for simulation.

### 4.3.3 NISAN and GuardedGossip

Lastly, GossipChain incorporates the checks on finger tables originally introduced in both NISAN [27] and GuardedGossip [14]. This means we judge the plausibility of the finger tables of nodes via bounds checking and witness list checking. These techniques are further explained in Chapters 2 and 3. This allows GossipChain to inherit the forebearers' active defenses, which introducing modifications to improve upon its drawbacks (i.e. NISAN's range estimation attack). Additionally taken from GuardedGossip (in an unmodified way) is how it uses NISAN's lookup procedure during finger table initialization.

### 4.3.4 Bounds Checking

In order to measure the plausibility of a finger table, a process known as *bounds checking* is used. Here, a node determines $\hat{n}$, an estimated number of nodes in the DHT. $\hat{n}$ is approximated by calculating the distance between the given identifiers in its finger table and what the optimal identifiers would be if the underlying DHT was full (i.e. $2^m$ nodes, in Chord, with $m - bit$ identifiers). This allows an approximate calculation of node density $d = 2^m/\hat{n}$. NISAN [27], the introducting paper of this technique, further defines a tolerance factor, here $t; t > 0$. A received finger table is considered plausible if its node density $d' < td$. GuardedGossip defines its tolerance value to be $t = \sqrt{1/f}; f := 0.2$, which is the same $t$ used in GossipChain. Information sent that fails the bounds check is discarded.

### 4.3.5 Witness List Checking

A *witnessed node* is any node ID come across during normal operation of the network. Each node maintains a list of nodes it has witnessed. This allows it to perform a sanity check to see if a node has ever come into contact with a more correct finger (i.e. closer to the *ideal* finger ID) for a node than what it receives from the finger table. To that end, if witness list checking is performed on a finger table from a node $B$, by a node $A$, then $A$ first calculates the ideal fingers of $B$, as if the DHT was fully populated. Next, it checks its witness list, which is a collection of node identifiers it has witnessed either during stabilization or gossiping, and determines if one of these nodes is a more correct finger for $B$ than what it is reporting its fingers to be. That is, does the witness list of $A$ contain a node with an identifier closer in distance to the ideal finger than what $B$ says? If $A$ does indeed have a better finger, then the data from $B$ is discarded, and considered untrustworthy. With a random 1/3 chance, $A$

will also perform a liveness check on the more ideal finger. If the witness is not live, it will be removed from the witness list, although this means the information from $B$ might be correct, it is still not used.

### 4.3.6 Lineage Check

Each chain has a so-called *lineage*, which is every node counted in its records. This means the path of each chain can be checked, along with the claimed relationship within the record. Each gossip round, a chain's lineage will therefore be checked for plausibility. This is, essentially, a modified and combined bounds and witness check. First, it will check that each reported finger in the record is *plausible*, according to the same threshold value in the bounds check, it can then be determined if we have seen a more realistic finger according to nodes we have witnessed. A chain which fails the lineage check is discarded. This can prevent chains of maliciously created records from propogating within the network.

### 4.3.7 Spot Check

Normally when a chain will be transmitted, the bounds check and witness list check will only occur against the node from which information is being requested. The spot check will choose a set of random nodes from the chain at varying times when running the protocol to *re-run* those checks. This is to prevent a malicious node from acting honest at some point, and dishonest later. It can be used to be sure various nodes are still reporting the same or similar information to when the chain was created. In order to do this, the finger table will be requested again via tunneling, and bounds checking and witness list checking performed locally, and is another point at which a chain could be discarded.

### 4.3.8 Digital Signatures

In order to provide veracity to the claims represented by a record within a chain, GossipChain assumes the use of a digital signature algorithm. This implies there is a public/private key pair for each node, and therefore a method for exchange of the public keys. Both key derivation and exchange are considered outside the scope of this thesis, and for the point of the simulation where it was tested with signatures, keys were exchanged out-of-band. At no point does GossipChain require an online CA, or any other form of centralized authority. In any case, the digital signatures allow for accountability of sent information, and notably

prevent a malicious node from creating fake records to paint otherwise honest nodes as malicious. For this reason it is considered fundamental to the network that digital signing is possible.

## 4.4 Introducing GossipChain

When developing the solution, trust chaining was decided as a way to *lessen* the network overhead of previous solutions and incorporate potential load balancing. Answers solving giving measurable anonymity, have been widely researched in the past concerning DHTs, however these previous solutions have drawbacks, which have been outlined in Chapter 3.

Ensuring network scalability, and providing better low-latency service, is a topic fairly ripe for exploration. For this novelly introduced is a bandwidth metric, allowing traffic to be efficiently routed through the network in a way which encourages high bandwidth nodes to take on more of the network load. If possible we should ensure the network is attractive and usable by a wider audience of users, as mass adoption is the greatest mitigating factor to many vulnerabilities present in P2P networks and solutions. In order to be able to claim security from scale, the network must provably be able to scale. The criteria here is to be at least as good as the current most popular anonymity network, namely Tor. More detailed information on the solutions mentioned in this paragraph can be seen in the related works, in Chapter 3.

### 4.4.1 Outline of a GossipChain

A GossipChain is collection of records collectively creating a *trust chain*. Versus some previous solutions, GossipChain attempts to send finger tables less often. Against a solution like GuardedGossip, which has a more similar overhead rate, GossipChain attempts to find malicious nodes and remove them from consideration for gossiping or circuit creation faster or more reliably via its checks. In GossipChain finger tables are only used when requesting and extending a chain. For circuit building, only the collected trust chains are used.

Figure 4.2 shows an overview of the GossipChain protocol. Bounds checking witness list checking, the lineage check, and the spot check are explained in Section 4.3. The steps of the protocol are as follows, assuming a fully set up and stabilized Chord circle, as well as a public source for a random seed, which is updated each round:

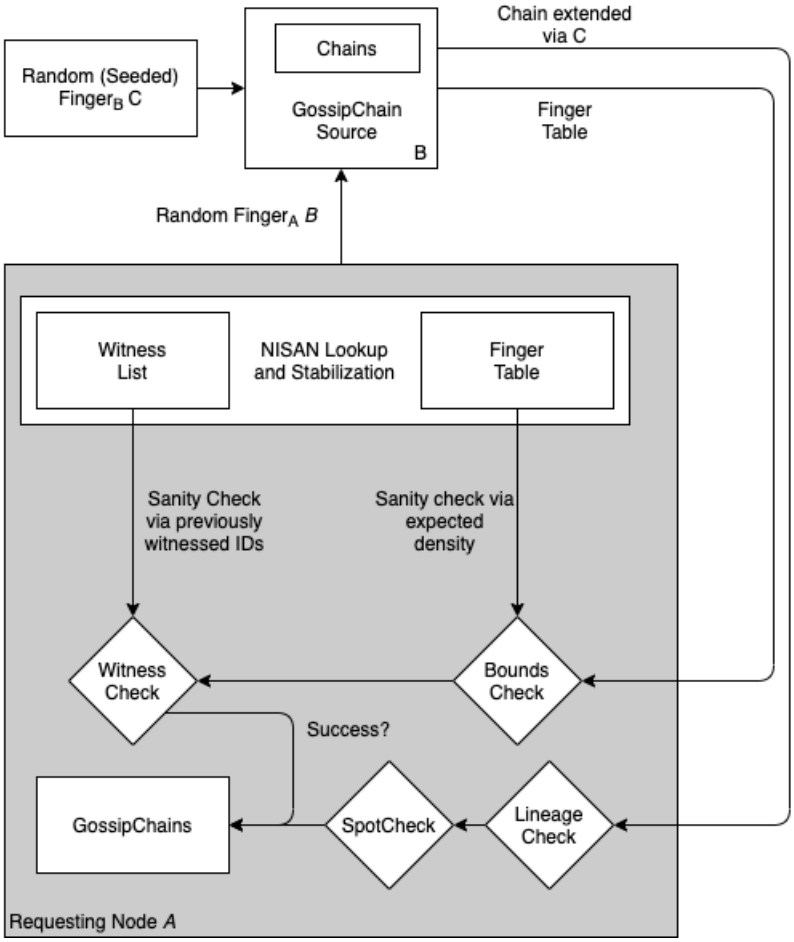


Figure 4.2: A graphical overview of a GossipChain Request

1. As a prerequisite, all nodes in the network create a set of records $R = \{r_0, r_1, ..., r_i, ..., r_m\}$, with $r_i$ a record formed from $finger[i]$.
2. Each record will form the head of a chain, giving each node a set of chains $C = \{c_0, c_1, ..., c_i, ..., c_m\}$ with $c_i = \{r_i\}$
3. A node $A$ will randomly choose a finger $B$ from which to request gossip
4. $A$ requests the finger table from $B$ as well as a complete list of chain hashes ($hashes = \{hash(c_i)\}; c_i \in C_B$)from $B$. These chain hashes will be used to ensure the proper chains are sent later.
5. $B$ limits total number of requests ($r_{max}$) from any given node over a few rounds, in order to deter and prevent denial of service attacks.

6. $A$ performs bounds and witness list checking on the finger table of $B$

7. $B$ then uses the public seed mixed with the identifier of $A$ (via a repeatable, known, mathematical operation, in the simulation simply XOR) to "randomly" choose a chain via a deterministic seeded random operation (or if requested, up to 3 chains) and a finger with which to extend the chain.

8. The newly extended chain is then sent to $A$

9. $A$ checks that the proper chain was sent, and the proper finger used to extend it, by deriving the seed and checking against the finger table and chain hashes.

10. $A$ performs modified bounds checking on the entire chain *lineage*, to be sure each reported link is a finger of the previous node. This process is explained in Section 4.3.6.

11. $A$ also randomly chooses some links in the chain and request their finger tables in order to perform full bounds and witness list checking, as a *spot check*. This is especially useful in longer chains, which may have nodes which honestly extended the chain at some point, but are acting maliciously at a later time. This is explained in Section 4.3.7.

12. Assuming all checks are passed, $A$ then accepts the chain(s)

13. Chains which fail the checks are discarded.

$A$ only accepts a chain assuming each step is successful. $A$ can then use the collected chains to form routes over the Chord circle. It would do this by considering all nodes in all chains without duplicates. It would then weight selection in a random function by the bandwidth score of the node and an inverse of the distance of the node identifiers. Therefore, $A$ would be more likely to choose a high bandwidth distant node, than a low bandwidth close one. An overview of this process is shown in Figure 4.2. In step (9), this hash check is possible because of the linked list-style of the chain. In actuality, the hash of the chain at any step can be checked, just by removing nodes from the end. This is because each record is capable to be considered on its own. The only connection between records, which makes them a chain in aggregate, is the entry of the previous records hash.

