A Survey of DHT Security Techniques

GUIDO URDANETA, GUILLAUME PIERRE and MAARTEN VAN STEEN
VU University, Amsterdam, The Netherlands

Peer-to-peer networks based on Distributed Hash Tables (DHTs) have received considerable attention ever since their introduction in 2001. Unfortunately, DHT-based systems have shown to be notoriously difficult to protect against security attacks. Various reports have been published that discuss or classify general security issues, but so far a comprehensive survey describing the various proposed defenses has been lacking. In this paper, we present an overview of techniques reported in the literature for making DHT-based systems resistant to the three most important attacks that can be launched by malicious nodes participating in the DHT: (1) the Sybil attack, (2) the Eclipse attack, and (3) routing and storage attacks. We review the advantages and disadvantages of the proposed solutions and in doing so, confirm how difficult it is to secure DHT-based systems in an adversarial environment.

Categories and Subject Descriptors: C.2.4 [**Distributed Systems**]: Distributed Applications; C.2.0 [**General**]: Security and Protection

General Terms: Algorithms, Design, Security

Additional Key Words and Phrases: Peer-to-peer systems, Distributed Hash Tables, Sybil attack, Eclipse attack, Secure P2P routing and storage

1. INTRODUCTION

The notion of a decentralized lookup service is very useful for many distributed applications [Cai et al. 2004; Ramasubramanian and Sirer 2004b; Zhou and van Renesse 2004; Dabek et al. 2001; Rowstron and Druschel 2001b; Castro et al. 2002; Stading et al. 2002]. Such a service provides the fundamental operation *lookup(k)*, which returns data associated with a key *k*. A common approach studied in the literature to implement this functionality is the use of structured peer-to-peer systems, also known as distributed hash tables (DHTs) [Ratnasamy et al. 2001; Stoica et al. 2003; Rowstron and Druschel 2001a; Zhao et al. 2004; Maymounkov and Mazières 2002]. Examples of deployed systems that rely on DHTs include BitTorrent, the Kademlia-based KAD file sharing network used by eMule, MLDonkey and other compatible programs, peer-to-peer search engines [Yang and Ho 2006; Tang et al. 2003], and botnets [Holz et al. 2008].

In a DHT, keys are distributed among a potentially very large number of nodes. Each node needs to know the identity of only a small subset of other nodes, such that a lookup can be routed deterministically to the node responsible for the requested key.

Authors' address: VU University, Department of Computer Science, Computer Systems Group. De Boelelaan 1081, 1081HV Amsterdam, The Netherlands.

Guido Urdaneta is supported by the Programme Alban, the European Union Programme of High Level Scholarships for Latin America, scholarship No.E05D052447VE.

This limited view of the membership, while essential for the system's scalability, makes it remarkably difficult to make DHTs tolerant to the presence of malicious and possibly colluding nodes in an open environment such as the Internet, where mutually untrusted parties are allowed to join the system. This limited knowledge about the system can be exploited by malicious nodes to launch attacks that can compromise the integrity of the system.

Besides generic attacks such as those for (distributed) denial-of-service and exploitation of implementation bugs, which apply to any distributed system, DHTs present a number of weaknesses specific to them. The most studied DHT attacks in the literature are: (1) the Sybil attack, where an attacker introduces a large number of bogus nodes that can subvert protocols based on redundancy, (2) the Eclipse attack, where the attacker tries to corrupt the routing tables of honest nodes by filling them with references to malicious nodes, and (3) routing and storage attacks, which cover various attacks where malicious nodes do not follow the routing and storage protocols correctly, for example, by corrupting data, or routing to incorrect or malicious nodes.

To illustrate, Figure 1 shows a typical organization of a DHT, in which each peer has an m-bit identifier. (In the example, $m = 5$, but normally $m = 128$ or larger.) In a DHT, peers are responsible for keeping information on entities, where each entity has a unique *key*, drawn from the same space as peer identifiers. For example, in Chord, all entities with a key k fall under the jurisdiction of the peer with the smallest identifier $id \geq k$. To allow for efficient lookups of keys, each peer maintains a routing table (to which we return below).

When focussing on the application-independent core of DHTs, one can indeed see that there are two major problems: creating malicious nodes and isolating nodes from benign ones. Creating malicious nodes is exactly what is done through a Sybil attack, whereas isolation is done by means of an Eclipse attack. Once malicious nodes have been installed and benign nodes have been eclipsed, the real damage to the DHT core can take place: manipulating lookup requests by forwarding requests to malicious nodes in addition to returning bogus results.

Apart from these, any application built on top of DHTs faces specific security threats. For example, file-sharing applications built on DHTs are vulnerable to the introduction of "poisoned data", where seemingly correct but bogus files are introduced in the system by an adversary with the purpose of disrupting user downloads. As previously said, DHTs can be used for building a wide range of applications which can be subject to an equally varied set of attacks. We do not consider application-specific attacks in this paper.

DHTs also have to deal with other issues that may be, but are not necessarily, the result of an attack. One of the most serious is churn, which consists of participating nodes dynamically joining and leaving the system, requiring algorithms to efficiently handle continuous restructuring of the overlay in addition to migrating data. Another important issue is the unequal popularity of the data stored in a DHT combined with the unequal capacity of the participating nodes, which can lead to load-balancing problems. Related to this is the possibility of flash crowds, where the popularity of specific items increases several orders of magnitude in a short time. There is a vast amount of literature related to the study of churn [Castro et al. 2004; Rhea et al. 2004; Godfrey et al. 2006; Li et al. 2005; Blake and Rodrigues 2003], load balancing [Zhu and Hu 2005; Rao et al. 2003; Karger and Ruhl 2006; Godfrey et al. 2004] and flash crowds [Ramasubramanian and Sirer 2004a; Yu et al. 2005] in DHTs.

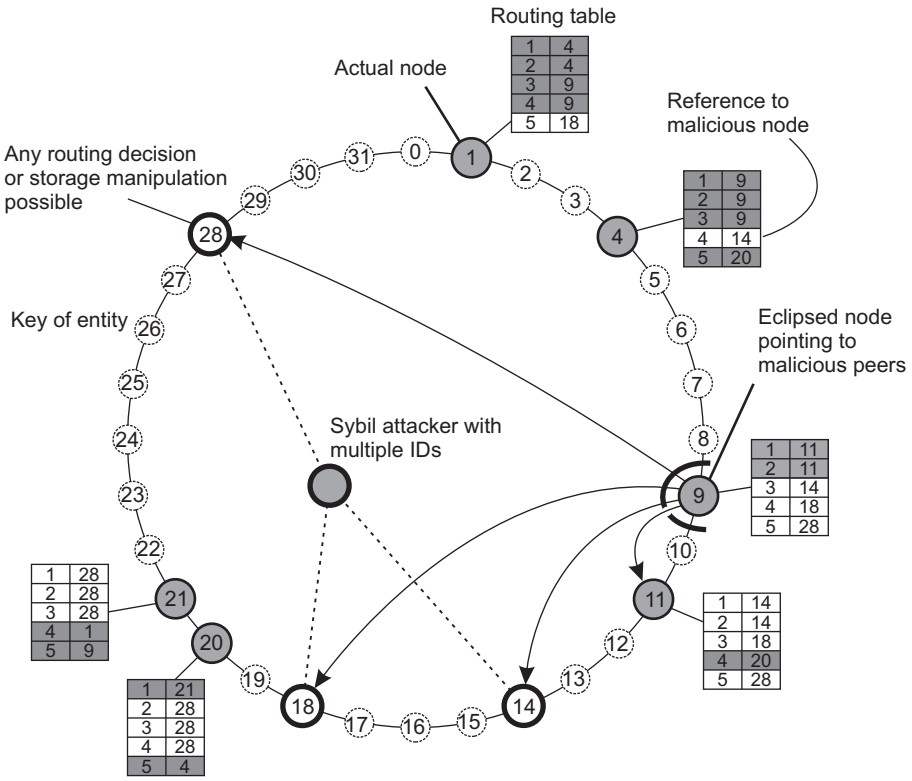


Fig. 1. The organization of a typical DHT, illustrating attacks on the core functionality.

Issues that are inherent to all DHT deployments, but which are independent of the associated protocols, such as churn and unbalanced loads, as well as application-specific attacks, are out of the scope of this paper. In the same light, we do not discuss denial of service attacks, which have been studied by Daswani [2004].

There have been several surveys that describe DHTs and peer-to-peer (P2P) systems in general. However, not many survey solutions to security issues in DHTs. Sit and Morris [2002] explore the subject and provide general guidelines. Castro et al. [2002] study DHT security issues under a generic DHT model, and provide solutions using Pastry as a representative of their model. Wallach [2002] discusses a wide range of security issues in several P2P systems, including Pastry, but does not enumerate the numerous proposals. Srivatsa and Liu [2004] make an extensive quantitative analysis of security threats in DHTs and some of the defenses. Levine et al. [2006] summarize general approaches to address the Sybil attack in a variety of scenarios, but do not discuss any specific measures. Reidemeister et al. [2005] study security issues specific to CAN [Ratnasamy et al. 2001]. Dahan and Sato [2007] criticize several practical aspects related to DHT security as well as their use in other systems that require security.

In this paper, we supplement these surveys by providing a comprehensive overview of the research in the area of DHT security, concentrating on numerous specific solutions. We focus on proposed defenses against the aforementioned attacks, discuss their advan-

tages and disadvantages, and identify possible areas for future research. To the best of our knowledge, this is the first study in security for DHT-based systems that provides such a comprehensive overview. Given the volume and quality of the research in this area, we come to the conclusion that developing secure DHTs is a far from trivial undertaking for which acceptable solutions are still sought.

The rest of the paper is organized as follows. Section 2 gives an overview of DHTs. Section 3 discusses the Sybil attack. Section 4 covers the Eclipse attack. Section 5 discusses routing and storage attacks. Section 6 discusses how security techniques have been applied in deployed open systems based on DHTs, and Section 7 serves as our conclusion.

2. OVERVIEW OF DHTS

DHTs are distributed systems composed of many nodes that implement the operation *lookup(k)*, which returns data associated with a key k. The data typically contains the network address of the node responsible for key k. Alternatively, a DHT may implement the operation *route(k)*, which simply routes a message to the node responsible for key k.

The most fundamental aspect of a DHT is the existence of a common identifier space for both nodes and keys, with each key k stored in the node with identifier closest to k according to some distance function.

In order to locate the node responsible for key k, a node forwards the lookup request to another peer whose identifier is closer to k according to the distance function. All nodes maintain links to a subset of the other nodes, thus forming an overlay network. A lookup request is forwarded by nodes until no node is found with an identifier closer to the requested key. In order to be scalable, the number of links per node must be small in comparison to the total number of participating nodes, and is typically of size $O(\log N)$. This partial view of the system makes DHTs scalable, but also makes them vulnerable to malicious nodes that do not behave according to the protocols as originally set out.

There are many ways in which a DHT can implement these concepts. For example, Chord [Stoica et al. 2003] uses an identifier space consisting of m-bit strings arranged in a circle with points representing the integers from 0 to $2^m - 1$. The distance from identifier x to identifier y is the clockwise numerical distance $(x - y) \bmod 2^m$. Each node with identifier x builds and maintains a routing table consisting of its immediate predecessor node on the circle, its immediate k successors, and a list of m fingers consisting of the nodes whose identifiers immediately succeed $\left(x + 2^{j-1}\right) \bmod 2^m$, for $1 \leq j \leq m$. Chord provides logarithmic lookup time, and requires a logarithmic amount of memory per node.

Another popular DHT is Pastry [Rowstron and Druschel 2001a] which uses strings of l digits with base $b = 2^p$ to identify nodes, where p is an integer. The distance is determined by the number of common prefix digits and complemented by a network proximity metric. Each node x maintains a routing table with ℓ rows that contain $b-1$ entries each. Each row r ($1 \leq r \leq \ell$) contains links to nodes sharing the first r digits with x, but with the $(r+1)$st digit being one of the $b-1$ possible values other than the $r+1$th digit of x. In addition, each node maintains a neighborhood set and a leaf set. The neighborhood set consists of the M closest nodes according to the network proximity metric, while the leaf set contains the $L/2$ nodes with numerically closest smaller IDs, and the $L/2$ nodes with numerically closest larger IDs relative to x. Pastry also provides logarithmic lookup time, and also requires a logarithmic amount of memory per node.

One of the most widely used DHTs in real-world applications is Kademlia [Maymounkov

and Mazières 2002]. Like Chord, it uses m-bit strings as node identifiers. The distance between two nodes is determined by the application of the bitwise XOR operation on their identifiers. This results in a metric similar to Pastry's. Unlike Chord and Pastry, Kademlia maintains redundant links to same fractions of the ID space. For each $0 \leq i < m$, each node in Kademlia keeps a list of links for nodes of a distance between 2^i and 2^{i+1} from itself. The list can grow up to size s and is called an s-bucket. Links from an s-bucket are selected to refer preferably to highly available nodes. To look up a key, a querying node chooses α nodes from its closest non-empty s-bucket, and sends them requests for their s closest known nodes to the target identifier. The querying node selects α nodes among the responses and repeats the procedure until no nodes closer to the target are found. Data is stored in the s closest nodes to a given key. When $\alpha = 1$, routing is similar to Pastry's and Chord's.

Routing tables in Kademlia can be augmented with network proximity information, which can be used to improve lookup performance.

3. SYBIL ATTACK

The Sybil attack was first studied by Douceur [2002]. It exploits the fact that in a distributed system, remote entities are perceived as informational abstractions known as *identities*. If the system fails to guarantee that each logical identity refers to a single physical entity, an attacker could create a large number of identities and dominate the overlay network by fooling the protocols and subverting mechanisms based on redundancy. The Sybil attack does not damage the DHT by itself, but can be used as a vector to artificially create a majority of colluding malicious nodes in the overlay. Many DHT defenses have been designed under the assumption that only a reasonably low fraction f of nodes are malicious. A Sybil attack breaks these defenses by effectively increasing f.

This attack is not specific to DHTs, but it is important to study because DHTs are vulnerable to it and the attack can be used to facilitate the execution of many other attacks. For example, if there are many malicious identities in the system, it becomes easier to pollute the routing tables of honest nodes, and control the majority of the replicas for a given key.

The most important conclusion of Douceur's study is that in a P2P system, having a logically central, trusted authority to issue identities is the only practical way to guarantee a one-to-one correspondence between the identities and the physical entities that operate the participating nodes.

3.1 Castro et al.

Castro et al. [2002] argue that, as Douceur stated, the only practical solution to node identifier generation is to use a trusted authority and reject any form of distributed identifier generation. They suggest using a set of trusted certification authorities to produce signed certificates that bind a random node identifier to a public key and the node's IP address. They suggest including the IP address in the certificate so that it is difficult for an attacker to swap certificates between nodes it controls and also to allow optimizations based on minimizing communication delays. This type of certified identifiers works well with DHTs such as Chord, Pastry, and Tapestry [Zhao et al. 2004], where the identifiers are fixed. However, they are not suitable for systems such as CAN [Ratnasamy et al. 2001], where the identifiers represent regions of a d-dimensional space that change when new nodes join. Also, IP-based schemes require special solutions when having to deal with machines behind NAT firewalls.

In order to prevent Sybil attacks, the authors propose to make it difficult for an attacker to obtain a large number of certified identifiers. One of their solutions is to charge money for each certificate. Another is to bind node identifiers to real-world identities, but this works only in systems that already have reliable authentication procedures.

We believe that using certificates is the most effective defense against the Sybil attack as long as it is possible to have a trusted authority that is able to discern if an entity requesting a certificate is a Sybil attacker or not. The main advantage of this approach is that it provides flexibility over what the system considers a "real" entity. For example, a criminal organization running a botnet has many physical nodes with different IP addresses, but it is possible that it is considered as a single entity by the certification authority. On the other hand, running such a trusted authority implies an administrative overhead whose feasibility depends on the application and real-world circumstances. Charging money for the certificates would help fund the operation of the authority and limit the number of certificates that a malicious entity can acquire, but has the disadvantage that it can discourage legitimate nodes from participating.

3.2 Dinger and Hartenstein

Dinger and Hartenstein [2006] propose a distributed registration procedure for Chord. Although distributed schemes are known not to be totally Sybil-resistant, they use a number of metrics to quantify a level of resistance. In their system, each virtual node calculates its ID as a hash of its IP address and port number, and registers itself at r registration nodes in the Chord ring. The r registration nodes are computed using the hash of the IP address and an integer j ($1 \leq j \leq r$). Registration nodes maintain a list of registered virtual nodes for each IP address and reject registration if the number of registered nodes for each IP address exceeds a system-wide constant a. They also modify the Chord stabilization algorithm in such a way that it confirms the legitimacy of the identifier, computes the appropriate registration identifiers for the new node, and checks the correct registration of the new node by asking the responsible registration nodes. If the number of positive replies is greater than $\lceil r/2 \rceil$, the new node will be accepted. If a new node joins successfully, then it is also integrated in the registration process and registration data is migrated from other nodes according to the standard Chord protocol. The basic idea of this system is to implement an approximation of a trusted registration authority by assuming that a majority of the non-trusted registration nodes will behave correctly.

In their simulation experiments, they use three measures to quantify the level of Sybil resistance of their system. The first is the fraction of malicious identities with respect to the total number of identities in the system. This value is calculated assuming that each good participant obtains one identity and each malicious participant obtains a identities. They compared the theoretical value with the actual value obtained in an experiment in which the number of registered identities grows to around 500, with a 0.02 probability that a participant trying to join is malicious, and with parameters $a = 2$ and $r = 5$. Once a malicious participant is accepted it tries to create as many Sybil identities as possible. Their result is that the fraction of malicious identities never exceeds its expected value, which for this experiment was approximately 0.0392.

Their second measure is the probability of a *false registration*, which they define as the probability that a malicious node is accepted, given a fraction of well-behaving nodes in the overlay, and the parameters a and r. The results show that this probability decreases if r increases. For $r = 5$ the probability is more than 0.3 with a fraction of 60% of honest

nodes and drops to a near-zero value with 90% of honest nodes. For $r = 12$ the probability is close to zero with 60% of honest nodes. For values of r greater than 24 the probability drops to zero with a fraction of honest nodes slightly greater than 40%. In all cases the parameter a is set to 1.

Their third measure of Sybil resistance is the fraction w of well-behaving nodes necessary to guarantee a probability of false registration less than 0.001. Their result is that increasing the replication factor r decreases w, but w converges to 0.5 because at least 50% of the nodes have to confirm the correct registration.

These results show that this approach provides a reasonable level of Sybil protection by regulating the number of identities that a malicious IP address can get. However, it introduces the possibility of new attacks. For example, consider the case where a coalition of malicious nodes introduces fake registration values for potential legitimate nodes by repeatedly joining and leaving in such a way that the fake registration values are moved to good nodes. This would effectively reduce the probability of a legitimate node to join the system, thus increasing the fraction of malicious identities in the system. Moreover, the only notion of entity that the system allows is the IP address (or prefix), which is not very useful if the adversary controls many IP addresses, something easy to achieve by assigning many addresses to a single computer, or with a botnet. Again, IP-based schemes require special attention when having to deal with NAT firewalls.

3.3 Wang et al.

Wang et al. [2005] propose a different approach to building secure DHTs. They reject using the IP address or payed certificates to identify nodes and consider these approaches impractical in a P2P environment. They argue that physical network characteristics can be used to identify nodes. They introduce a concept called *net-print* for this purpose. The net-print of a node is built using a node's default router IP address, its MAC address and a vector of RTT measurements between the node and a set of designated landmarks. This is a form of self-certifying data, which can be verified by other nodes, making identity theft difficult. The authors state that a machine may claim multiple identities in the same subnetwork and launch a Sybil attack, but the scope of this attack would be limited and can be detected by challenging every identity with a unique computational puzzle concurrently.

The main disadvantage of this approach is that changes in network conditions may cause subsequent identity tests to fail. It is therefore necessary to tolerate a certain deviation between the values reported by a node and the measures obtained in the verification, but determining the appropriate tolerance requires a difficult trade-off between security (which demands low tolerance), and resilience to network condition changes (which requires high tolerance). Another problem is that any change in the network measurement infrastructure implies a change in the identities of all nodes. In addition, it is not possible to support mobile hosts with this system.

3.4 Bazzi and Konjevod

Bazzi and Konjevod [2005] propose a Sybil defense based on network coordinates. This work exploits the fact that location is a physical property of entities that can help determine their true identities. The model assumes that the participating nodes form a d-dimensional Euclidean space $\mathbb{R}^d$ or a sphere $\mathbb{S}^d$. The distances in this space are assumed to approximately satisfy the metric properties of symmetry and triangle inequality. It is also assumed that the distance between two points in the space is a nondecreasing function of roundtrip

delay between them. The proposed protocols distinguish two types of nodes: *beacons* and *applicants*. Some applicants and some beacons may be malicious.

The two basic elements for the operation of the system are *geometric certificates* and a *distinctness test*. A geometric certificate is a set of distance values between an applicant and the beacons signed by both the beacons and the applicant. A geometric certificate is the result of the execution of a protocol that might require the nodes (applicant and beacons) to send and respond to probe messages as well as reporting the results to various beacons. A distinctness test is a function $D : C \times C \rightarrow \{true, unknown\}$ that assigns a value true or unknown to a pair of geometric certificates. If the result is true, the entities are considered distinct.

The authors present certification protocols for different settings, such as:

—Honest participants

—Non-colluding malicious applicant

—Multiple colluding applicants in a broadcast environment

—Multiple colluding applicants in a point-to-point environment

—A fraction of colluding beacons

The main practical property of this approach is that the distinctness test introduces an equivalence relation where nodes in the same class cannot be distinguished from each other using the test. This allows applications to implement more reliable redundancy protocols, for example, by storing replicated data in nodes belonging to different classes. It should be noted that this system does not aim to identify individual nodes, but to guarantee that identities in different groups are not controlled by the same entity. A weakness is that an adversary controlling real nodes in different groups can easily beat the defense: identified by the protocol as belonging to different groups, the nodes are actually controlled by a single entity, which is the essence of the Sybil attack. In addition, the authors assume that there is a unique attainable distance between two nodes. Practical distance estimation shows that this assumption may be false [Szymaniak et al. 2004; Szymaniak et al. 2008] and that considerable effort is needed to come to stable estimates, if possible.

3.5 Bazzi et al.

Bazzi et al. [2006] propose a secure routing algorithm that can be used to defend against Sybil attacks. The main idea is to determine hop-count distances to a set of beacons and use these to tell nodes apart. Using hop-count coordinates has been studied notably in the context of wireless networks [Fonseca et al. 2005].

In this system, each node has a public-private key pair. Each honest node can be identified by a unique public key, while a Sybil attacker can have multiple key pairs.

The proposed protocol for determining hop-count distances has two components: a protocol that allows a node to determine if another node is its physical neighbor (that is, a neighbor that can be contacted without intervention of another node), and a protocol that uses key chains to enable a destination node to certify its distance to other nodes in the network.

To determine if a destination node is a physical neighbor, it is assumed that a broadcast mechanism to physical neighbors exists. The algorithm has two phases. First, a node sends a random bit encrypted with the public key of the destination node. The destination can then recover the bit by decrypting the message. In the second phase, the sender broadcasts

a cleartext message and the destination can XOR the first bit of the message with the previously recovered random bit and resend the message back to the sender. This algorithm can be repeated multiple times to arbitrarily reduce the probability of a malicious node correctly guessing the correct answer. This allows the sender to determine that the destination is a physical neighbor, or, more formally, that one of its physical neighbors has access to the private key of the destination.

The second part of the protocol is to actually determine hop-count distances between nodes. In this case, the main purpose of the protocol is to prevent malicious nodes from reporting distances that are smaller than their actual distances to a given destination.