### 4.4.2 Network Bootstrapping

A node's lifecycle in the network begins when it first joins the network. The collective act of nodes joining and leaving the network is known as *churn*. Similarly, a network's lifecycle begins with a process known as *bootstrapping*.

A node's identifier is based on the IP address of the user, and a random value derived from some publicly available component, e.g., the hash of a public blockchain at network creation. This is in order to prevent a malicious user from having undue influence on their placement within the Chord DHT. If a user is able to influence their position within the circle, it opens up the network to a series of attacks which can target specific users, or create malicious environments allowing the eclipse attack or other malicious neighborhood attacks.

Before being introduced to the network, the node must create a public private key pair. This key pair will be used for the sections of the protocol requiring use of digital signatures. After joining the network the first time Chord updates and stabilization occurs as normal, with an additional GossipChain step. During bootstrapping, the node will keep track of all nodes it views which are added to its *witness list*, which is a list of node identifiers it has already seen. A set of initial GossipChains is created from the nodes it has direct knowledge of after the underlying Chord stabilization phase is completed. This means the joining node creates a list of initial GossipChains which contain records of it and each of its fingers.

When wishing to newly join the network, a node needs to have knowledge of a few nodes to begin the bootstrapping process. It is assumed that these nodes are exchanged out-of-band with trusted parties. This thesis is not concerned with what happens if all of the bootstrapping nodes are malicious, but inherits the protections from NISAN [27] that allow bootstrapping to be trusted so long as one node is not malicious. Therefore, bootstrapping occurs as in Chord and NISAN, with an additional GossipChain bootstrapping step. The previously stated creation of one chain per finger in the finger table of the node after Chord bootstrapping. All nodes seen during bootstrapping are also added to the witness list, as in GuardedGossip [14].

### 4.4.3 Stabilization and Churn

Stabilization occurs as normal in Chord, and explained in Chapter 2. The only modifications mirror those of GuardedGossip. That is, nodes seen during stabilization are added to the witness list, which is purged of older entries in a first-in-first-out method. After Chord

stabilization, a gossip round is undertaken according to the aforementioned GossipChain protocol.

The decision to expire chains is designed to introduce uncertainty of knowledge. They either expire by time, or are chains which are gossiped to another node and are discarded after gossiping occurs with a $v = \frac{1}{4}$ chance. New chains are also randomly created during the gossiping round, when chains are requested. This is done with the same $v$ chance, or when a gossip request cannot be answered otherwise. New chains are created by making a new record $r_i$ with information from a random finger. $r_i$ is then set as the head of a new chain $\hat{c} = \{r_i\}$.

### 4.4.4 Circuit Creation and Bandwidth Consideration

To encourage routing in a bandwidth aware manner, the *bandwidth score* is based on values reported by a *bandwidth oracle*. The oracle is assumed to always accurately and correctly report bandwidth. The implementation of this is left for future research. The intuition is that nodes with higher bandwidth are more highly propagated throughout the network, and able to be used for routing by a higher ratio of nodes.

Within each record, a representation of the bandwidth of a node is included, and propagated. It is from this bandwidth that GossipChain weights its node selection process during circuit creation. To select hops for circuit creation, a node $A$ creates a set (i.e. without duplicates) of all nodes in all chains it knows about, with their associated bandwidths. $A$ will then choose in a weighted random way, its hops from within this set. From here, the node will extend the circuit via tunneling. This can be done for any desired circuit length.

Without modification, there is an implicit bias in node selection for nearby nodes. This can result in information leak attacks similar to those against Tarzan. This is because, clearly, a node will receive gossip information of nearby nodes most often. It follows that they have an outsized presence versus more distant nodes within a given node's chains, necessitating a way to give further away nodes a way to make up this difference.

Therefore, there is a second weight to the random selection, which is the distance of an identifier to the selecting node. This means a node is more likely to select, although not guaranteed to select, a node which is 1) distant and 2) high bandwidth (represented via the score). Additionally, it will be shown that the GossipChain algorithm approaches full network knowledge as time approaches infinity.

$$score_{r_i} = f(bandwidth_{r_i}, ID_{r_i}, ID_{node}) = bandwidth_{r_i} * distance(ID_{node}, ID_{r_i}) \quad (4.1)$$

The determination of the pool of selection nodes is simply: $records = \{record | record \in chains\}$ and $pool = set(r | r \in records)$. From there for each $r \in pool$, a score is created given by the formula in Equation 4.1. There are many ways to implement a weighted random choice algorithm, but for the purposes of this thesis and simulation, the `numpy.random.choice` Python module was used, but any appropriate implementation could be used. Once each $score_{r_i} \in scores$ is normalized to between 0 and 1, they can be used as an array of weights. This allows for a weighted selection, with `random.choice(pool, p=scores)`. Whether this selection algorithm works as intended will be analyzed in Chapter 5.

# 5 Evaluation

This chapter presents an evaluation of GossipChain to determine if it is successful in accomplishing the goals set forth at the beginning of this thesis, which will show its fitness for secure and anonymous communication. The structure of the chapter is as follows, the simulation developed to evaluate the solution is introduced first, followed by the series of experiments run, and the resulting characteristics of GossipChain are outlined, which were determined based on the outcomes of the experiments.

The implementation of GuardedGossip was provided, as written by Till Hering. The GossipChain solution re-used code where possible, especially the Chord implementation. The GossipChain modules are written in such a way as to easily interchangeable with GuardedGossip implementation by Hering, so the same experiment routines can be run against both GossipChain and GuardedGossip.

## 5.1 Evaluation Criteria

In general, this solution will be compared against the state of the art mentioned in Chapter 3. As it is an attempt to improve scalability versus Tor, complexity in terms of network overhead (Tor's main limitation) is the main measurement regarding complexity presented. Comparisons will also be drawn to other solutions, such as Octopus DHT, especially in regards to the limitations introduced via its use of a centralized authority. Success will be defined as creating a system which is as anonymous and at least similarly as performant as Tor, Octopus DHT and GuardedGossip. It must also leak less network information during network information discovery, i.e. a node knows only as many nodes with their network internet network location information as to enable anonymous and efficient routing, and to know any more should be difficult. The details of how this is measured is split into various subsections below. Computational complexity is assumed to be low enough to be performant via the running of the simulation.

### 5.1.1 Simulation

The simulation is coded using Python 3.8, in a single threaded application. This was chosen to make the simulation more readily comparable to previously written simulations of other network solutions, namely GuardedGossip, which was also written in Python 3 using a single thread. In general, all reported results were run against a 1000 node network, except where otherwise noted. Tests were performed in a cloud environment, and were afforded a 12-core (24 thread) Intel Xeon processor, with 48 gigabytes of Random Access Memory (RAM). The operating system environment used was Debian 10.

A malicious fraction of $f = .2$ was chosen to mimic the related works. The simulation tracks nodes which are honest and malicious, and defines multiple scenarious under which malicious nodes act. Malicious nodes know all relevant parameters related to bounds checking and witness checking, and can manipulate their chains and finger tables as they see fit, in order to attempt to break the protocol. The simulation is run in discrete measurable steps, with each node running one cycle of the GossipChain algorithm at a time. This mirrors the original simulation of GuardedGossip, to allow for easier step-by-step comparison.

Simulations were each run over 100 times, and the results displayed are the averages. This was done in order to ensure the results are representative. The simulation was crafted in such a way as to prioritize measurement. Ratios of runtime and memory consumption should hold, however in an implemented production environment GossipChain, which would presumably at minimum be multi-threaded there would be an improvement in runtime. Memory consumption of the entire network of 1000 nodes was around 1GB. Runtime per round was heavily dependent upon chain length. With chain length limited to a max of $\log(n)$, with a network size of 100000 nodes each node took less than 1 second, to process a request. Given the independent nature of records in relation to the structure of the chains, in a practical implementation this is highly parallelizable. Therefore runtime would be signifcantly lower.

The steps of initialization follow that of the protocol outlined in Chapter 4. This means that the protocol is run after a full initialized and stabilized Chord circle is provided. In the case of churn where nodes are free to join and leave the network, except nodes were only kicked or joined from/to the network after the stabilization was finished, and measurements were taken. The appropriate modifications were made incorporating the NISAN lookup process, again as stated in Chapter 4, and as inherited from GuardedGossip, described in Chapter 2.

### 5.1.2 Security and Success in the Context of GossipChain

It is fairly intuitive to pin the measurement of how secure the system is to the total unique malicious nodes present in a node's chains when taken all together. In an ideal system, all manipulations of the malicious nodes are detected by the honest nodes, and the fraction of malicious nodes within the chains should lessen over time, as the protocol runs through multiple iterations. This means the vast majority of measurements concerns the details of nodes' collected chains.

#### Simulation Parameters

This section will go over the specific parameters (Table 5.1) of the simulation as it was tested. When possible the parameters are as close as possible to related works, as they reported in their requisite papers, cited in Chapter 2 and 3.

| **Parameter** | **Description** | **Value** |
|---|---|---|
| $m$ | Number of bits in an identifier | 32 |
| $N$ | Chord circle maximum size | $2^m$ |
| $n$ | Number of nodes in simulation | $10^3$ |
| $f$ | Fraction of malicious nodes | 0.2 |
| $b$ | Bandwidth score range | $1 \dots 10$ |
| $chain_{max}$ | Max chain length | $\log_2 N$ |
| $n_{chain}$ | Number of Chains sent per round | $1 \dots 3$ |
| $n_{new}$ | Number of new fingers added to each chain | 1 |
| $\gamma$ | Bounds checking threshold value | $\sqrt{\frac{1}{f}}$ |
| $c_s$ | Number of chains spot checked per round | 1 |
| $r_s$ | Number of records spot checked per chain in $c_s$ | $1 \dots 3$ |
| $v$ | Chance of chain removal and new chain creation during GossipChain round | $\frac{1}{4}$ |

Table 5.1: Setup Parameters for the GossipChain Simulation

Care had to be taken to ensure meaningful results yet maintain an agile simulation which could be run many times testing different parameters. Running the heavily instrumented simulation is processor intensive, particularly in the initial stabilization phase, or when running the cryptographic operations. In a real world implementation, where this is run using parallelization over multiple threads (or even machines,) this would not be arduous. However, in a single threaded application, it can add up. Therefore, the network size was chosen to

be 1000 nodes, and the identifier space was chosen as 32-bit. Besides these, there are many other adjustable parameters which had to be set to craft the simulation. The fraction of colluding nodes was set to $f = 0.2$, which is the upper limit of what related works had set. This was chosen to run the network in a worst-case scenario situation. The bandwidth score of each node is set randomly on node creation, and is between 1 (least bandwidth) and 10 (most bandwidth). When requesting GossipChain information from nodes, up to 3 chains are sent, each of which could have 1 ($n_{new}$) new finger added to it. Each chain is limited to a length of $\log_2 N$, called $chain_{max}$, somewhat arbitrarily, but related to the ability to theoretically route to all other nodes on the circle within $\log_2 N$ hops, and nodes on the chain would probably begin to repeat. There is a chance, before the responding node sends a list of chain hashes (Section 4.4.1 step 2), that old or low bandwidth chains are discarded, if the total number of chains are greater than 10, and new chains are created. This chance is $\frac{1}{4}$. Each of these parameters was considered and then tested as given in the table. For the bounds checking threshold, $\gamma$ and its value of $\sqrt{\frac{1}{f}}$, see the foundational work in NISAN by Panchenko et al. [27].