The basic algorithm calculates distances towards a destination node d by constructing a minimum spanning tree. Initially, the tree includes only d and the distance to d is infinite at all other nodes. Each member node x of the tree periodically advertises to its physical neighbors its shortest known working path to d with a message of the form $adv(BK_x, dt, H_x)$, where BK_x is the public key of node x, dt is a timestamp updated periodically at each node, and H_x is a description of the path from d to x using the chain $< dt, BK_0, BK_1, \cdots, BK_m, C_0, C_1, \cdots, C_m >$. In this chain, BK_i are the public keys of the nodes in the path from d to x, C_0 is a self-signed digital certificate generated by d, and C_i are digital certificates for the i-th node in the path, signed by the $(i-1)$-th node in the path, with C_1 signed by d. The distance from x to d is the number of certificates in the chain H_x.

When a node y receives an advertisement $adv(BK_x, dt, H_x)$ from its physical neighbor x, it checks if the timestamp dt is recent and whether it allows reducing y's path length. If this is the case, it sends a reply message $rpl(BK_x, BK_y, dt)$ to x, which is essentially a certificate request.

When x receives $rpl(BK_x, BK_y, dt)$, it computes a certificate C_x consisting of a hash of the pair (dt, BK_y) signed with x's private key. The certificate essentially means that x recognizes y as its physical neighbor of which a statement is sent to y in an acknowledgment message $ack(BK_y, dt, C_x)$.

When y receives $ack(BK_y, dt, C_x)$, it checks the certificate C_x and, if correctly signed, it updates its current path to d with the hop-chain

$$H_y =< dt, BK_0, BK_1, \cdots, BK_m, BK_x, C_0, C_1, \cdots, C_m, C_x >$$

which is the H_x received in the advertisement message sent by x with BK_x and C_x appended. After this is done, y will periodically broadcast new advertisement messages $adv(BK_y, dt, H_y)$ to its physical neighbors.

When a node y stops receiving the periodic advertisement messages of its parent x in the routing tree, it resets its distance to d to an infinite value and stops advertising its route to d through x.

This protocol tolerates malicious nodes without collusion, but the authors have extended it to withstand the following attack models: (1) Initial collusion of nodes, in which all malicious nodes share some initial information (e.g., included in a virus), but cannot communicate with other malicious nodes, (2) runtime collusion of adjacent nodes, and (3) runtime collusion of non-adjacent nodes, but in the absence of collusion between adjacent nodes.

The authors claim that nodes can be identified with vectors consisting of hop-count distances to a set of beacons calculated with the previously described protocol, since the distances between nodes cannot be affected by malicious nodes, provided the network has enough redundant paths to ignore.

We observe that this solution can be relatively simple to implement and looks very

promising in wireless scenarios. However, it has not been tested experimentally, so it is unclear how effective it really is, especially in a dynamic environment with frequent changes in the network topology.

The implications of using this scheme in an overlay network on top of a wired network are also not clear. If physical neighbors are defined as belonging in the same physical underlying network link then it is possible that many participating nodes have no physical neighbors. On the other hand if physical neighbor means that a node does not need to route a message using other overlay nodes, then any node may have the possibility to become a "physical" neighbor of any other node.

3.6 Danezis et al.

Danezis et al. [2005] offer a method that uses social information to make routing in DHTs more resistant to Sybil attacks. More concretely, they propose changes to the Chord iterative routing method so that it is less vulnerable to Sybil attackers attempting to disrupt routing lookup requests.

Their approach relies on the *bootstrap graph* of the system, i.e., an initial overlay network that connects designated *attachment* nodes that can be used by others to join the network. They assume the bootstrap graph to be a tree. The reasoning is that the easiest way for an adversary to add malicious nodes is to first convince a legitimate node to admit a single malicious node, perhaps using social engineering, and then use the malicious node as the entry point for adding a large number of additional attackers. Thus, the single good node and the malicious node used as attachment point will appear in all routes from a good node to a malicious node in the bootstrap graph.

The Chord protocol is modified so that, in addition to node identifiers and addresses, each node stores the path in the bootstrap graph from itself to each node it knows (fingers, successor and predecessor). Another modification to the Chord protocol is that in lookup requests the current hop does not return a single next-hop node, but all the nodes it knows about, together with the bootstrap paths. Then, the requester can decide on the next hop using either the standard Chord strategy (the node numerically closest, yet less or equal to the requested key) or a strategy based on diversity. The purpose of the diversity strategy is to favor alternative paths that include nodes that are not frequently used. This is done applying the following rule: The requester maintains a histogram with the frequency with which each node in the network has been on the path of the queries so far. This histogram is referred to as a *trust profile*. Then, for each possible next hop, the variation of the histogram is computed, and the node chosen is the one that produces the least increase on the trust put on a single node. This is implemented by sorting in descending order the trust profile for each possible node, then ranking the nodes by lexicographically sorting the previously sorted trust profiles, and choosing the smallest node in this rank as the next step.

The diversity strategy distributes queries across the network trying to avoid the bottlenecks introduced by possible Sybil attacks. However, this strategy does not produce any progress towards routing to the target node. For this reason, the authors implement a *zigzag* strategy in which they use the standard and diversity strategies alternatively. In a simulation of 100 lookups in a system with 100 good nodes, the zigzag strategy outperforms the standard strategy as the number of malicious nodes grows. Zigzag produced better results with as few as 50 malicious nodes. The case in which there are several malicious attachment points was also studied. With 100 good nodes and 100 malicious nodes, zigzag outperformed the standard strategy only if there were less than 80 attachment points. With

200 malicious nodes, zigzag always outperformed the standard strategy, but the difference decreases as the number of attachment points grows.

An additional modification to Chord that the authors tested was to modify the way the finger table is constructed. A fraction of the fingers is selected in such a way that they produce the least increase to a trust profile created using information provided by the successor. A simulation experiment with 100 lookups in a system with 100 good nodes with one Sybil attachment point shows that both the standard and zigzag strategies benefit from the new finger table.

This system appears to have several disadvantages. First, it is unclear to what extent the logarithmic lookup times as provided by Chord are maintained. Second, it increases the overhead by requiring that each lookup step returns all neighbors plus the bootstrap paths. Third, it assumes social relationships between participants, with an implicit capability to detect if a node is malicious before joining. This requirement may prove to be difficult to maintain in practice, and it is unclear whether it will necessarily lead to attacks against only a few attachment points. Finally, the experiments were limited to simulated networks with no more than 100 honest nodes, and it is not clear if the system can scale to larger system sizes or with more attachment points.

3.7 Yu et al.

Yu et al. [2006] propose a decentralized protocol based on a social network that tries to limit the influence of Sybil attacks. In the social network, the edges of the graph represent a human trust relation such as friendship. The basic assumption is that a Sybil attacker can create any number of identities and edges among the Sybil identities, but it is limited in the number of edges that it can establish with honest nodes, which are referred to as attack edges.

The SybilGuard protocol partitions the nodes into groups such that the number of groups that include at least one Sybil identity (Sybil groups) is bounded by the number of attack edges, independently of the number of Sybil identities.

Each node constructs its own partition of the network using a procedure based on random routes. Random routes are a special kind of random walk where each node uses a precomputed random permutation as a one-to-one mapping from incoming edges to outgoing edges. For example, if a node has 3 edges e_1, e_2, e_3 and uses permutation (e_2, e_3, e_1) then all routes in the system that come from edge e_1 will use edge e_2 as the next hop. Random routes have two important properties. First, once two routes intersect at an edge in the same direction, they will merge. Second, the routes are back-traceable because the permutation on each node is fixed.

Each node has a route of length w for each of its edges. The reasoning for defining groups using random routes is that routes originating in the honest region of the network are unlikely to include nodes in a Sybil region, since the number of attack edges is small. Note that it turned out that w can be fairly large: 2000 for a one-million node network.

When a node wants to verify that another node is honest, it checks for intersections in their routes. A verifier node V is said to accept a suspect node S only if at least half of V's routes intersect with any of S's routes, meaning that most likely S will belong to the same (honest) region as V.

The authors provide algorithms for building the routes in a decentralized fashion. Apart from the routing table defined by the random permutation, each node maintains two types of data structures: registry tables and witness tables. The purpose of these data structures

is to prevent nodes from lying about their routes.

Each node has a public/private key pair and a symmetric key for each edge, shared with the corresponding neighbor in the social network, previously accepted as friend through social interaction. The distribution of symmetric keys is done offline. Each node registers with all the nodes in its random routes, and this information is maintained in registry tables.

Each node maintains a registry table per edge. The ith entry in a registry table for edge e contains the public key of the node with a route that has e as its ith hop. The procedure to construct registry tables is very simple. Suppose node A has an empty registry table and has a neighbor B connected via edge x. Then A sends to B (via a secure channel encrypted with their shared symmetric key) a table with A in its first position. Then B sends its table for edge x to its friend C via edge y. The registry table for edge y in node C will have B in position 1 and A in position 2. Since routes must be of size w, the wth entry is always dropped when a node propagates a registry table to a neighbor.

Registry tables allow nodes to register with peers in their random routes, but nodes also need to know which peers are on its random routes. This information is maintained in witness tables. Each node maintains a witness table for each of its edges. The ith entry in a witness table for edge e contains the public key and network address of the ith node in the route starting on edge e. The witness tables are propagated in a similar way to the registry tables, but in the reverse direction.

The verification procedure uses these tables in the following way. When node V wants to verify a node S, it compares all of S's witness tables with its own witness tables and checks for intersections. Then, for each intersection node X, V contacts X and verifies that S is registered with X. If the majority of V's routes have verifiable intersections with S routes, then S is considered an honest node.

When users and edges are added to or deleted from the social network, the internal data structures of the affected nodes must be updated as well. The most important change is on the permutation-based routing table. In order to prevent drastic changes in registry and witness tables, the change in the routing table is incremental. To add a new edge to a node with d edges, the node chooses a random integer k between 1 and $d+1$, inclusive, and replaces the edge in position k of the routing table with the new edge, and puts the replaced edge on position $d+1$. To delete the edge in position k of the routing table from a node with d edges, the node in position d is placed on position k and the table size is reduced to $d-1$. The churn is expected to be low, since social relationships are slow to change, so the registry and witness tables are updated with the execution of the corresponding maintenance protocols. These tables need to be updated only when there are changes in the social relationships.

The authors tested the system with simulations using Kleinberg's synthetic social network model [Kleinberg 2000]. They tested networks of up to 1,000,000 nodes with degree 24. When no malicious nodes exist the results show that 99.96% of the nodes have at least 10 intersections with a group size w as small as 300 (the protocol requires only one intersection). They also tested the case with malicious nodes and g attack edges. For the million-node network, they varied g from 0 to 2500. The results show that the probability of all the routes remaining in the honest region is almost always 100% when $g \leq 2000$. When $g = 2500$ this probability drops to 99.8%. The probability that an honest node accepts another honest node is 99.8% as well.

These results show that this scheme provides a reasonable level of Sybil resistance re-

gardless of the nature of the Sybil identities. However, there are several difficulties that complicate its real-world deployment. First, it requires genuine, trustworthy relationships among the participating entities that are accurately reflected in the system, a requirement which may prove to be difficult to maintain. Second, it requires mechanisms to securely establish these relationships in the system before the creation of links to other nodes according to the rules of the higher level application (such as the DHT protocol). Finally, it forces each user to securely distribute and administrate a different symmetric key with each of its friends in the social network. Whether these conditions can be met for actual applications remains to be seen.

3.8 Yu et al.

Yu et al. [2008] also proposed an update to SybilGuard called SybilLimit. Like in SybilGuard, each node has a public/private key pair, each social connection has a unique symmetric key, and there are protocols for maintaining random routes and verifying nodes. The SybilLimit random route protocol establishes permutation-based routing tables identical to SybilGuard's. However, instead of using a single routing table, each node must maintain $r = \Theta(\sqrt{m})$ independent routing tables, where m is the number of edges in the graph. Each suspect node S will execute an instance of the random route protocol for each of its routing tables. The protocol consists of S sending its public key K_S and a counter initialized to 1 on each route. Each receiving node will increment the counter and propagate the message according to its routing table, until the message reaches a value $w = O(\log n)$ in node B, through its edge with node A. In this case, node B records the key K_S under the name "$K_A \rightarrow K_B$". This is known as S registering its public key with the tail "$K_A \rightarrow K_B$". The tail "$K_A \rightarrow K_B$" is sent back to S using the inverse route. Each verifier node V also executes r independent instances of the protocol, the tails are propagated back to the verifier, but V does not register its public key at the tails.

When a verifier V wants to decide whether to accept a suspect node S or not, V checks that two conditions are satisfied: the *intersection condition* and the *balance condition*. The intersection condition consists in that the intersection of the set of V's tails and the set of tails with which S is registered must be non-empty. The authors show that this is an instance of the Birthday Paradox and that having $r = \Theta(\sqrt{m})$ is necessary and sufficient to have this non-empty intersection.

Having many random routes increases the possibility that some routes enter a region controlled by Sybil identities. It can then be expected that under a Sybil attack, the verifier tails that intersect with Sybil suspects' tails (and thus belong to routes that enter the Sybil region) will be used more often in the verification procedure. In order to detect this imbalance, the verifier will associate a counter to each tail. Everytime a suspect satisfies the intersection condition, the intersecting tail with the lowest counter will be selected and compared with $b = h\max(\log r, (1+\sum_{i=1}^{r} c_i)/r)$ where h is a positive constant (the authors use $h = 4$ in their experiments), and c_i are the counters for each intersecting tail. The suspect is rejected if $c_{min} + 1 > b$, otherwise, c_{min} is incremented. The authors show that the number of Sybil nodes accepted by V is bounded within $O(g \log b)$ where g is the number of attack edges.

The procedure to estimate r uses a *benchmarking technique*. Every verifier V maintains two sets of suspect nodes, a benchmark set K and a test set T. K is constructed by repeatedly executing random routes and adding the ending node to K. T contains the suspects V wants to verify. r is initially set to 1, and is repeatedly doubled. For each value of r, V

verifies all suspects in T and K and stops when a high fraction (e.g., 95%) of the nodes in K is accepted. This method is shown to never overestimate $r = r_0\sqrt{m}$.

To confirm their theoretical results, the authors evaluate SybilLimit with simulation experiments using a synthetic network of 1 million nodes and 10.9 million edges, generated using Kleinberg's social network model with $w = 10$ and $r = 10,000$, and three real world networks: Friendster with 932 thousand nodes and 7.8 million edges, $w = 10$, $r = 8,000$; LiveJournal with 900 thousand nodes and 8.7 million edges, $w = 10$, $r = 12,000$; and DBLP with 106 thousand nodes and 626 thousand edges, $w = 15$, $r = 3000$.

SybilLimit provides an asymptotic guarantee of $O(\log n)$ Sybil nodes accepted per attack edge, where n is the number of honest nodes. The experiments show that this translates to between 10 and 20 accepted Sybil nodes per attack edge.

In general, this work makes similar assumptions to its predecessor SybilGuard and thus the same observations regarding the practical realization of these assumptions apply.

Lesniewski-Laas [2008] proposes to augment SybilLimit with routing tables of size $r = O(\sqrt{m\log m})$. The additional $O(\sqrt{\log m})$ factor allows routing tables to be used for $O(1)$ DHT routing, similar to approaches described in [Gupta et al. 2003; Gupta et al. 2004].

3.9 Borisov

Borisov [2006] proposes to add computational puzzles to Chord in order to defend against Sybil attacks. The proposed scheme consists in augmenting the periodic ping messages every node sends to its neighbors with a sequence number and a challenge. If node x has neighbors $y_1, y_2, \cdots, y_m$, then x will send each y_i a sequence number $n^{(x)}$ and a challenge $c_{n^{(x)}}$ together with each ping. The challenge $c_{n^{(x)}}$ is defined as

$$c_{n^{(x)}} = H\left(y_1||n^{(y_1)}||c_{n^{(y_1)}}||\cdots||y_m||n^{(y_m)}||c_{n^{(y_m)}}||r_{n^{(x)}}||c_{n^{(x)}-1}\right)$$

where H is the SHA1 hash function, $||$ is the concatenation operation, $n^{(y_i)}$ and $c_{n^{(y_i)}}$ are the sequence number and challenge in the last ping message x received from each y_i, $r_{n^{(x)}}$ is a random number chosen by x, and $c_{n^{(x)}-1}$ is the challenge generated by x in the previous round.

Every t time steps, each node x must solve a puzzle based on the current value of $n^{(x)}$ and $c_{n^{(x)}}$. The puzzle consists of finding a number $s_{n^{(x)}}$ such that

$$H\left(x||BK_x||n^{(x)}||c_{n^{(x)}}||s_{n^{(x)}}\right) = h$$

where the last p bits of h are all zeros, and BK_x is x's public key. Solving this puzzle requires brute-force evaluation of up to 2^p candidate values for $s_{n^{(x)}}$.

When a neighbor y_i wants to confirm that $c_{n^{(y_i)}}$ was included in the computation of $c_{n^{(x)}}$, x can respond with the values y_j, $n^{(y_j)}$, $c_{n^{(y_j)}}$ corresponding to all its other neighbors, as well as $c_{n^{(x)}-1}$ and $r_{n^{(x)}}$ together with the solution $s_{n^{(x)}}$. Once y_i confirms that $c_{n^{(y_i)}}$ was included in the computation of $c_{n^{(x)}}$, it can contact a neighbor z of x and verify that $c_{n^{(z)}}$ is based on $c_{n^{(x)}}$ using the same procedure. It is possible that y_i needs to use an older challenge because $c_{n^{(y_i)}}$ may not have been propagated yet.

If the diameter of the overlay network is d, then y_i can start from $c_{n^{(y_i)}-d}$, and x can prove that it has solved a challenge $c_{n^{(x)}-l}$ that includes $c_{n^{(y_i)}-d}$, and any subsequent node z in a Chord lookup can prove that it has solved $c_{n^{(z)}-l'}$ which also includes $c_{n^{(y_i)}-d}$.

The author proposes that every node x compute $s_{n^{(x)}}$ every t time steps, which means that

nodes must be fast enough to compute 2^p hashes in no more than t time steps. In this case, the author shows that if a node x needs to perform a lookup that involves node y, x can start with $c_{n^{(x)}-d-2t}$ and y will be able to prove that it has solved a puzzle $c_{n^{(y)}-l'}$ which is based on $c_{n^{(x)}-d-2t}$ and whose solution has been computed within the last $d+2t$ time steps.

In order to tolerate churn, Chord's network maintenance algorithm is modified so that entries in the routing table remain at least $d+2t$ time steps, even if better nodes are added to the overlay within that period. In case of nodes leaving, the solution is to use suboptimal entries in the routing table. The author points out that Chord has been shown to have a high level of static resilience [Gummadi et al. 2003], with suboptimal routing being able to correctly route the majority of requests. Moreover, if self-certifying data is used, two separate routing tables could be used, one without the capability to do verifications and another one with the proposed algorithm, to be used only when data verification tests fail.

This approach gives a node the flexibility to choose its position in the overlay, thus making it easier to take advantage of load balancing algorithms. However, this flexibility allows a relatively small fraction of malicious nodes to easily take control of a certain region of the overlay, potentially blocking access to data stored in the attacked position.

3.10 Rowaihy et al.

Rowaihy et al. [2007] propose a hierarchical system based on computational puzzles to defend against Sybil attacks. The system creates a tree where the root must be trusted and reliable. The root allows other trusted nodes, such as major ISPs, to join the system. These in turn allow smaller providers, which in turn allow ordinary, untrusted users.

When a node x wishes to join the system it must start an admission sequence that starts at a leaf and ends at the root. Before joining the system, x generates a public-private key pair and its identifier, with its identifier being a cryptographic hash of the public key. Then it must discover a leaf node y and solve a puzzle generated by y, which consists of x guessing a random number R such that $H(BK_x||TS||R)$ matches a value specified by y, where H is the hash function, TS is a timestamp, BK_x is x's public key, and $||$ is concatenation. Once x proves that it has solved the puzzle, y generates a token that proves that x solved y's puzzle. This token is signed using a symmetric key shared between y and y's parent node.

After solving y's puzzle, x contacts y's parent, shows the token and solves another puzzle. x repeats this process until it reaches the root node, which provides a special token that proves that x has been admitted. This final token is signed using the root's private key, which allows all participating nodes to authenticate the token.

Once x is accepted into the system, it is not included in the admission hierarchy until it has been part of the network for a long time. When a node is promoted to be part of the admission tree, it uses an authenticated Diffie-Hellman exchange with its parent to establish the shared key used to validate intermediate tokens.

When a node that is part of the tree leaves the system gracefully, it informs its children and the one with more time in the network takes the place of the node leaving. This procedure is executed recursively in the children of the replacement node until a leaf is reached. If a node leaves due to a failure, all its children must rejoin the network.

There are two attack models of interest in this system: when the attacker is a member of the tree, and when it is not. In the first case, the attacker can generate tokens reducing the number of puzzles that must be solved. This can be detected by the parent of the attacker by observing the rate at which token requests are generated. In the second case, the only thing an attacker can do is to slowly obtain identities. To counter this, token expiration times are

introduced. Once a node knows its token will expire, it can get another one beforehand. If the attacker has n identities, it will have to get n new tokens, which limits its capability to maintain a large number of identities over time.

To protect the network during the startup process when there are few nodes, a startup window is defined. During this time, the puzzle difficulty decays progressively until the normal difficulty is reached. This makes it more difficult for malicious nodes to obtain identities at the beginning.

The authors formally analyze the system in addition to running simulation experiments, with the most important result being that the number of identities an attacker can accumulate is mt/l, where m is the number of real attackers, t is the expiration time for tokens, and l is the time needed to solve all the puzzles required to obtain a token from root node. Experimental results confirm the theoretical results.

We observe that this solution can limit Sybil attacks, but at the cost of requiring the equivalent of a centralized online certification authority, reliable nodes at the top levels of the hierarchy, and a continuous consumption of computing resources by honest nodes to maintain their membership.

3.11 Margolin and Levine

Margolin and Levine [2007] propose a protocol called *Informant* which is based on game theory principles to detect, rather than prevent, Sybil attacks. Informant makes the following assumptions: (1) all participants (including Sybil identities) are rational, (2) each identity has a public-private key pair, (3) there is a currency system that can be used to implement economic transactions, (4) benefits and harms associated with participation in the system can be expressed in terms of this currency, and (5) there is a reliable broadcast mechanism.

In Informant there are some trusted nodes known as *detectives* that are assumed to follow the protocol correctly. When the protocol is activated, a detective x broadcasts a signed message that signals the start of an instance of the protocol, specifying an identifier i for the protocol instance, and the duration τ of the instance. Immediately afterwards, an auction starts, in which x broadcasts a message indicating the reward R it is willing to provide for knowledge of a single Sybil relationship. If no responses are received in τ time units, x can increase R and start another round, until R reaches a maximum of $B/2$, with B being the detective's monetary benefit for learning about a Sybil relationship. If no responses are received when R reaches its maximum, the protocol ends. If some *informant* node y decides to report a Sybil relationship between itself and *target* z, it must report it to x together with the payment of a security deposit of D monetary units. After x gets the payment D, it pays $D+2R$ monetary units to z, and broadcasts a message indicating that y and z are Sybils. If x wants to learn more information about Sybils, it can start another instance of the protocol.