### Ideal Values of Characteristics of GossipChain

It is intuitive to see it is advantageous to *minimize* the fraction of malicious nodes within any given node $A's$ chains. More simply the total knowledge of all nodes available for circuit creation (from the perspective of $A$) should have as few malicious nodes as possible (i.e. the fraction approaches 0 in successive GossipChain rounds). This is practically impossible, however it is the *ideal*. This should be done while attempting to *maximize* the diversity of nodes available for selection for circuit creation, and ensuring it is a varied and diverse list via entropy.

These optimizations should occur within a framework allowing for a solution to the other goals previously established which allow for a scalable and useable network. Therefore, we desire a network that is *resilient to high churn rates*, *complete*, *responsive*, running with *low*

| Variable | Description | Ideal Value |
|---|---|---|
| $chain_{ent}$ | Entropy of nodes in chains | $max(H) = \log_2 N$ |
| $frac_{mal}$ | Fraction of malicious nodes in chains | 0 |

Table 5.2: Optimization goals of GossipChain

*bandwidth*, *quickly stabilized* and ready to function. It is considered acceptable to sacrifice some of the aforementioned optimization in order to exist within these constraints.

## 5.2 Experiments

This section will explain the experiments and how they are set up to perform these optimizations while maintaining the constraints mentioned in Section 5.1.2. These experiments can talk to how the system operates within the framework defined above, which should act as a proxy for real-world performance. After the first set of experiments, there is an exploration of potential possible passive active attacker scenarios under which GossipChain might operate. It is here that the ability of GossipChain to withstand passive and active attacks is discussed.

### 5.2.1 Completeness and Connectivity

As stated in Section 5.1.2, we desire a network which is complete, allowing access to any node as required. To reduce bias in used nodes and prevent attacks in the class of Range Estimation Attacks, the pool of nodes available for circuit creation should be large enough to be considered uniform and unbiased. To prevent network censorship, simultaneously the pool of nodes should be limited to prevent node enumeration.

To test the completeness of the discovery process; i.e. how much of the network is accessible at any time, and/or selectable for circuit creation, a history of nodes kept within chains was kept. This was allowed even as the chain was removed. Against all sizes of networks that were simulated, the connectivity of the network always approached a level sufficient for the above stated goal. Against a network of 1000 nodes, it took roughly 150 rounds. As can be seen in Figure 5.1, the network rather quickly reaches a high level of discovery, determined by how quickly nodes had seen or accessed 95% of the network.

It was expected that the chains would be able to efficiently spread information, as each subsequent round the chains grow and send more information. Therefore it is as intended that initial rounds exhibit slow growth, followed by a large and sudden increase of seen nodes.

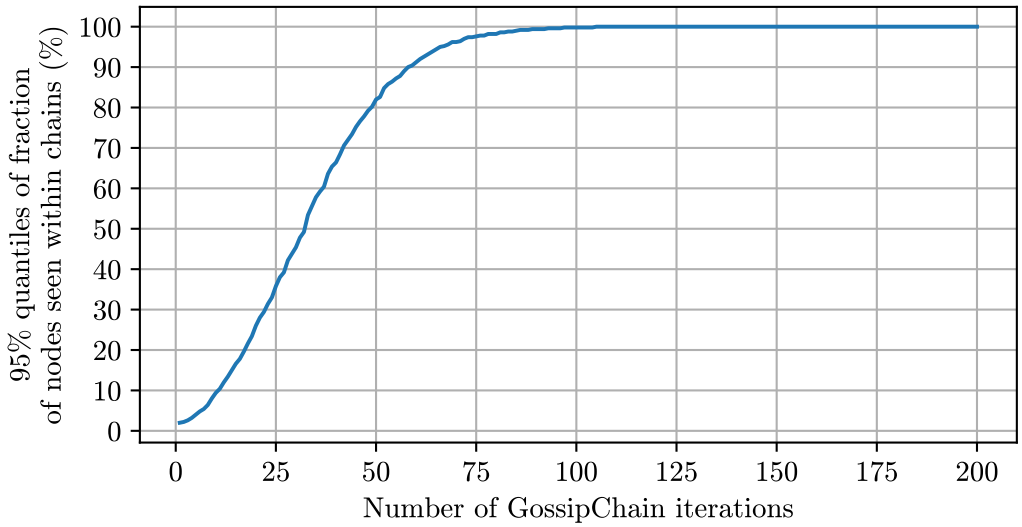


Figure 5.1: GossipChain Connectivity

### 5.2.2 Entropy and Randomness

To prevent predictability of nodes within the chains, it is necessary that the collected list of chains have a high level of uncertainty. To measure the level of uncertainty, the entropy was measured. Initially there is a brief explanation of entropy from a theoretical perspective, followed by a practical exploration. The results of the practical exploration are displayed in Figure 5.2.

Entropy is a measure of all possible outcomes of a random variable. To do this we need to obtain two scores, a maximum entropy and a score of uncertainty for the random variable itself. Shannon's original definition of entropy from an information theory perspective gave the formula in Equation (5.1) to determine the entropy of a random variable [57]. For the purposes of this thesis, the base is $b = 2$, and $n$ is the number of nodes in the network. It follows that the max entropy will be given by Equation (5.2)

$$H = -\sum_{i=1}^{n} p_i \log_b(p_i) \qquad (5.1) \qquad max(H) = \log_2 n \qquad (5.2)$$

Next, in order to solve for $H$ in (5.1), different outcomes/events and their probabilities have to be defined. For this there are two pools of nodes with vastly different behaviors. Honest nodes which attempt answer GossipChain requests truthfully and malicious nodes which answer falsely. The probabilities within GossipChain that a malicious node is selected will

correlate directly with the fraction of malicious nodes, $f$. The fractions of malicious and honest nodes are given by Equation (5.3) and (5.4), respectively. Recall that $m$, from Table 5.1, is the bit length of the identifier, which defines the size of the finger table.

$$n_{mal} = f \cdot m \qquad (5.3) \qquad\qquad n_{hon} = (1 - f) \cdot m \qquad (5.4)$$

These would be the only equations that were required, if finger tables were accepted blindly– however, they are not. Due to the bounds checking and witness checking the actual acceptance rate is influenced by a false negative rate $\beta$ and a false positive rate $\alpha$. This gives the new Equations 5.5 and 5.6, defining the events $p_{mal}$ and $p_{hon}$. Inherited from GuardedGossip are the same bounds and witness list checks, and as was done in that work, GossipChain also attempts to minimize both $\alpha$ and $\beta$, and can assume $\alpha = \beta$. More information on this can be found in the original work on which GuardedGossip is based [58].

$$p(n_{mal}) = p_{mal} = \beta\frac{1}{n_{mal}} \qquad (5.5) \qquad p(n_{hon}) = p_{hon} = (1 - \alpha)\frac{1}{n_{hon}} \qquad (5.6)$$

Finally it is known that $(n_{mal}p_{mal} + n_{hon}p_{hon}) = 1$. As this accounts for all possibilities. Since there is a new node subject to the same checks each round, it also follows that any given chain can be simplified to a list of nodes reported in records which looks as follows: $\{p_{mal}, ..., p_{hon}, ...\}$. Now that the probabilistic events are defined, work can begin on filling in the entropy equation shown by Equation (5.1). The entropic nature then of the nodes chosen in the first round is:

$$H_1 = -\Big(n_{mal}p_{mal}\log(p_{mal}) + n_{hon}p_{hon}\log(p_{hon})\Big) \qquad (5.7)$$

And each subsequent round is dependent on the previously received chains. From here a practical analysis was done in the crafted simulation. Again, the simulation was modified to track chain entries over time (i.e. without the $v$ removal chance). Further, in each round all possible new Records were considered, and the entropy then calculated as above. The results are shown in Figure 5.2.

As expected, GossipChain never reaches the maximum entropy level given by $\log_2(n); n = 1000$. However, versus the ancestor solution of GuardedGossip, it follows an extremely similar trend. GossipChain takes longer to reach a stable level of acceptable entropy than GuardedGossip.

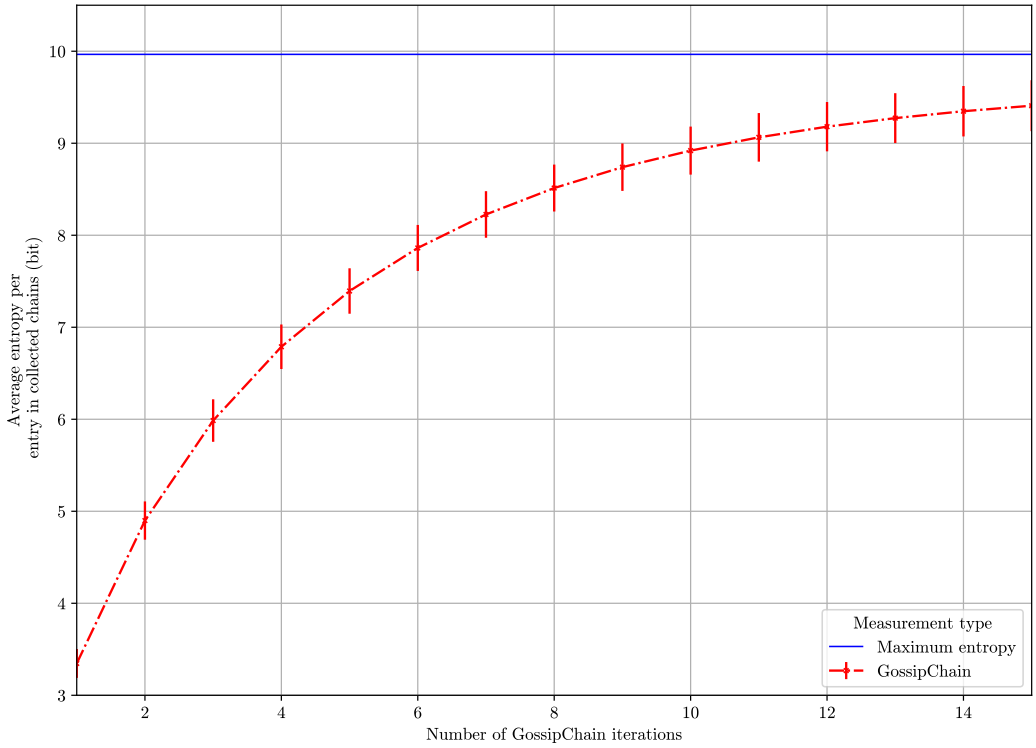


Figure 5.2: Practical Exploration of Chain Entropy with $n = 1000$

The most likely explanation for this is that in the early stages it is more likely to receive nodes less-distant to the requesting node. Later, more varied information is sent containing larger chains with distant nodes. The composite graph comparing the two solutions is presented in Figure 5.3.

As the amount of rounds considered are relatively few, this is considered acceptable as a similar level of entropy is reached. To provide additional assurance that the level of randomness provided by GossipChain is acceptable to provide defenses against passive attacks, a $\chi^2$ goodness-of-fit test was also performed. In simple terms this test allows us to test whether the null-hypothesis is likely true that a random distribution was selected uniformly. This is done through a comparison of the distribution with a uniform one [59].

Given a set of observed and expected outcome frequencies defined as $O = (o_1, \ldots, o_n); E = (e_1, \ldots, e_n)$, we can determine the $\chi^2$ statistic as:

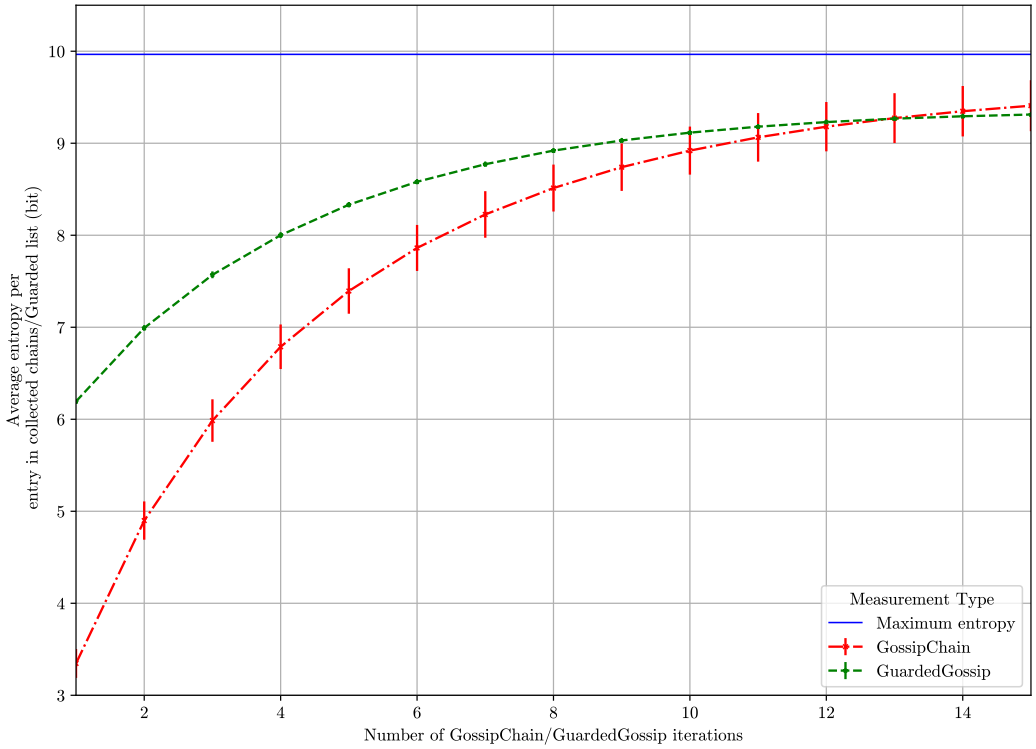


Figure 5.3: Comparison of GossipChain and GuardedGossip Entropy with $n = 1000$

$$\chi^2 = \sum_{j=1}^{n} \frac{(o_j - e_j)^2}{e_j} \tag{5.8}$$

Also necessary is a statistical significance level, taken historically as $a = 0.05$, and the degrees of freedom $df$, which is one less than the total number of possible classes. This gives $df = n-1$. To prove the null hypothesis the $\chi^2$ value should be less than the *chi-squared critical value*. For the purposes of this thesis, this value was determined from the SciPy toolkit[1] in python, via the library `sciypy.stats.chi2.ppf(q=0.05, df=9999)`, with $n = 10000$ the critical value was 10232.

To perform the test networks of $n \in \{1000, 5000, 10000\}$ nodes were created. After network stabilization, a collection of nodes were sampled, and the observation set $O$ created after $y = .95n$ *new* nodes were discovered. $E$ is assumed to be the uniform distribution, and so

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2.html

$e_j = \frac{y}{n}$. After series of tests, the resulting $\chi^2$ statistics were in the range from 9000 to 10000, although a few instances were above the critical value. The passing rate was, however, greater than 95%, and so this test is considered successful. Meaning, it is acceptable to assume uniform distributions of nodes in chains.

### 5.2.3 Complexity

It is important to judge the theoretical complexity of GossipChain and compare it to other solutions in the related works, most notably that of GuardedGossip on which GossipChain is based and Tor as the most popular solution. The goal is to prove the scalability of the network. If the complexity is too high, the network is rendered unusable.

The complexity presented here is an estimation assuming a network of 50000 nodes for each network, with the same assumed network load. In the case of GossipChain, when there was a variable which changes in size over time, like the size of the transmitted chain, it was assumed to be the worst case. In this case, that means it is assumed that each node is sending a full-length chain of $\log(n)$ records. Again, with $\log(n)$ records, theoretically, all nodes should be routable so a chain longer than that would not be necessary.

At the scale necessary to show all of the relevant related works in the graph of Figure 5.4, the overhead differences between GuardedGossip and GossipChain are shown as negligible. GossipChain has a slightly higher bootstrapping load. This is expected, as GossipChain adds checks which are only processor intensive to the client, but do not involve significant additional network overhead. In fact, the *lineage check* from Section 4.3.6, requires no additional network overhead. The increase can come from the increased size chains in later rounds of network operation.

Versus the other solutions, GossipChain represents a network overhead improvement similar to GuardedGossip. Only Octopus [16] shows lower network overhead, however it requires a centralized authority in its operation. The mild increase in overhead compared to GuardedGossip is expected, because in later rounds the chains grow larger than the total number of finger tables GuardedGossip transmits. The difference however is not exceedingly large, and should be considered in conjunction with the potential improvements in malicious node detection shown later on in this chapter.

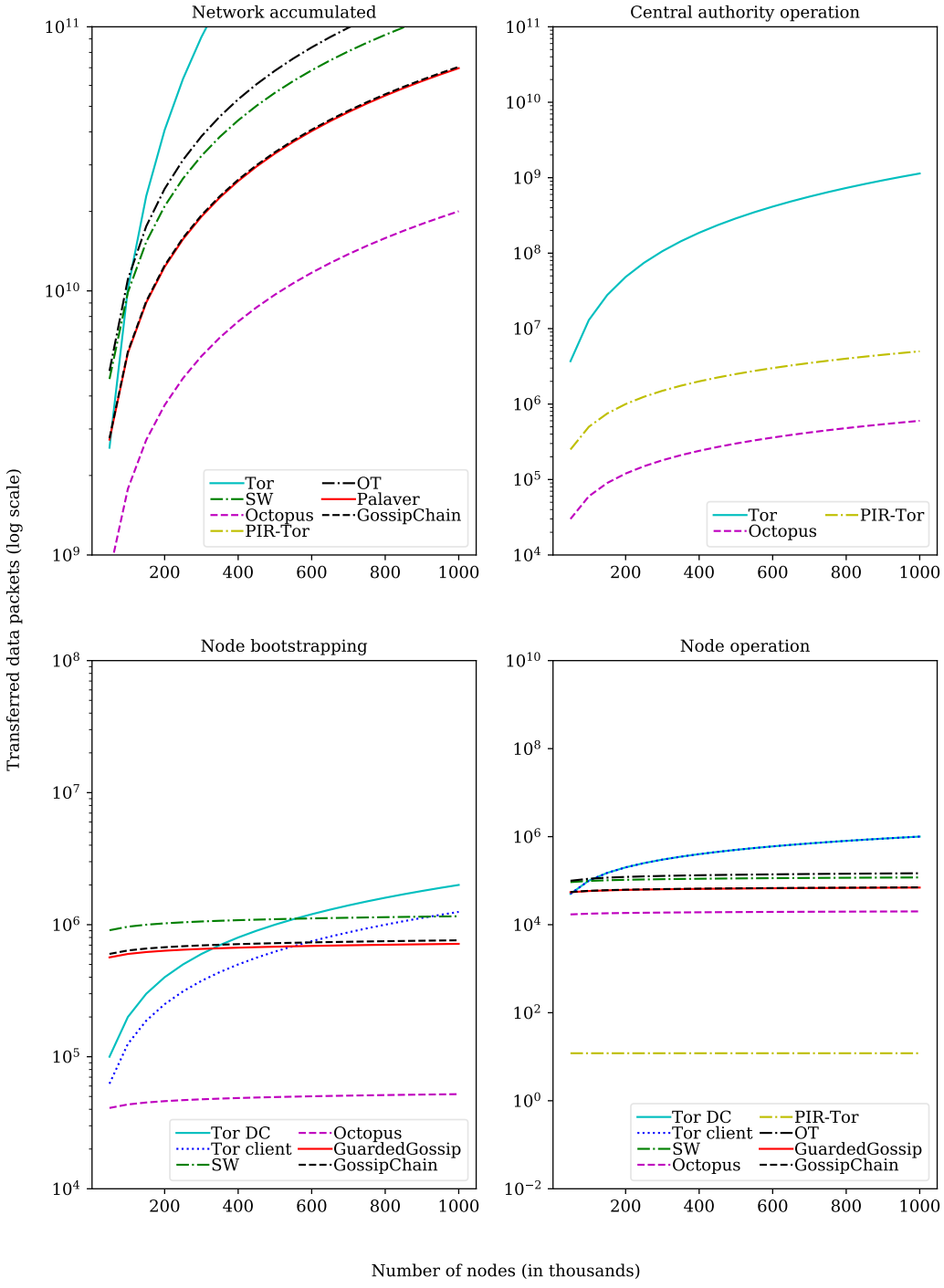


Figure 5.4: Comparison of computed complexities of GossipChain and Related Works

### 5.2.4 Churn

Churn is one of the most difficult realities of a P2P network. To figure out the influence of churn on nodes' chains, the network was run over 400 iterations at various rates of up to 1% of nodes leaving and joining the network per round. As stated earlier, the malicious fraction was maintained at $f = 0.2$. All network parameters remain unchanged from those that exist in Table 5.1.