The authors analyze the protocol under the following conditions: (1) there is either a single Sybil attacker with multiple rational identities or no Sybil attacker at all, (2) Sybil attackers can be either *low-cost*, which reveal themselves with a reward $C < B = 2R$, or *high-cost* which reveal themselves with a reward $C' > B = 2R$, and (3) the detective runs the protocol to decide whether to continue participating or to leave. They prove that the detective will run the protocol unless $(1-\delta+\delta\gamma)B < \delta\gamma H + 2\delta(1-\gamma)R$, where δ is the probability that a Sybil attack is taking place, γ is the probability of a Sybil being high-cost, B is the benefit of participating in the application, and H is the monetary value of the harm caused by a Sybil attack. This means that this protocol is useful if the expected harm from

high-cost Sybils ($\delta\gamma H$) does not greatly exceed the benefit B provided by the application.

One problem with this protocol is that it may encourage Sybil attackers that have no interest in subverting the application protocols, but that are interested in being paid to reveal their presence. In this case, the authors propose running the protocol with frequency and unpredictability such that the cost of maintaining a Sybil exceeds the potential profits. The authors also show that if there is a cost E to enter each identity, having $R < E$ makes launching a Sybil attack not profitable.

We consider that a game-theoretic approach is an interesting way to address this problem, but it introduces a complex modeling problem. First, a digital currency must be established, and second, utility functions, benefits and costs must be modeled in a way compatible with the currency system. It is not clear that both of these problems can be solved in the context of a Sybil attack against a DHT-type system (which is not the purpose of the authors). It has been shown that these issues are common to the application of game theoretic approaches to distributed computer systems in general and that they are difficult to solve [Mahajan et al. 2004].

3.12 Discussion

Table I summarizes and provides a comparison of the defenses against Sybil attacks.

Table I: Comparison of defenses against Sybil attacks.

Technique	**Authors**	**Advantages**	**Disadvantages**
Certificates signed by a trusted authority, possibly payed.	Castro et al. [2002]	Allows good control over who is allowed to join the system.	Introduces processing and administrative overhead. Puts barriers to legitimate nodes trying to join the system. Certificate revocation may be costly.
Distributed registration.	Dinger and Hartenstein [2006]	Does not put barriers to enter the system. Decentralized.	Does not protect from attacks involving a large number of IP addresses. The proposed protocols introduce possibilities for new attacks.
Use of physical network characteristics to identify nodes.	Wang et al. [2005]	Does not put barriers to enter the system.	Depends on network measurements that can change over time for the same node, thus failing to provide a consistent identity. Changes to the network measurement infrastructure may invalidate the identity of all nodes.

Continued on next page

Table I – Continued

Technique	Authors	Advantages	Disadvantages
Use of network coordinates to group nodes.	Bazzi and Konjevod [2005]	It can prevent Sybil attacks consisting of a single node reporting multiple identities, or many nearby colluding nodes.	Does not prevent a Sybil attack where the attacker controls a sufficiently large number of nodes in multiple network positions. May require a trusted network measurement infrastructure.
Use of network coordinates to differentiate nodes.	Bazzi et al. [2006]	Simple algorithms. Hop-count distance is not used a stable node identifier, but only to tell physically separated nodes apart.	Concept of physical neighborhood not clear for overlay networks. Does not prevent a Sybil attack where the attacker controls a sufficiently large number of nodes in multiple network positions. Requires appropriately placed trusted beacons.
Use of bootstrap graph based on social network.	Danezis et al. [2005]	Does not put barriers to enter the system. Decentralized.	Introduces significant overhead. Requires social relationships between participants. Has not been shown to scale beyond 100 honest nodes.
Use of social network.	Yu et al. [2006] [2008]	Decentralized. Scalable to large number of nodes. Relatively low barrier to entry. Simulation results show good resistance to Sybil attacks (improved in SybilLimit over SybilGuard).	Suitable only when a social network is feasible. Offline sharing of symmetric keys might be difficult in practice.
Computational puzzles.	Borisov [2006]	Can effectively limit the number of Sybil identities that computationally limited adversaries can generate. Decentralized.	Forces honest nodes to continually spend computing resources solving puzzles. Nodes can choose their ID, which facilitates targeted attacks. It may be difficult to select the appropriate puzzle's difficulty in a heterogeneous environment.

Continued on next page

Table I – Continued

Technique	Authors	Advantages	Disadvantages
Computational puzzles generated hierarchically.	Rowaihy et al. [2007]	Can effectively limit the number of Sybil identities that computationally limited adversaries can generate.	Requires a centralized online trusted authority. Requires reliable nodes in the upper levels of the certification hierarchy. Forces honest nodes to continually spend computing resources solving puzzles. Nodes can choose their ID, which facilitates targeted attacks. It may be difficult to select the appropriate puzzle's difficulty in a heterogeneous environment.
Use of economic incentives (game theory).	Margolin and Levine [2007]	In principle, it is a decentralized solution.	Requires the implementation of a currency, which may carry significant security issues with it. Introduces a difficult modeling problem, since it requires expressing all costs and utilities in terms of a currency. Detects, but does not prevent attacks.

As we can see in Table I, defenses against the Sybil attack belong to one of the following categories: (1) centralized certification, (2) distributed registration, (3) physical network characteristics, (4) social networks, (5) computational puzzles, and (6) game theory.

Centralized certification [Castro et al. 2002] provides the most effective Sybil defense, but it relies on two assumptions: that it is possible to set up a central certification authority trusted by all participating nodes, and that such an authority is able to accurately detect Sybil attackers. These assumptions are strong, but they may be realized in certain contexts such as registration-based services like Skype [Skype Limited 2008]; collaborative web hosting [Pierre and van Steen 2006], where nodes may register with a trusted broker; or systems where credentials issued by a commercial certification authority are acceptable.

A major inconvenience of this approach is that the cost of maintaining the central authority may be high, and it constitutes a clear target for attackers. Having an offline CA makes attacks more difficult and reduces their impact, but it is certainly not a perfect solution. In addition, centralized management of identities carries with it the burden of certificate revocation when identities are no longer valid for any reason, including online detection of a Sybil attack.

Distributed registration [Dinger and Hartenstein 2006], while unable to fully prevent Sybil attacks, tries to mitigate their effects while relaxing the assumptions made by centralized certification. The problem with decentralized approaches is that, like DHTs themselves, they are susceptible to attacks because of the limited view of the system and the lack of mutual trust among participating nodes. The approach by Dinger and Hartenstein [2006], in particular, is susceptible to attacks where a node has multiple IP addresses, which is not difficult to produce.

Defenses that use physical network characteristics assume that these characteristics are

difficult or impossible to fake, and that Sybil attacks are launched from either a single computer, or a set of computers with very similar characteristics. They can try to identify nodes [Wang et al. 2005], or just differentiate nodes with sufficiently different network characteristics [Bazzi and Konjevod 2005; Bazzi et al. 2006]. One advantage of these methods is that they are transparent to participating nodes, so they introduce fewer barriers to entry for honest nodes. However, they require a trusted online infrastructure to produce network measurements, which introduces requirements similar to the centralized certification approach. Since this infrastructure must be online, it may be even more vulnerable than a certification authority to many targeted attacks. Furthermore, an attacker could beat this defense by controlling nodes placed at different geographical locations. It should also be noted that stable network coordinates are difficult to produce [Szymaniak et al. 2004; Szymaniak et al. 2008], which introduces a trade-off between security and resilience to variable network conditions. This trade-off is especially difficult if network coordinates are used to identify nodes.

Defenses that use social information [Yu et al. 2006; Yu et al. 2008; Danezis et al. 2005] have been shown to be very effective at limiting Sybil attacks. In this approach, every node acts like a small certification authority that is very accurate at identifying attackers. The main disadvantage of this approach is that it requires participating nodes to form a reasonably connected online social network. This is possible to realize in applications that are inherently social, such as those that make use of instant messaging. Similar ideas have also been applied to file-sharing applications [Pouwelse et al. 2007]. An important disadvantage of the most effective social-based approaches is that they require offline symmetric key distribution among socially connected peers, but the practicality of a procedure to achieve this has not yet been established.

Computational puzzle approaches try to limit the number of fake identities generated by each physical attacker by having honest nodes request other nodes to solve puzzles that require a significant amount of computational resources. The idea is that this limits the capability of an attacker to generate multiple identities. Douceur [2002] proved that this kind of direct validation works only if all identities are validated simultaneously. Since this is impossible in a dynamic system where nodes can join and leave at any time, the proposed Sybil defenses [Borisov 2006; Rowaihy et al. 2007] require a periodic revalidation of all identities. One disadvantage of this approach is that honest nodes must continuously spend valuable computing resources to remain in the system, and the complexity of the puzzles must be selected such that the most limited honest node can be able to solve them and keep some free capacity for other operations. It has been shown that heterogeneity in a practical peer-to-peer system can be significant, with the most powerful nodes having several orders of magnitude more CPU power than the least powerful nodes [Anderson and Fedak 2006]. Even more important, using computational puzzles alone does nothing to control node identifier assignment, which means that malicious nodes may select their own identifiers and thus control a region of the overlay with a relatively small number of nodes.

Finally, game theory has also been proposed to deal with Sybil attacks [Margolin and Levine 2007]. Although intellectually appealing, this approach has the fundamental problem that it assumes that each Sybil identity is rational. However, Sybil identities are not independent from each other; the only malicious entity that may be considered rational in practice is the adversary. It is not likely that an attacker will give away Sybil identities for an economic reward, unless its specific purpose was to get such a reward, in which case

the attacker would not be attacking the system that is being protected, but the defense itself. This awkward model shows the main difficulty of applying game theory to distributed systems, which is how to model the system as a set of players with utility functions. Most game theory approaches use a currency to express utilities, but the utility for an adversary and the costs for honest participants are generally too complex to be expressed with a single number. Moreover, implementing a secure currency in a decentralized system is subject to many security issues, possibly including Sybil attacks themselves. We believe that this kind of approach has an interesting theoretical value, but is not likely to be effective in practice to defend against a Sybil attack in a DHT.

4. ECLIPSE ATTACK

Nodes in an overlay network have links to a few other peers commonly referred to as neighbors. If an attacker controls a sufficient fraction of the neighbors of correct nodes, then some correct nodes can be "eclipsed" by the malicious nodes. This attack is also known in the literature as routing table poisoning. An Eclipse attack can be used to facilitate other attacks, especially routing and storage attacks.

Sit and Morris [2002] were the first to study this attack in the context of DHTs. They state that systems in which the neighbors do not have special verifiable requirements are the most vulnerable to this type of attack. The easiest way to exploit this weakness is through incorrect routing updates. For example, as we explained in Section 2, the top levels of Pastry's routing tables require a common prefix of only a few digits. This increases the number of valid identifiers that an attacker can supply during routing table updates in comparison to systems that impose strong constraints, such as Chord [Stoica et al. 2003]. This particular attack can make the fraction of malicious entries in routing tables of honest nodes to tend towards one, since the number of sources of malicious entries can increase with every update.

Another possible attack scenario is the subversion of the network proximity measurement mechanism. For example, Hildrum and Kubiatowicz [2003] show that an attacker may reduce its apparent distance by using a colluder present in a shorter route to forward a spoofed response to a heartbeat message. Castro et al. [2002] state that the measurements can also be subverted with indirection mechanisms such as mobile IPv6, or if the attacker controls a large infrastructure, such as an ISP or a large corporation. It should also be noted that, if one assumes that large-scale subversion of network proximity measurements is not possible, then network proximity can be used to help prevent the Eclipse attack [Hildrum and Kubiatowicz 2003].

Another obvious scenario for attacking proximity-based DHTs, is to place many malicious nodes in each other's proximity. From there on, they can easily collude and attack other nearby nodes.

An Eclipse defense can be considered successful if the fraction of malicious entries in the routing tables of honest nodes does not differ much from the fraction f of malicious nodes in the system, as this is the expected fraction of malicious entries in any random sample of the nodes provided that identifiers are randomly generated. However, routing using a single path may easily be unlikely to succeed. For example, if $f = 0.25$ and the path length is 5, the probability of successful routing is $(1-0.25)^5 \approx 0.24$, which would be unacceptable in most applications. Hence, the routing-table update protocols are always complemented with some form of redundant routing, which is the basis to defend against

the routing and storage attacks that we discuss in Section 5.