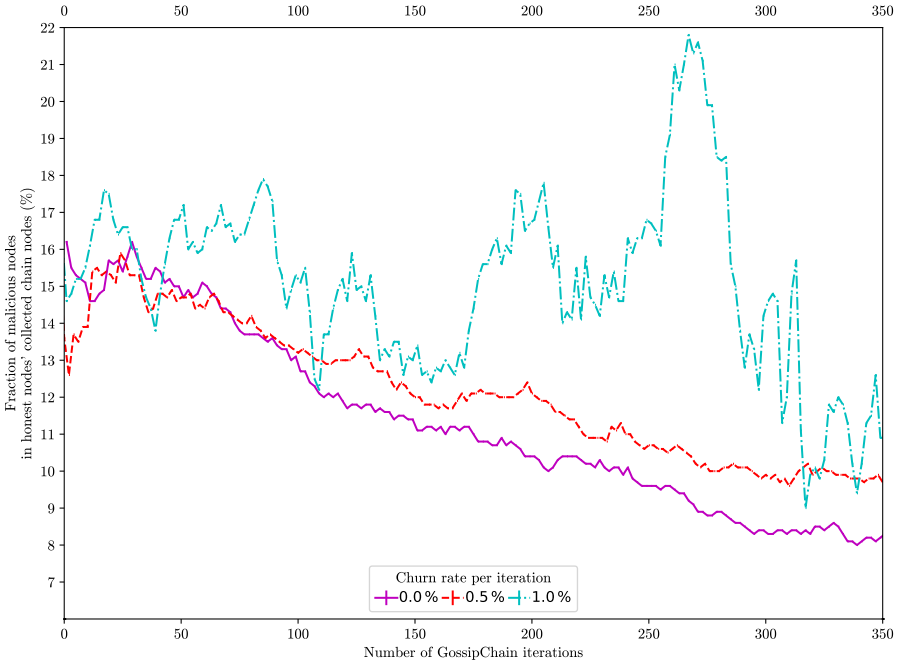


Figure 5.5: Effect of churn on GossipChain

The graphs below in Figure 5.5 and Figure 5.6 show the effect of churn rates on both GossipChain and GuardedGossip respectively. Higher rates of churn are, as expected, more damaging generally to both of these protocols. The initial regarding GossipChain at high rates of churn, was that it would result in a malicious fraction rate in the chain lists relatively close to $f$, because of how the network performs bootstrapping of nodes. At a rate of around 1.0% it can be seen that GossipChain behaves rather more unpredictably than expected. This is likely a result of how nodes create new chains from each of their fingers on network join. This means that the amount of collected nodes from their chains has the same malicious fraction as their finger table exactly. This is in contrast to GuardedGossip which performs more processing on the nodes which wind up in its guarded lists.

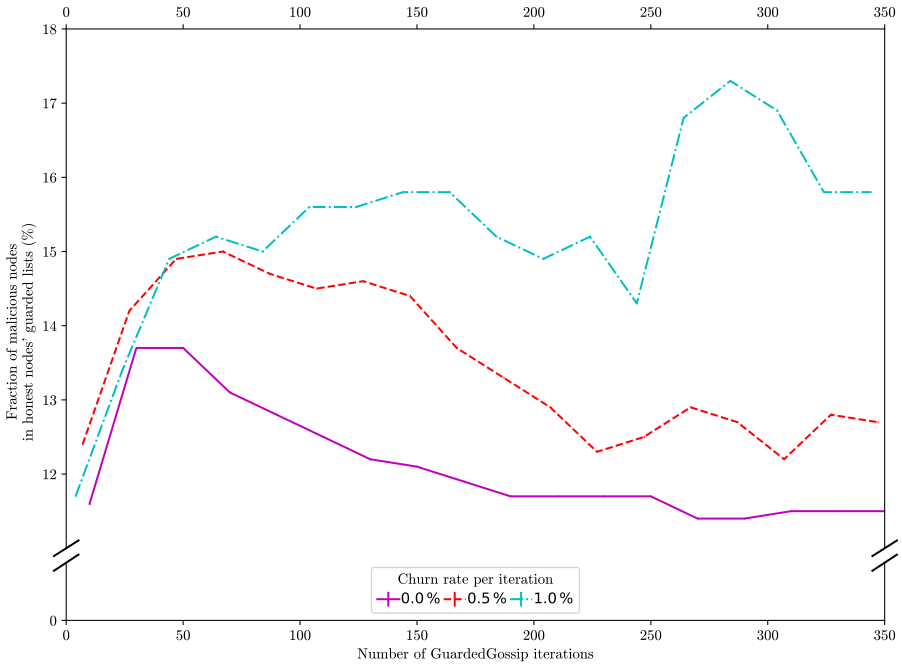


Figure 5.6: Effect of churn on GuardedGossip

A small modification could be made to perform the bounds and witness list checking in this step, however this will result in each node having less chains to propagate through the network in the early stages. As crafted in the simulation, it was considered better to allow the detection to occur later, as more information was available to prevent a case where a node is repeatedly sending the same chain out of *necessity*. It is clear however, that the network is capable of handling rates of churn, and the detection of malicious information still works as expected.

### 5.2.5 Bandwidth-aware Circuit Building

In order to encourage efficient routing through the network, GossipChain is designed to be bandwidth aware, via the bandwidth score. Although this thesis is not concerned with how these scores are granted, it was considered advantageous to be able to incorporate some form of bandwidth awareness to ensure the network runs efficiently.

To test the ability of the network to account for the bandwidths of the nodes, each node was provided a random bandwidth $score \in [1 \dots 10]$. A network of 10000 nodes was created, and

run until stabilized, using the information gained from the previous sections. Next, 10000 three-hop routes were created according to the selection procedure defined in Section 4.4.4. each node was then categorized into a bucket according to its bandwidth, and a percentage of the total network containing nodes of each bandwidth was calculated. Then, a tally was done for each circuit concerning which bandwidths the nodes within the circuit had. The results of which are given in Table 5.3.

| **Bandwidth Score** | **Percentage of Network** | **Percentage of Circuits** |
|---|---|---|
| 1 | 11 | 2.7 |
| 2 | 10 | 3.6 |
| 3 | 12.5 | 5.3 |
| 4 | 8 | 7 |
| 5 | 9.5 | 7.4 |
| 6 | 12 | 9.4 |
| 7 | 12.5 | 18 |
| 8 | 9 | 13 |
| 9 | 10.5 | 13 |
| 10 | 10.5 | 21.5 |

Table 5.3: Bandwidth statistics from circuit-routing through a stable network

In a network where all nodes are counted "equal", each node should route an equal amount of traffic. Table 5.3 refers to the selection statistics by bandwidth of 100000 circuits, each circuit consisting of 3 nodes, $n_1, n_2, n_3$. In general, if the nodes were to be grouped into ten categories with equal probabilities, one would expect that each category would have an equal network and route selection percentage (i.e. column 2 and 3)[2]. As GossipChain weights route selection by the bandwidth score, lower bandwidths should have lower route utilization versus their network footprint, and as the score increases the trend should diverge further and further. As shown in Table 5.3, nodes with a bandwidth score of 10 account for only 10.5% of the network, but are selected for 21.5% of routes. It should be understood that since the distance of the nodes is also taken into account, and it is a weighted random function, the process is inherently stochastic, and we can only measure the trend over a significant number of circuits.

[2]Note that numbers are rounded to one decimal place, and so the percentages do not equal 100%

### 5.2.6 Active Attack Scenarios

In attempting to understand how the network operates, and its characteristics and behavior, several scenarios were run under alternating conditions. Of course, the initial simulation involved a network with blatantly malicious nodes accounting for $f$ of all finger tables. Various scenarios were also considered to gauge potential weak points to a network like GossipChain: first, those who drop chains in order to propagate only malicious chains, thereby attempting to increase the malicious fraction of nodes in honest nodes' chains; next, if a malicious member joins the network only to deny service when requested; third, the ability of the network to be robust against those who act honest in some phases, but later turn to dishonesty regarding some aspects of the protocol; and finally, if a node could arbitrarily choose its position in the network.

#### GossipChain in a "Normal" Environment

During the simulation various parameters and methods were tuned to craft GossipChain into a solution which is secure, and practically usable. It is useful at first to see how GossipChain acts in an environment where malicious nodes would only attempt to collect information passively, and perform no active manipulations. As expected, in this "normal" environment, the fraction of malicious nodes remains around $f = 0.2$. This is expected as none of the present checks can detect malicious behavior. Furthermore it demonstrates the uniform randomness of the GossipChain procedure, although as seen in smaller networks it can be more likely that the fraction lowers slightly.

Initial expectations were that in a environment where the only manipulations are based on characteristics of chord, the fraction of nodes within the collected chains which are malicious were to be lower than $f$. As can be seen in Figure 5.8, there is a steady downward trend acting asymptotically to a percentage lower than the expected 20%, and in smaller networks to around 7% fraction of malicious nodes. This is because within the simulation, it rather aggressively performs the spot checks from Section 4.3.7. The spot check is performed both by the receiving and sending node. If the sending node first discovers the chain to be malicious, it will simply not propagate it. To see this in action, Figure 5.9 shows the malicious fractions in a network which does not perform the spot check, while malicious nodes perform finger table manipulations.