The most common attack model used by the proposed solutions is one in which the malicious nodes collude and try to maximize the poisoning of the routing tables of all honest nodes by always supplying malicious references in the routing table update protocols. However, this is not necessarily the only possible scenario. For example, the adversary may try to attack only a small subset of the nodes, a specific key, or specific rows of the routing tables. The attacker may also try to disseminate poisoning in a slow way, by attacking nodes sequentially and behaving correctly most of the time. None of the proposed solutions are evaluated against these more subtle attacks. Singh et al. [2006] discuss localized attacks, which require an honest node to be surrounded by malicious nodes in terms of network distance. They state that defending against such attacks remains an open problem.

4.1 Castro et al.

To defend against Eclipse attacks in systems that exploit network proximity, such as Pastry or Tapestry, Castro et al. [2002] propose the use of two routing tables. One table (which we refer to as the *optimized routing table*) exploits the potentially vulnerable network proximity information while the other (referred to as the *verified routing table*) contains only entries that can be verified and do not take into account network proximity. In normal operation they use the optimized routing table and resort to the verified routing table using a secure routing primitive in case of failures. Pastry's normal routing table entry at level l and for domain d for a node with identifier i contains a reference to a node that shares the first l digits with i and has the value d in digit $l+1$. Since several nodes may satisfy this criteria, the one with the shortest network distance is selected. For example, in the node with identifier `26AE9`, the entry $(2,5)$ is filled with the node that has shortest network distance among those whose identifier has the form `265XX`. For each entry (l,d), the verified routing table of node i contains the node with identifier closest to the point that shares the first l digits with i, has the value d in digit $l+1$, and has the same remaining digits as i. Taking our example, the entry $(2,5)$ of node `26AE9` will be filled with the node numerically closest to `265E9`.

The authors define a routing failure test to determine if routing with the more efficient network proximity-based routing table fails, and a secure routing primitive to be used with the verified routing table. We describe these in Section 5.2.

The authors did not perform any experiment to evaluate the behavior of this scheme. However, as shown by Condie et al. [2006], attacks will progressively poison the optimized routing table. The result is that after some time, most of the routing would have to be done using the verified routing table, with the additional overhead of first trying with the poisoned, optimized routing table and the routing failure test. Under such a scenario, it would be better to use only the verified routing table with redundant routing and discard the routing failure test.

4.2 Condie et al.

Condie et al. [2006] propose a defense for the Eclipse attack based on induced churn. They assert that the dual routing table strategy proposed by Castro et al. [2002] is vulnerable to the fact that the poisoning in the optimized routing table tends to increase over time. Condie et al. address this by periodically resetting the optimized routing table to the contents of the verified routing table, and use the optimized routing table to perform lookups in most cases. The verified routing table is used in conjunction with redundant routing in lookups

that assist other nodes to join the overlay and to maintain the verified routing table itself. This approach is useful provided that the poisoning of the optimized routing table increases slowly over time. To keep this increase slow, they limit the rate at which routing tables are updated, since this is the main source of this increase.

To prevent attacks exploiting knowledge of how routing tables are updated over time, an unpredictable identifier assignment method is introduced. At each reset, every node gets a new random identifier, positioning itself in a different part of the identifier space. The authors affirm that if good nodes move continuously, then it is difficult for the malicious nodes to attack them in the same way after every reset.

The system uses a trusted *timed randomness service* to obtain random numbers. This service periodically generates a new random number placed into a signed certificate that includes both the number and the timestep at which it was generated. When nodes update their routing tables, they drop entries whose identifiers were generated using old random numbers.

It is important that not all nodes churn periodically at the same time because this would result in a very unstable and highly loaded system around the global churn time. It would also move uniformly from less poisoned to more poisoned states, which can be exploited by malicious nodes to select the appropriate time to attack. To prevent this, the authors define a *staggered churn*, in which the set of nodes is partitioned into groups, with each group churning at a different timestep. Churn groups are defined according to the prefix of the IP addresses of the nodes.

Another optimization is the precomputation of the next routing state before the reset. This requires each node to know its new identifier before performing the reset. This precomputation allows a smoother transition when a node moves.

The authors evaluate their proposal by simulating a network of 50,000 nodes with different values for the fraction of malicious nodes. The results show that their approach significantly reduces the fraction of poisoned routing table entries and increases the probability of successful routing. They measure the probability of successful routing and show that redundant routing with the verified routing table alone does not provide any significant benefit when the fraction of malicious nodes exceeds 5%. Routing table reset without redundant routing improves results significantly, but it is the combination of both techniques that produces the best results. With a fraction of 15% of malicious nodes, the combined approach has a probability of successful routing barely below 1, while routing table reset alone achieves a probability of approximately 0.35 and redundant routing alone around 0.15. With a fraction of 25% of malicious nodes, the combined approach achieves a probability of approximately 0.8, a routing table reset about 0.3 and redundant routing approximately 0.1.

This method seems to provide an adequate defense against the Eclipse attack. However, the induced churn causes the stored data to be moved constantly, making the approach more susceptible for storage attacks (which we discuss in Section 5). For each churn epoch, every data item is moved at least once, which makes the overhead $\Omega(K)$ per epoch, where K is the number of stored keys. The authors study the overhead of the system, but only consider the costs of updating and maintaining routing tables, and liveness tests among nodes. For many applications, the overhead caused by moving the keys and their associated data will dominate bandwidth consumption, especially in a replicated environment. In addition the system has the administrative cost of an online trusted randomness service,

which would likely introduce a centralized component that may not perform appropriately for all participating nodes.

4.3 Hildrum and Kubiatowicz

Hildrum and Kubiatowicz [2003] propose a solution for Pastry and Tapestry that assumes a trusted mechanism for measuring network distance (e.g., by pinging). Their basic idea is to fill each level l of the routing table with the closest neighbors in terms of network distance among the neighbors that share the first l digits. The result is that each entry of the routing table has the r entries closest in network distance. For example, in a node with identifier `26AE9`, the entry $(2,5)$ is filled with the r closest nodes in network distance that have the form `265XX`.

The reason why this approach is resistant to an Eclipse attack is the fact that if the fraction of malicious nodes is sufficiently small, then it is difficult for the malicious nodes to be the closest in network distance to a large fraction of the good nodes. The redundancy introduced in the routing table makes this even more difficult.

The authors test their algorithm in a simulation of 50,000 nodes with varying fractions of malicious nodes that return information only about other malicious nodes. The underlying topology used was a grid, where overlay nodes where chosen at random.

The results show that using a value of $r = 3$ with a fraction of malicious nodes equal to 50% provides a fraction of bad routing table entries of less than 15%, and changing r to 5 provides less than 10% of bad entries.

The main advantage of this system is its simplicity. However, it depends on trusted and stable network distance measurements, and the authors do not mention how these can be implemented in practice. Moreover, the underlying grid topology used in the experiments may produce distance measurements that are not realistic in practice. According to Singh et al. [2006], this defense works only in small overlays where nodes are sufficiently separated.

4.4 Singh et al.

Singh et al. [2006] acknowledge the need for an Eclipse defense that allows network proximity optimizations, but they state that the defense proposed by Hildrum and Kubiatowicz [2003] is practical only if all pairs of nodes are sufficiently separated in the network, and conclude, based on simulation experiments, that this defense may be effective only for small overlays.

They propose a defense based on the fact that during an Eclipse attack, the in-degree of the malicious nodes must be higher than the average in-degree of nodes in the overlay. Thus, one way to prevent an Eclipse attack is forcing honest nodes to select only nodes with an in-degree below a certain threshold.

Another related attack consists of malicious nodes exhausting the in-degree of honest nodes by pointing to them. Thus it is also necessary to bound every node's out-degree.

The degree bounds are enforced by a distributed auditing process. This system is implemented by having each node x maintain a list of nodes that have x as a neighbor. This list is referred to as the *backpointer set* of x. Node x forwards traffic only from nodes in its backpointer set. Periodically, x challenges its neighbors requesting their backpointer list. If the response has more entries than the in-degree threshold, or x is not included, then x removes the node from its neighbor set. A similar procedure is applied to the members of x's backpointer list to ensure that their out-degree is below the threshold and includes x.

The only way this system can work is if the node being challenged does not know the identity of the challenger. To achieve anonymity, nodes forward their challenges through intermediate nodes called anonymizers. Challenges are timed randomly to complicate the detection of the identity of the challenger.

When node x wishes to challenge node y, the anonymizer is selected randomly among the ℓ closest nodes to the hash $H(y)$. Since node IDs are assumed random, the expected fraction of malicious nodes in this subset is the same as the fraction f of malicious nodes in the overlay.

Since it is possible to have malicious anonymizers, the authors propose to label a node as malicious if it answers less than k out of n challenges correctly. With this approach, the probability of an honest node to be considered malicious (false positive) is

$$\sum_{i=0}^{k-1} \binom{n}{i} f^{n-i}(1-f)^i$$

and the probability that a malicious node passes the audit undetected is

$$\sum_{i=0}^{k-1} \binom{n}{i} [f+(1-f)c/r]^i [(1-f)(1-c)]^{n-i}$$

with c being the probability of the malicious node answering the challenge and r the ratio of the size of the true set versus the maximum allowed. As an example, with $n = 24$, $k = 12$, $r \geq 1.2$, and assuming $f \leq 0.25$, the probability of a false positive is around 0.2% and malicious nodes are detected with a probability of at least 95.9%.

The authors evaluate their strategy through simulations. In their experiments the fraction of malicious nodes is set to 20%. They first evaluate the effectiveness of an ideal degree-bounding strategy independently of how it is enforced. The results show that when the bound is tight, the fraction of malicious nodes in the routing tables of honest nodes is 0.24 for all overlay sizes from 1000 to 20,000 with a degree bound of 16 nodes per routing table row. If the bound is loose, this fraction grows significantly with the size of the overlay (0.24 for 1000 nodes and 0.45 for 20000 nodes with a degree bound of 48).

To evaluate the auditing technique they simulate a network with 2000 nodes, $n = 24$ and $k = 12$. The degree bound is set to 16 per routing table row and backpointer set row, with malicious nodes having no bounds. At one random instant in every two-minute period, a node audits all its overlay neighbors. In a static membership scenario, the results show that every node in the system reaches an in-degree equal or below the allowed bound 16. This means that the procedure is able to detect malicious nodes that violate the in-degree threshold, and thus reduce the scale of the attack.

The experiments showed that the auditing technique is able to reduce the fraction of malicious entries in the first row of the routing tables to less than 30% and the general fraction to less than 25%. These results are valid with stable membership, as well as with a churn of 5% and 10% per hour. For higher rates of churn, the fraction of malicious entries grows with the churn rate. The results suggest that there is a trade-off between the fraction of malicious entries in routing tables and the auditing rate. With respect to false positives, the auditing scheme exhibited only a 10^{-3} false positive rate.

One advantage of this defense is that it does not assume the existence of services that are difficult to implement, and does not require cryptography. However, the experimental results show that the system is effective only when the degree bound is small, which results

in increased lookup times in the absence of attacks.

4.5 Awerbuch and Scheideler

Awerbuch and Scheideler [2006] propose a secure DHT scheme that introduces the concept of regions in a $[0,1)$ identifier space. For each node that joins the overlay with identifier x, the k-region $R_k(x)$ is a portion of the identifier space of size closest to k/n from above that contains x, where n is the number of honest nodes. Routing is done from region to region (for more details, see Section 5.10).

In this system, every time a new node joins the overlay, it receives a fresh random identifier generated by a group of participating nodes using a verifiable secret sharing scheme. A malicious node could try to continuously join and leave the system until it receives some desired identifier. This allows an attacker to concentrate many malicious nodes in one or more regions, thus polluting routing tables of honest nodes.

The defense against this type of attack is a protocol referred to as the *cuckoo rule*. This protocol establishes that when a new node joins the overlay, all nodes in the k-region of the new identifier must leave the system and rejoin with new random identifiers. The authors prove that this protocol guarantees that regions are balanced in the number of nodes within a factor close to $(1+\varepsilon+1/k)$, and that honest nodes are a majority in every region as long as $\varepsilon < 1-1/k$, where ε is such that εn is the number of malicious nodes.

In [Awerbuch and Scheideler 2007], the authors propose an extension to the cuckoo rule, called the *cuckoo&flip* rule, which protects the overlay not only from join-leave attacks, but also from a combined attack in which the adversary is able to remove from the overlay a limited number of honest nodes by means of a typical denial-of-service attack while a join-leave attack is performed.

In the cuckoo&flip rule, whenever a node y leaves the system, a randomly selected region R where the node y resides is flipped with a region R' randomly selected from the whole ID space; that is, all nodes in R are moved to R' and *vice versa*. Then, all nodes in R must leave and join using the cuckoo rule.

A previous work by Scheideler [2005] presents a simple protocol called the k-rotation rule that can guarantee that if nodes are arranged in a ring, any sequence of $O(\log n)$ consecutive nodes contains a majority of honest nodes. However, this protocol does not take into account the concept of nodes being laid out in an identifier space and consequently is unable to establish any kind of balancing in such a space.

Another previous work by the authors [Awerbuch and Scheideler 2004] proposes to maintain malicious nodes spread at random locations in the identifier space by using a distributed random ID generation scheme and enforcing limited lifetimes for participating nodes, which results in a continuous churn similar to the approach by Condie et al. [2006]. This is combined with a Chord-like routing protocol, but with messages forwarded between regions of nodes instead of simple nodes.

4.6 Discussion

The most basic defense against the Eclipse attack is to constrain the identifiers of nodes that can be used in routing tables, as is done in Chord [Stoica et al. 2003]. This is valid only if node identifiers are random and stable, and malicious nodes are spread over the identifier space. To achieve these conditions, the simplest approach is using stable node identifiers issued by a central authority. As an alternative, Awerbuch and Scheideler [2006] propose an approach based on inducing churn every time a node joins, assigning new

random identifiers to the nodes participating in the churn.

On the other hand, using properties of node identifiers as the sole criterion to select routing table entries prevents performance optimizations such as proximity neighbor selection. This type of optimization can be easily implemented in systems like Pastry because they place weak requirements on the top levels of their routing tables. The result is that many nodes satisfy the identifier requirements, and network measurements can be used to select optimal neighbors.

The problem with this is that malicious nodes can easily find their way into the routing tables of honest nodes by subverting routing table update protocols or network measurement infrastructure. Condie et al. [2006] report that having 15% of malicious nodes in the overlay results in around 80% of malicious entries in standard Pastry routing tables.

Most of the literature about the Eclipse attack is focused towards defenses that try to preserve network optimizations with Pastry-like routing tables. The simplest approach is probably the use of redundant routing table entries [Hildrum and Kubiatowicz 2003], where it is expected that some the entries will be honest and thus sufficient to allow successful redundant routing.

Another approach is to combine optimized routing tables with constrained routing tables [Castro et al. 2002]. The main drawback of this approach is that the optimized routing table will be progressively poisoned, and the constrained table will eventually be used most of the time, with the extra overhead of using a poisoned table and a complex failure detection procedure.

Condie et al. [2006] propose to augment this method by forcing each node to periodically leave the overlay and rejoin with a new identifier and two fresh routing tables, one of which will be optimized until the node has to rejoin again. This is similar to the approach by Awerbuch and Scheideler [2004].

Another way to defend against this attack is to control the number of incoming and outgoing links per node. The reasoning is that the degree of malicious nodes is expected to be greater than the degree of honest nodes. The approach by Singh et al. [2006] has the advantage of being fully decentralized, but it requires small degree bounds which results in an increase of the lookup time in the absence of attacks.

It can be seen that defending against the Eclipse attack involves a trade-off between performance and complexity. Moreover, an Eclipse defense can be considered successful if it can guarantee that the probability of a routing table entry being malicious is equivalent to the general fraction of malicious nodes in the overlay, which means that these techniques are not enough to guarantee proper operation of the DHT unless they are combined with other mechanisms such as redundant routing.

Table II summarizes and provides a comparison of the defenses against Eclipse attacks.

Table II: Comparison of defenses against Eclipse attacks.

Technique	**Authors**	**Advantages**	**Disadvantages**
Constrained routing tables. Entries are chosen based solely on node ID.	Stoica et al. [2003]	Simplicity. Effective against Eclipse attacks by definition.	Prevents performance optimizations based on network measurement. Requires stable random node identifiers.

Continued on next page

Table II – Continued

Technique	Authors	Advantages	Disadvantages
Region-based redundant routing tables, consensus-based node ID assignment, and on demand churn when nodes join.	Awerbuch and Scheideler [2006]	Guarantees that malicious nodes are spread over the ID space and thus do not pollute region-based routing tables. Does not depend on stable identifiers.	Complex algorithms. Likely to prevent performance optimizations based on network measurements.
Use of two routing tables. One optimized with network measurements and the other constrained and used in case of a test failure.	Castro et al. [2002]	Does not assume the existence of services that may be difficult to implement.	Does not address the issue of progressive poisoning of the optimized routing table. The routing failure test is not very accurate, is sensitive to nontrivial parameters, and is vulnerable to attacks. Requires stable random node identifiers.
Resetting of optimized table entries and continuous induced churn.	Condie et al. [2006]	Limits the progressive poisoning of the optimized routing tables.	The induced churn introduces a significant overhead. The routing failure test is not very accurate, although this is mitigated with the resetting of routing tables. The administration of the trusted randomness service may be difficult in practice.
Use of redundant routing table entries based on network proximity.	Hildrum and Kubiatowicz [2003]	Simplicity.	Depends on trusted and stable network distance measurements, which may be difficult to implement in practice.
Control of the in-degree and out-degree of overlay nodes via anonymous auditing.	Singh et al. [2006]	Does not assume the existence of services that may be difficult to implement. Does not require cryptography.	Experimental results show that the system effectively limits the poisoning of the optimized routing tables only when the degree bound is small. This results in an increase of the lookup time in the absence of attacks.

5. ROUTING AND STORAGE ATTACKS

Sybil and Eclipse attacks do not directly disrupt the DHT, but they can be used as a vector to permit or amplify future attacks. Such attacks may attempt to prevent a lookup request from being successful. For example, an attacker may refuse to forward a lookup request. It may forward it to an incorrect, non-existing, or malicious node. Finally, it may pretend to be the node responsible for the key. Another possibility for an attacker is to route requests correctly, but deny the existence of a valid key or to provide invalid data as a response. These attacks are generally classified as routing attacks, which try to disrupt routing, and

storage attacks, which attempt to provide bogus responses to queries. In this section we study several solutions that have been proposed to deal with this type of attacks.

5.1 Sit and Morris

Sit and Morris [2002] were again the first to study these attacks. They proposed a general solution based on three techniques. The first one requires to use iterative routing so that the requester can check the lookup progress and detect anomalies such as responses that get "far" from the requested key in overlays such as Chord, Pastry and Tapestry. Second, one should assign keys to nodes in a verifiable way, so that it is difficult for a node to claim responsibility for a specific key. In systems where keys are assigned to the closest node in the identifier space, it is enough to derive node identifiers in a verifiable way. The authors cite as an example the method used by Chord, in which the identifier is based on a cryptographic hash of its IP address and port number. Third, they propose to use identities based on public keys. This solution allows other nodes to check the origin of messages and the validity of their content.

Regarding the possibility of attackers refusing to serve or store keys, Sit and Morris assert the need for replication and provide some general guidelines, but no specific defense mechanisms. Namely, replication must be implemented in such a way that no single node is responsible for replication or for facilitating access to the replicas. Instead, all the nodes holding replicas must ensure that the required number of replicas is maintained at all times. Clients must consult at least two replicas in order to be sure of the lookup results.

The main contribution of this work is that they present the security issues affecting DHTs and provide guidelines to address them. However, the proposed guidelines are not specific enough to result in the construction of a secure system.

5.2 Castro et al.

Castro et al. [2002] propose a replica function that maps a key to multiple nodes. In the case of Pastry, a key k is replicated to the members of the leaf set of the node responsible for k according to the standard Pastry protocol.

The authors reject the use of checked iterative routing because it doubles the cost with respect to recursive routing, and, to be effective, it requires tests at each hop. Their experiments show that the possibility of false positives in these tests adds 2.7 extra hops on average to the routes without significantly increasing the probability of successful routing. As an alternative, they propose a secure redundant routing primitive for Pastry that takes a message and a destination key, and ensures with high probability that at least one copy of the message reaches all correct replicas. As described in Section 4.1, they implement this by using two routing tables: an optimized table built taking network proximity into account, and a verified routing table.

They normally use the optimized table, and resort to redundant routing with the verified routing table when optimal routing fails. They introduce a routing failure test to decide if optimal routing failed. This test compares the average distance d_r between consecutive node identifiers in the set of replicas returned by the lookup with the corresponding average distance d_p for the neighbor set of the requester. A result is rejected if $d_r < d_p \times \gamma$, where γ is a constant that controls a trade-off between the probability of false positives and false negatives.

This test, however, is subject to a number of attacks, most notably, a *node suppression attack* in which the adversary has enough malicious nodes near the sender or the receiver,

and makes its nodes close to one of these locations when other nodes leave, thus altering the average distances measured by the test and increasing either the probability of false positives or false negatives. The attacker can alternate between the two modes, making detection of this attack even more difficult. The authors ran simulations targeting a probability of false negatives of 0.001, and show that for a fraction of colluders equal to 0.3, using $\gamma = 1.23$ and 256 samples to compute the distances in the test, the result is that the probability of false positives is 0.77 under a node suppression attack, and 0.12 without the attack.

Redundant routing is implemented by sending copies of the message over diverse routes to the different replicas of a key. This is straightforward if the replicas are distributed uniformly over the ID space, but it is not sufficient if the replicas are in the neighbor set of the node originally responsible for the key. In this case, the lookup is sent through r different members of the sender's neighbor set using the verified routing table. Any honest node that receives the message and has the node responsible for the key in its neighbor set returns its own ID certificate (issued by a trusted certification authority). The sender collects the certificates and makes a list L with the $l/2+1$ nodes numerically closest to the key on the left and on the right and marks them as *pending*. After a timeout, or after the r responses are received, the sender sends the list L to the nodes in L marked as *pending*, and marks them as *done*. Any honest node that receives the list sends the original message to the nodes in its neighbor set that are not in L, or it sends a confirmation to the sender if there are no such nodes. As a result, the sender collects new certificates, updates L, and sends L again to all nodes marked as *pending*. The definitive replica set is computed from L once the procedure has been executed three times, or all nodes in L have sent a confirmation.

The authors state that with this procedure, the probability of reaching all correct replicas is approximately equal to the probability that one of the original r messages is delivered through a route without malicious nodes, which is $(1-f)^{1+\log_b N}$, where b is the base for node identifiers, f is the fraction of malicious nodes, and N is the expected number of nodes in the overlay. It is not specified how to verify the integrity of the replicas, but this can be accomplished using techniques such as majority voting.

A measure proposed to reduce the use of redundant routing and to address storage attacks, is to use *self-certifying data*, that is, data whose integrity can be verified by the client with mechanisms such as digital signatures. This allows the client to resort to redundant routing only when the integrity check fails or there is no response. The technique can be extended to mutable objects by storing *group descriptors* in the overlay. A group descriptor for a data item contains the public keys and IP addresses of the item's replica holders, and it is signed by the item's owner. To maintain consistency, the authors propose to use a Byzantine-fault-tolerant replication algorithm, such as BFT [Castro and Liskov 2002].