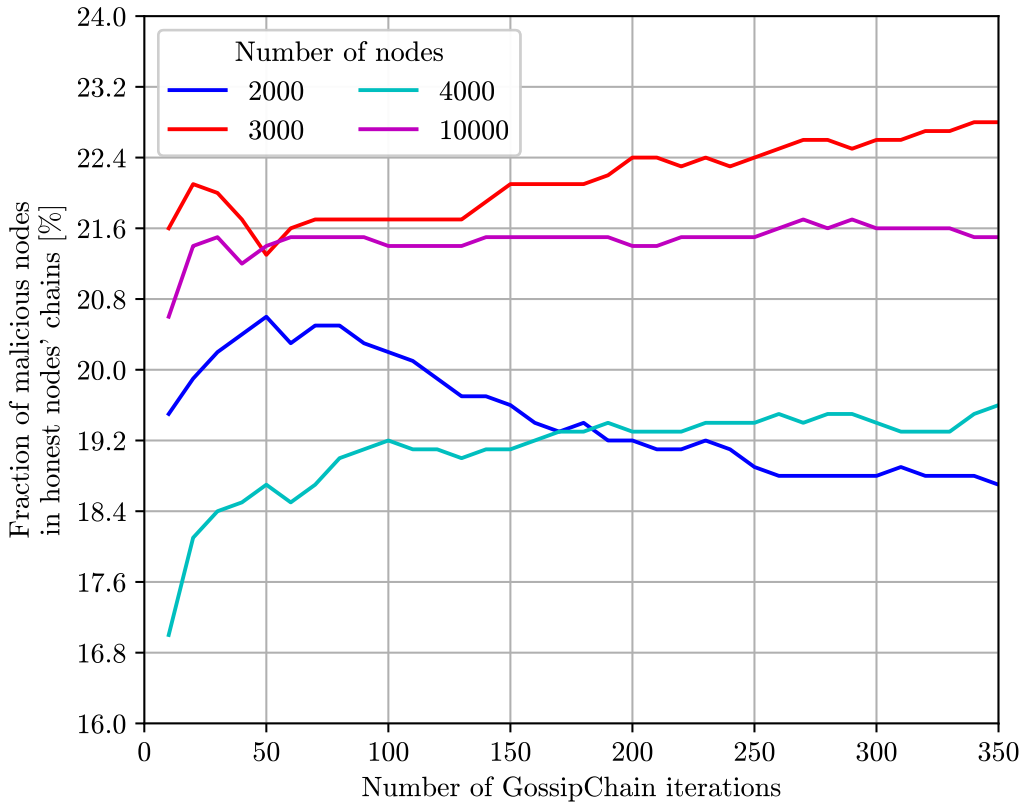


Figure 5.7: Malicious node perform no attack scenarios, and act honestly

Since there is no real penalty for refusing to answer the protocol (the effects of which are described below,) this is actually fine. If the sending node discovers it is holding a malicious chain, it marks it as "not to be propagated", and also marks the node which failed the spot check as "potentially malicious". It removes the failing node from consideration for circuit building, but maintains the ability to use the remaining nodes from the chain. The reasons for this are two-fold: It is not clear if the node was acting honestly on creation-time of the record, or not, and it would minimize the pool of available nodes too much. As a future work, these checks could be further developed allowing for more thorough analysis of a chain's point of failure.

### Dropped Chains

The first scenario evaluated, shown in Figure 5.10, is when an attacker attempts to drop majority honest chains and/or propagate only malicious nodes within its chains. As can be

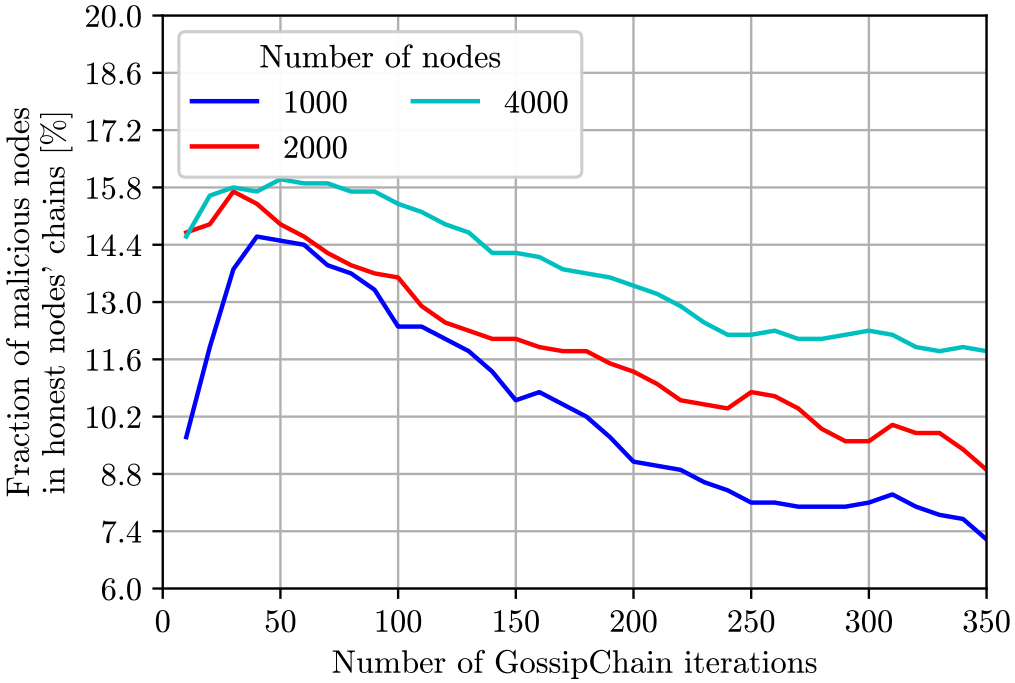


Figure 5.8: Malicious nodes perform finger table manipulations only

seen in the figure, this is generally detected showing a minimal effect on the network when compared to Figure 5.8 with only finger table manipulations.

This is due mostly to the *lineage check*, described in Section 4.3.6. The lineage check enforces that each node that follows in a chain is a plausible finger of the one before it. Since the malicious nodes are spread throughout the network, and arbitrary positioning within the Chord circle is handled via other methods as described and then verified in Sections 4.3.1 and 5.2.6. This means to have a fully malicious chain accepted by another node, there must be a set of nodes with the finger relationship which collude and are malicious. This is naturally not impossible, but as shown here it is in a practical sense unlikely.

### Denying the Protocol

Next, it was important to determine the susceptibility to denial of service if a node simply refuses to answer the protocol, attempting to starve the network of gossip information. There are few real protections against this in GossipChain, only the expectation that for a node not to be considered malicious, it must respond to the protocol at minimum with a *new* chain.

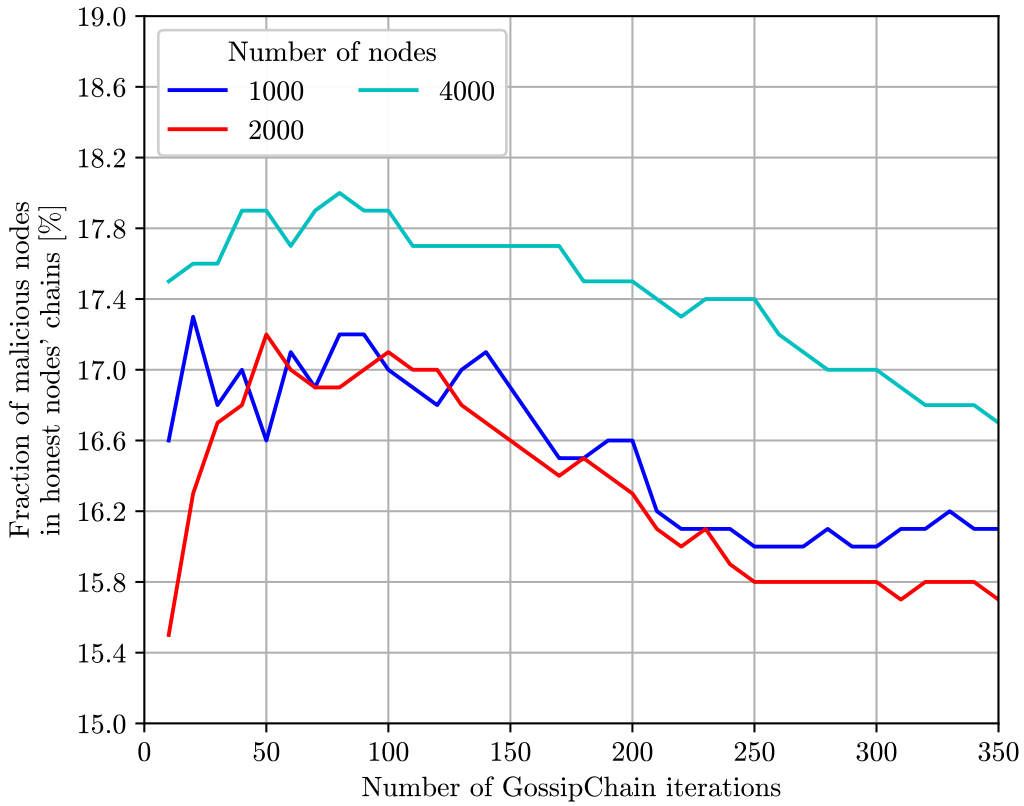


Figure 5.9: Malicious Fraction of Nodes without running the Spot Check

It is expected that this behavior is possible, in any case. However, it should not be advantageous to an attacker, unless the malicious fraction is far above $f = 0.2$. This is because more would be gained from attempting to respond with tainted information. Since nodes have no incentive to block and wait for information, and only to ask again in the next round from a different node. Within the simulation, nodes are equally likely in each round to forward information from any finger, and therefore at most a malicious actor can hope to stymie one round of information transfer for any node which has them as a finger, and which happened to choose them as a gossip source.

This prediction is borne out by the graph in 5.11. Again, this behavior affects the network little. This is even without counting those who refuse to respond to requests as inherently malicious. Should an attacker also refuse to send information in that step throughout the whole protocol, their chains would naturally "age-out" of consideration as more vocal nodes send information. And therefore the malicious intent does not even need to be detected,

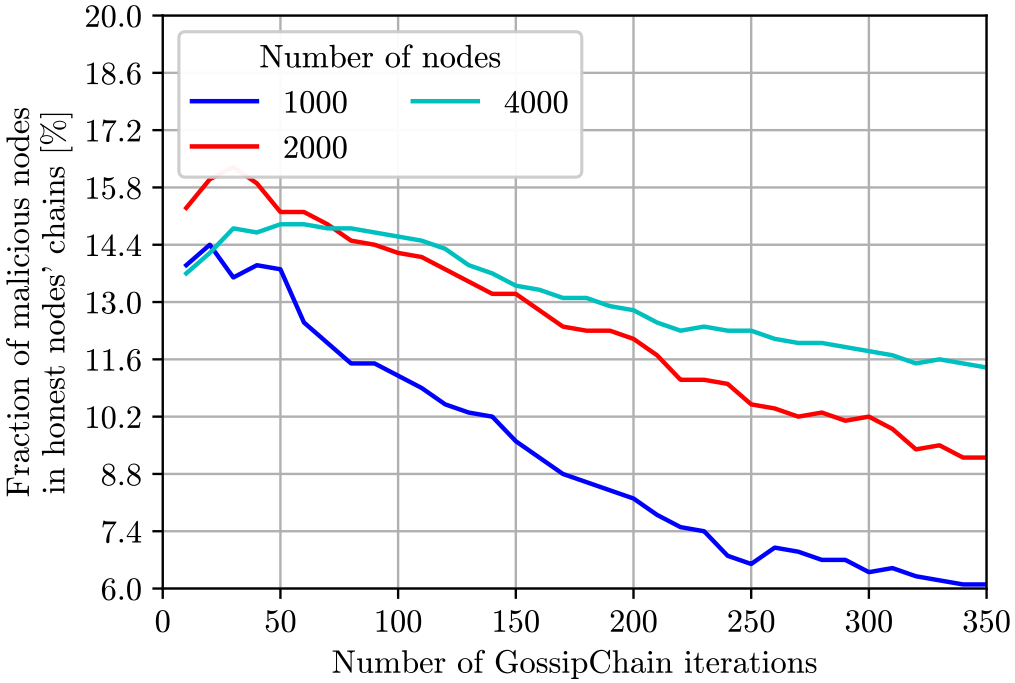


Figure 5.10: Malicious nodes drop majority-honest chains

but instead will cease to matter in the natural course of running the network. This is also a potential reason this chart shows a quicker decline in the malicious percentage of nodes within the chains more quickly than the others for the 4000 node network.

#### Delayed Dishonesty

Another issue, one which the *spot check*, is intended to solve is what were to occur if a node begins the protocol honestly, in order to obtain presence in many chains, and then begins to propagate malicious information, or modify collected chains. To measure this, each malicious node was set to turn malicious after 25 rounds. The expectation is that when a node turns malicious it would be detected by the *spot check*. The graph should show a steep drop off around the fiftieth round, this is done as a further verification that the procedure allows for the detection of malicious nodes.

Here a few decisions on how to react to such behavior had to be made. Chiefly, when a node is discovered as dishonest, what should the honest node do with the now semi-invalid chain? An initial expectation was that as the spot check marked chains as unavailable for propagation,

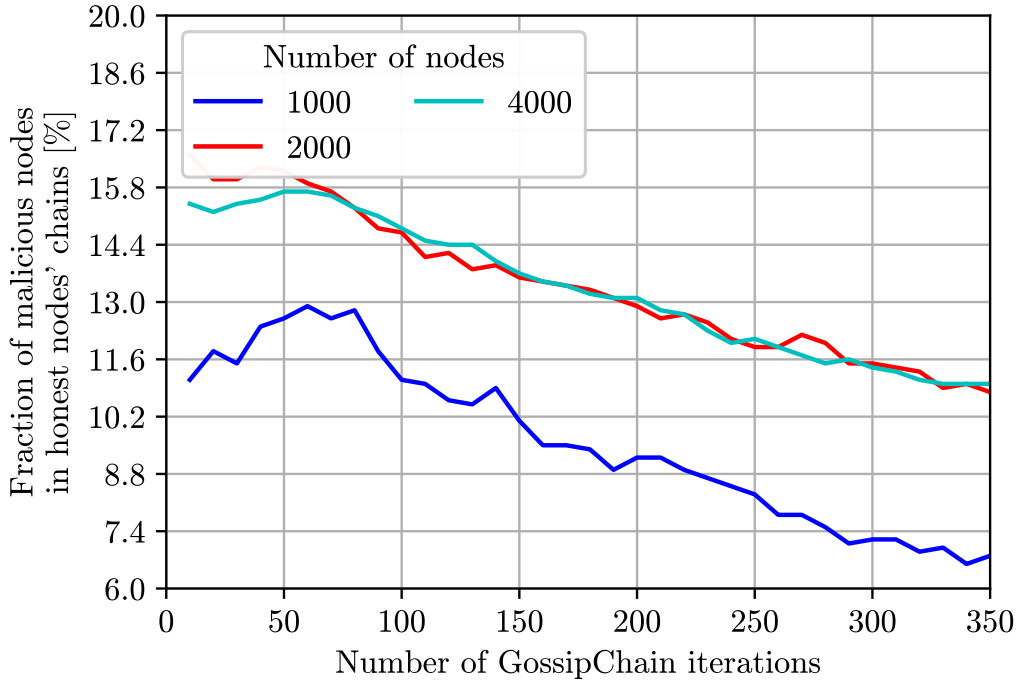


Figure 5.11: Malicious nodes refuse to propagate chains

that the whole chain became untrustworthy. However, to do this would unnecessarily limit the amount of nodes available for circuit creation in a later stage. Therefore, it was decided that although a node will no longer gossip that chain to others, it will keep the chain until it naturally ages out or is removed probabilistically during the gossiping phase. However, it keeps track of the nodes which failed the spot check and removes them from consideration in circuit creation.

As the spot check, described in Section 4.3.7, adds uncertainty to when a node can be checked regarding the protocol, and even could be done through tunneling to hide the origins of the requester, this sort of behavior is risky for an attacker wishing to remain undetected. In Figure 5.12 we see, again, the impact of this sort of behavior is minimal. As nodes begin to propagate false information, either through the bounds and witness list checking, or via the spot check. It should be expected however, that over a longer period of time and runs, the graph shown in the figure should have a higher average malicious fraction than those active scenarios referenced earlier in the earlier subsections of this section (i.e. 5.2.6). Here it is most likely again the stochastic nature of the various networks that it is statistically similar.

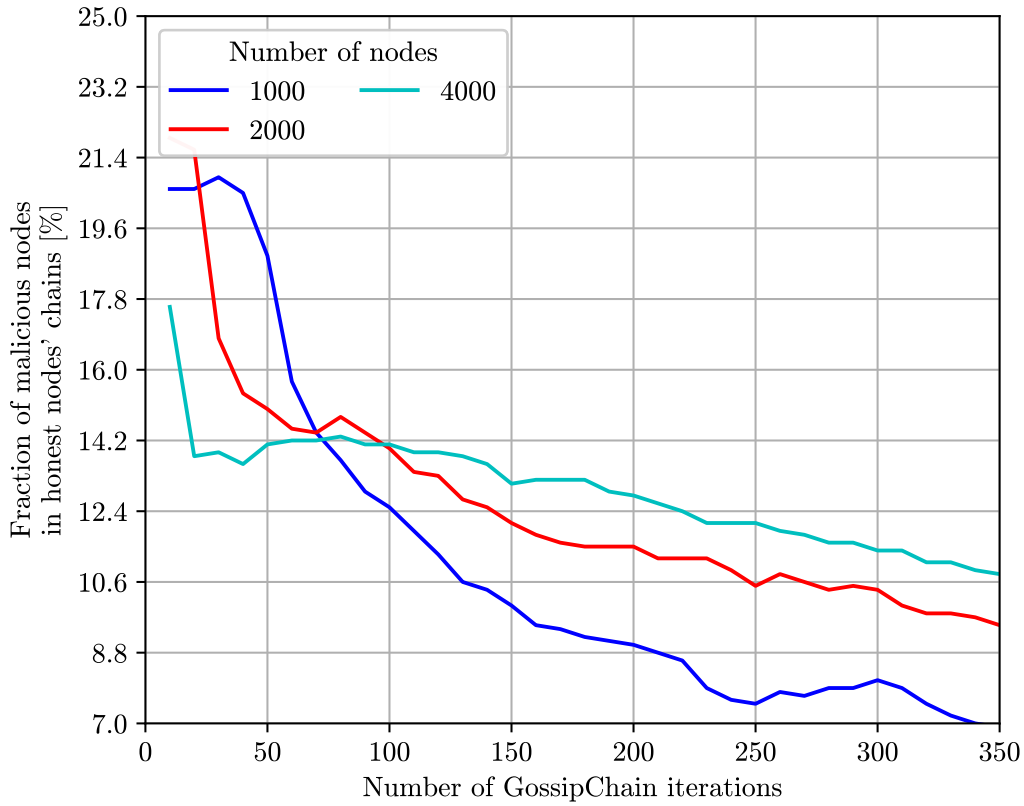


Figure 5.12: Malicious nodes begin honestly, attempting to get into as many chains as possible, then act dishonestly

### Manipulation of Chord-circle Location

The process of identifier assignment described in Section 4.3.1, occurs during bootstrapping. As all in-network nodes have out of band information about the time of the sample of the public source of information, a bootstrapping node can verify the creation of another nodes identifier. For the simulation, this was simple XOR operation of an IP address-like structure, and this public random source. This means, that when an attacker attempted to join the network with a falsified ID, which did not match the IP address connecting to its bootstrapping node, it was rejected from joining the network. Therefore it can be seen that the network would only be susceptible to manipulation of the chord circle location if the pool of bootstrapping nodes were colluding and also malicious.

# 6 Conclusion

With the ever prevalent nature of both interconnected systems and surveillance of those systems, privacy and security are increasingly at a premium. These two overarching goals have precipitated the creation of various anonymous communication networks. All of these networks have sought to provide methods to discover intermediary relays through which to route traffic, while obfuscating this information from outside observers.

The largest network providing this capability worldwide is Tor [11], which does it in a centralized manner. The use of centralization can present a series of problems. First, centralized servers can create bottlenecks causing issues with scalability. They can also present uniquely attractive targets for attack. Especially considering they necessarily know the entire network state. Similarly, decentralized solutions cannot be done in a naive way, because they are constituted by a series of untrustworthy network participants.

Related works, gone over in Chapter 3, have attempted to solve this problem in various ways. Some improved on Tor's centralized approach by adding efficiencies to the discovery protocol. Others crafted decentralized solutions, which although protecting against one issue of centralized approaches, were found lacking for various reasons. Either through network overhead, required centralized components, or easy network participant enumeration.

GossipChain represents an extensible protocol which allows users to obtain network information and communicate in a way which is secure and anonymous. It builds upon work pioneered by Chord [24], GuardedGossip [14] and NISAN [27], but incorporates or implements their innovations in a novel way through trust chains. The use of trust chains to form a *lineage* of network relationships allows a more thorough checking of malicious behavior. The effectiveness and efficiency of this design choice was demonstrated in 5.

It was demonstrated that GossipChain has an acceptable level of complexity regarding network overhead, lower than many related works, most importantly Tor. It also exhibits an acceptable level of security against a fairly malicious environment with an attack fraction of $f = 20\%$, a fraction that meets or betters the related works. The networks run were large

enough to imply scalability, and it showed resilience against expected network actions like churn. Via measurements of entropy and through the $\chi^2$ metric, it was shown that node selection is roughly uniform, and resistent to passive attacks. Furthermore, the introduction of a bandwidth statistic while introducing minimal additional network overhead allowed for routing in a bandwidth aware manner, and encourages a load-balanced network.

Future work turning it into a usable network for practical purposes would only result in efficiency gains versus the simulation. Most interestingly, it was showed that introducing the novel *spot check* and *lineage check* allows for the crafting of a node selection pool for circuit creation with a malicious fraction around that of $f$. Even better, during the scenarios when those checks were made more aggressive, it allowed for a lower fraction than $f$, although the degree to which it was lowered was heavily dependent on network size.

In short, GossipChain shows the ability of trust chains to improve the security of network information discovery protocols in a scalable manner. It provides a promising base on which to extend various checks–namely the spot check and lineage check, which already provide improved effectiveness and security in the face of malicious nodes, on top of previous works' bounds and witness list checking.

# A Witness List Check Algorithm

```
    def witnessListCheck(node, finger_table):
        """Performs Witness List check on finger table"""

        # Decide randomly if mismatching witnesses will be checked
        check_failed_witness_ids = check_follow_through_probability

        passed = True
        failed_witness_ids = []
        for i, presented_finger_id in finger_table:
            # Get best finger for this entry from our witness list
            witness_finger_id = better_finger in witness_list or
                presented_finger_id

            if not witness_finger_id == presented_finger_id:

                # Check if the closest Witness List entry is closer
                    to the ideal finger, than the presented one
                witness_distance = witness_finger_id - node.
                    ideal_finger_ids[i]
                if witness_distance < 0:
                    witness_distance += size_id_space

                presented_distance = presented_finger_id - node.
                    ideal_finger_ids[i]
                if presented_distance < 0:
                    presented_distance += size_id_space

                if witness_distance < presented_distance:
                    passed = False
                    if check_failed_witness_ids:
                        failed_witness_ids.append(witness_finger_id)
                    else:
                        break

        # Check if witness node is actually alive
        for witness_node in failed_witness_ids:
            if alive(witness_node):
                passed = False

        return passed
```

Listing 1: Witness List Check Algorithm

# B Bounds Check Algorithm

```
1     def boundsCheck(self, node, finger_table):
2         """Performs Bounds Check on finger table"""
3
4         # Use global value for bounds checking thresholld
5         gamma = threshold
6
7         # Compute finger table deviation
8         distance_sum = 0
9         for i, finger_id in enumerate(finger_table):
10
11            distance = finger_id - node.ideal_finger_ids[i]
12            if distance < 0:
13                distance += size_id_space
14
15            distance_sum += distance
16
17        node_density = distance_sum / self.chord.size_ft
18        passed = (node_density <= self.expected_node_density *
              gamma)
19
20        return passed
```

Listing 2: Bounds Checking Algorithm

# C Spot Check Algorithm

```
1     def spot_check(self, chain):
2         entrys = random.sample(chain, [1...3])
3         for entry in entry:
4             node = self.chord.getNodeByID(entry.getSender())
5             if not node:
6                 return False
7
8             ft = node.getFingerTable(self)
9
10            if not self.boundsCheck(node, ft) and not self.
                  witnessListCheck(node, ft):
11                return False
12
13            return True
```

Listing 3: Spot Check Psuedocode

# D Acronyms

**PKI** Public Key Infrastructure
**CA** Certificate Authority
**Kad** Kademelia
**DHT** Distributed Hash Table
**DHTs** Distributed Hash Tables
**PIR** Private Information Retrieval
**P2P** Peer-to-Peer
**DKG** Distributed Key Generation
**ORAM** Oblivious Random Access Memory
**SGX** Security Guard Extensions
**TEEs** Trusted Execution Environments
**RAM** Random Access Memory

# List of Figures

# List of Tables

# Bibliography


[1] Dawinder S Sidhu. The chilling effect of government surveillance programs on the use of the internet by muslim-americans. *U. Md. LJ Race, Religion, Gender & Class*, 7:375, 2007.

[2] Charlie Savage, Julia Angwin, Jeff Larson, and Henrik Moltke. Hunting for hackers, nsa secretly expands internet spying at us border. *The New York Times*, 4, 2015.

[3] Andrew Clement and Jonathan A Obar. Canadian internet "boomerang" traffic and mass nsa surveillance: Responding to privacy and network sovereignty challenges. *Law, privacy and surveillance in Canada in the post-Snowden era*, pages 13–44, 2015.

[4] Tamara Dinev, Paul Hart, and Michael R Mullen. Internet privacy concerns and beliefs about government surveillance–an empirical investigation. *The Journal of Strategic Information Systems*, 17(3):214–233, 2008.

[5] What's the value of your data? `https://techcrunch.com/2015/10/13/whats-the-value-of-your-data/?guce_referrer_us=aHR0cHM6Ly90ZWNoY3J1bmNoLmNvbS8&guce_referrer_cs=Yb6jAdzrrv-ceEnSiZekhA`, 2015.

[6] I know where you've been: Digital spying and divorce in the smartphone age. `https://www.npr.org/sections/alltechconsidered/2018/01/04/554564010/i-know-where-you-ve-been-digital-spying-and-divorce-in-the-smartphone-age?t=1589872966704`, 2018.

[7] What americans think about nsa surveillance, national security and privacy. `https://www.pewresearch.org/fact-tank/2015/05/29/what-americans-think-about-nsa-surveillance-national-security-and-privacy/`, 2015.

[8] The cambridge analytica files. `https://www.theguardian.com/news/series/cambridge-analytica-files`, 2020.

[9] Aclu: Nsa surveillance. `https://www.aclu.org/issues/national-security/privacy-and-surveillance/nsa-surveillance`, 2020.

[10] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[11] The tor project. `https://www.torproject.org/`, 2019.

[12] Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous commmunication using redundant structured topologies. volume 54, November 2009.

[13] Michael J Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, 2002.

[14] Andriy Panchenko, Asya Mitseva, Torsten Ziemann, and Till Hering. Guardedgossip: Secure and anonymous node discovery in untrustworthy networks. Technical report, Brandenburg Technische Universität, 2020.

[15] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected areas in Communications*, 16(4):482–494, 1998.

[16] Qiyan Wang and Nikita Borisov. Octopus: A secure and anonymous DHT lookup. *CoRR*, abs/1203.2668, 2012.

[17] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[18] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 448–457, USA, 2001. Society for Industrial and Applied Mathematics.

[19] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database,

computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373. IEEE, 1997.

[20] Prateek Mittal, Femi G Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. Pir-tor: Scalable anonymous communication using private information retrieval. In *USENIX Security Symposium*, pages 31–31, 2011.

[21] Sajin Sasy and Ian Goldberg. Consensgx: Scaling anonymous communications networks with trusted execution environments. *Proceedings on Privacy Enhancing Technologies*, 2019(3):331–349, 2019.

[22] Ian Clarke et al. A distributed decentralised information storage and retrieval system. 1999.

[23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[25] Jihong Song and Shaopeng Wang. The pastry algorithm based on dht. *Computer and Information Science*, 2(4):153–157, 2009.

[26] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[27] Andriy Panchenko, Stefan Richter, and Arne Rache. Nisan: Network information service for anonymization networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 141–150, New York, NY, USA, 2009. Association for Computing Machinery.

[28] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Practical robust communication in dhts tolerating a byzantine adversary. In *2010 IEEE 30th International*

*Conference on Distributed Computing Systems*, pages 263–272. IEEE, 2010.

[29] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 654–663, New York, NY, USA, 1997. Association for Computing Machinery.

[30] Rupeng Yang, Man Ho Au, Qiuliang Xu, and Zuoxia Yu. Decentralized blacklistable anonymous credentials with reputation. *Computers & Security*, 85:353–371, 2019.

[31] István Hegedűs, Gábor Danner, and Márk Jelasity. Gossip learning as a decentralized alternative to federated learning. In *Distributed Applications and Interoperable Systems*, pages 74–90. Springer International Publishing, 2019.

[32] Peng Wang, Ivan Osipkov, N Hopper, and Yongdae Kim. Myrmic: Secure and robust dht routing. *U. of Minnesota, Tech. Rep*, 2006.

[33] Marc Rennhard and Bernhard Plattner. Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 91–102, 2002.

[34] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 308–318, 2010.

[35] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, pages 21–es, 2004.

[36] Max Schuchard, Alexander W. Dean, Victor Heorhiadi, Nick Hopper, and Yongdae Kim. Balancing the shadows. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society, WPES '10, Co-located with CCS'10*, pages 1–10, 12 2010.

[37] George Danezis and Richard Clayton. Route fingerprinting in anonymous communications. In *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pages

69–72. IEEE, 2006.

[38] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 151–166. Springer, 2008.

[39] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[40] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 17–26, 2006.

[41] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 590–599, New York, NY, USA, 2009. Association for Computing Machinery.

[42] Robin Snader and Nikita Borisov. A tune-up for tor: Improving security and performance in the tor network. In *NDSS*, 2008.

[43] Intel. Intel academic research sgx. `https://software.intel.com/en-us/sgx/documentation/academic-research`, 2020.

[44] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. *Technical report*, 2018. See also USENIX Security paper Foreshadow [45].

[45] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018. See also technical report Foreshadow-NG [44].

[46] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. LVI: Hijacking

Transient Execution through Microarchitectural Load Value Injection. In *41th IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[47] Chelsea H. Komlo, Nick Mathewson, and Ian Goldberg. Walking onions: Scaling anonymity networks while protecting users. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1003–1020. USENIX Association, August 2020.

[48] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 92–102, 2007.

[49] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, December 2003.

[50] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Transactions on Information and System Security (TISSEC)*, 15(1):1–28, 2012.

[51] Prateek Mittal. A security evaluation of the salsa anonymous communication system. 2010.

[52] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

[53] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. Adding query privacy to robust dhts. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 30–31, 2012.

[54] Moni Naor, Benny Pinkas, and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[55] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[56] Parisa Tabriz and Nikita Borisov. Breaking the collusion detection mechanism of morphmix. In *International Workshop on Privacy Enhancing Technologies*, pages 368–383. Springer, 2006.

[57] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[58] Till Hering. Palaver: Secure and anonymous peer-to-peer random node lookup in untrustworthy environments. 2014.

[59] C. Genest, R. Lockhart, and M. Stephens. Chi-square and the lottery. 2001.