Note that the number of redundant paths may be large because a single malicious node completely invalidates all the paths in which it is included. Hildrum and Kubiatowicz [2003] show that to make the probability of successful routing a constant, the number of paths must be polynomial in the number of nodes, and this is under the assumption that all paths are independent, which is not guaranteed. In addition, the performance is highly dependent on how frequently the optimized routing table is used. The routing failure test itself depends on the accuracy of the returned distances, which may be controlled by malicious nodes and can thus lead to a significant increase of redundant routing. Moreover, as we have seen in Section 4, under an Eclipse attack the optimized routing table is easily

poisoned to the point that the expensive redundant routing will be used most of the time.

5.3 Hildrum and Kubiatowicz

Hildrum and Kubiatowicz [2003] show that the multiple path approach proposed by Castro et al. [2002] has an asymptotically low probability of success. Using the formulation from Castro et al. [2002], the probability that all r routes fail is more than

$$\left(1-(1-f)^{1+\log_b n}\right)^r \approx \exp\left(-r \cdot n^{\frac{\ln(1-f)}{\ln b}}\right)$$

where f is the fraction of malicious nodes and b is the base for Pastry-like node identifiers. To make this a constant, r must be a polynomial in the total number of nodes.

The authors propose two techniques for fault-tolerant routing using iterative routing assuming a trusted network proximity measurement service. As mentioned in Section 4.3, each entry in the routing table stores not one neighbor, but $l = O(\log n)$.

With the first proposed technique, the request starts with a list of r level-l nodes. The requester contacts each node and requests their r closest level-$(l+1)$ nodes. The requester then removes duplicates and measures the network distance to the remaining nodes, selects the r closest and repeats the first step until there are no nodes at the next level, which happens when the key is found. This results in a *wide path* that has the property that it requires a message to reach only one good node at each level of the path.

With the second technique, the first step is to select all level-$(l+2)$ nodes from the level-l nodes (they prove that with high probability there is at least one such node), then request the level-$(l+1)$ nodes from the level-$(l+2)$ nodes, and then pick the closest r of these level-$(l+1)$ nodes to be the definitive set of level-$(l+1)$ nodes. These steps are repeated until the key is found.

The authors analyze both solutions and conclude that the first algorithm is simpler and more practical, but its analysis is more complicated and only holds when $fc^2 < 1$, where c is an expansion constant of the network such that $c^2 < b$. The analysis for the second algorithm holds for any value of f.

The authors test the first algorithm in a simulation of 50,000 nodes with varying fractions of malicious nodes that always produce bogus responses. The underlying topology used was a grid, where overlay nodes where chosen at random.

The results show that using a value of $r = 3$ with $f = 0.3$ provides a probability of successful routing of greater than 0.9. Using $r = 5$ and $f = 0.4$ provides a probability of success greater than 0.95.

We observe that this scheme has the advantage of simplicity, and the wide paths give much greater fault tolerance than multiple paths without sacrificing the benefit of network proximity. However, the system depends on secure and stable network distance measurements, which may not be readily available in practice. Moreover, the underlying grid topology used in the experiments may produce distance measurements that are not realistic.

5.4 Sánchez et al.

Sánchez Artigas et al. [2005] propose a methodology to defend against routing attacks by augmenting existing overlays with independent paths. Independent paths are presented as an alternative to multiple non-independent paths like those proposed by Castro et al. [Castro et al. 2002] because a single malicious node is able to invalidate only one path, while

in the non-independent case, it is possible for a malicious node to disrupt multiple paths.

The authors applied their methodology to build Cyclone, which is based on Chord. Cyclone partitions the nodes so that the finger tables of a node contain links only to other nodes that share the p rightmost bits of the m-bit identifier. The result is that the system is divided into $r = 2^{m-p}$ independent Chord rings. The successor lists do not have this restriction and can be used as the first or last hop in a lookup. The independent paths are realized by routing through the r independent Chord rings.

The authors tested Cyclone on simulation experiments. They first compared the average path length of Cyclone with standard Chord in networks from 128 to 1024 nodes and found that both exhibited nearly the same average path lengths.

Their second experiment evaluated the resiliency of Cyclone to malicious nodes. A lookup is considered to have failed if no messages arrive at the destination from the r independent paths. The results show that the probability of lookup failure decreases as the number of independent paths increases. For example, for $N = 1024$ and $r = 8$ with a fraction of 30% of malicious nodes, Cyclone failed 15% of the requests, while standard Chord ($r = 1$) failed 70%.

This approach has the advantage that it leverages existing protocols to defend against routing attacks. The required number of paths may be high because a single malicious node invalidates the path in which it is included. This system, however, provides an improvement over multiple non-independent paths.

The definition of successful routing used in the experiments is useful only in applications where the delivery of the message to the wrong destination does not result in corruption of the application data. However, the topology is suitable to be extended to defend against storage attacks by modifying the routing algorithm to make it aware of multiple replicas and verification mechanisms such as majority voting.

5.5 Ganesh and Zhao

Ganesh and Zhao [2005] propose a defense against an attack where an intermediate node along a lookup path maliciously claims to be responsible for the requested key. They refer to this attack as the *identity attack* and propose a solution for DHTs with Pastry and Tapestry-like routing tables.

The defense is based on *existence proofs*, which are signed certificates that prove the existence of nodes in some range of the identifier space. When a node makes a lookup request, it can use a namespace density estimate to determine if the node claiming to be responsible for the key is likely to be telling the truth. If not, the querier estimates the prefix the responsible node should share with the requested key, and send a verification request to proof managers responsible for that prefix. If a better node exists, it will have signed a recent certificate that allows the the querier to confirm the existence of the attack.

To implement existence proofs, an offline certification authority distributes to each participating node public-private key pairs corresponding to each prefix included in the node identifier. For example, node `2AF3` would receive key pairs for prefixes `2`, `2A`, `2AF` and `2AF3`. Periodically, each node publishes existence proofs for its prefixes signed with the corresponding private key to a number of proof manager nodes. The proof managers for a specific prefix are the participating nodes responsible for a hash of the concatenation of the prefix together with salt values. For example, for prefix `2AF3`, proof managers could be nodes responsible for $H(\texttt{2AF3} - 1)$, $H(\texttt{2AF3} - 2)$, $H(\texttt{2AF3} - 3)$, where H is a cryptographic hash function.

To detect if a lookup response is suspicious, an honest querier checks its local routing table to find the longest prefix column that has all its entries filled. If the length of this column is greater than the common prefix of the requested key and the identifier of the response node, the response is considered suspicious and the querier searches for existence proofs of nodes with a longer common prefix by querying the corresponding proof managers. If proofs are found, the existence of an attack is confirmed.

Reporting existence proofs for all prefixes can be expensive. To reduce this cost, nodes produce proofs only for prefixes in the *cusp* region. The cusp is a region of a node's routing table with a mixture of empty and nonempty entries. The authors show that, with high probability, the cusp has a size ≤ 2 independent of network size, and use a cusp size of 3 in their experiments. The start of the cusp is defined to be the first routing level that contains an empty entry. For example, if the first routing level with an empty entry for node `2AF3BC11` is the routing level corresponding to prefix `2AF`, the node will produce existence proofs corresponding to prefixes `2AF`, `2AF3` and `2AF3B`.

The authors evaluate this system with a variety of simulation experiments and measure its effectiveness using two metrics: the *trigger rate*, which measures how often an attack triggers a verification; and the *verification rate*, which measures how often an attack can be proved. Their results show that the verification rate is 100% in all experiments, but false positive rates were not measured. With an attack model where malicious nodes disturb all possible lookups, including verification requests and the routing of existence proofs en route to proof managers, but without churn, the verification rate is above 90% in a network of 4096 nodes with 20% of malicious nodes and 2 proof managers per prefix. The verification rate drops to 80% when the fraction of malicious nodes is 40%. When churn is added to this attack model, increasing the replication factor for proof managers helps improve the verification rate. In a network of 4096 nodes with an average node lifespan of 6 minutes and 20% of malicious nodes, the verification rate is above 95% with 8 or more proof managers, and approximately 75% with 2 proof managers.

We observe that this system uses redundant routing and storage to ensure that existence proofs, but not data, can be found reliably. The result is that the defense does not really prevent or mitigate attacks, but merely helps detect them. If an existence proof includes the node's identifier and network address, then it may be possible to contact the real responsible node and counter the attack. This would be equivalent to using redundant routing towards a single node holding the data, but does not solve the problem of that single responsible node being malicious. Furthermore, online distribution of public-private key pairs may lead to additional security issues. The implications of malicious nodes receiving prefix private keys are not clear.

5.6 Harvesf and Blough

Harvesf and Blough [2006] propose to place replicas at equally-spaced locations in a Chord ring. They state that this method is more robust than the standard method of placing replicas at consecutive locations because the replicas are accessed using diverse routes instead of a single route. They even prove that this method can produce d disjoint routes if 2^{d-1} replicas are placed in a fully populated Chord ring.

They evaluate this method using simulations. In the experiments, each key K has a replication factor r and is inserted into a Chord ring at the locations responsible for $K, K+M/r, K+2M/r, \ldots, K+(d-1)M/r$, where $M = 2^m$, the size of the identifier space. They compare this method with random replica placement, fixed but non-equally-spaced

replicas, and a variant of the standard consecutive Chord replication approach using multiple routes instead of a single route. Their results show that equally spaced replication provides a greater number of disjoint routes. In an experiment with 1024 nodes, using 20-bit identifiers, a replication degree of 4, and 25% of compromised nodes that do not route correctly, the equally spaced approach produced at least one uncompromised path 98% of the time, just like using randomly-placed replicas, while the Chord variant and non-equally-spaced replication routed successfully only 25-60% of queries.

Their measure of routing success assumes that the data in the system is self-verifying, that is, they assume that having at least one route without compromised nodes is enough to regard a query as successful. This assumption is not realistic unless a protocol to ensure such self-verification is provided. Even with read-only self-certifying data, access to a single replica node is not enough to reliably verify whether a key exists or not, thus, in practice, it is usually necessary to have access to multiple replicas. The authors suggest a voting mechanism as a possibility, but in that case, the reported figure for routing success would be severely reduced.

In addition, the provided proof about disjoint paths is not practical, since it requires a fully populated ring. In practice, the identifier space is selected to be large enough so that the system can grow in size without worrying about a shortage of identifiers.

5.7 Wang et al.

Wang et al. [2007] propose a modification to Chord called Myrmic, based on a mechanism that allows a requester to verify that a node is the correct holder of a given key.

Myrmic uses an offline certification authority to provide random identifiers to nodes, similar to what we discussed in Section 3.1. In addition, it introduces a new online trusted authority called *Neighborhood Authority (NA)*. This *NA* is involved only in membership management events, such as when a node joins or leaves. It is not required for lookups.

The main purpose of the certificates generated by the *NA* is to identify the range of keys for which each node is responsible. A certificate $Cert(x)$ for node x has the following format:

$$\begin{aligned} Cert(x) &= sign_{NA}\{List(x)||issueTime||expireTime\} \\ List(x) &= \{I(pred^{l}(x)),\ldots,I(pred^{1}(x)),I(x),I(succ^{1}(x)),\ldots,I(succ^{l}(x))\} \\ I(x) &= (nodeId(x),IPaddress(x)) \end{aligned}$$

where $||$ is the concatenation operation, $pred^{k}(x)$ is the k-th predecessor of node x, and $succ^{k}(x)$ the k-th successor:

$$\begin{aligned} pred^{k}(x) &= \underbrace{pred(pred(\ldots pred}_{k \text{ times}}(x)\ldots)) \\ succ^{k}(x) &= \underbrace{succ(succ(\ldots succ}_{k \text{ times}}(x)\ldots)) \end{aligned}$$

Node x is responsible for keys in the interval $(pred(x),x]$, with $pred(x)$ and x included in $Cert(x)$.

When a new node joins the system, the *NA* issues a certificate and distributes it to the new node, its l successors and its l predecessors. It also updates the certificates of the successors and predecessors to include the new node, and distributes the updated certificates to their

respective neighbors.

The reason the certificates are copied to the nearest neighbors in the ring is that they can serve as witnesses to the freshness of a certificate. When a certificate is revoked, the neighbors are notified by the *NA*. When a node wants to verify that node x is responsible for key K it first checks the certificate $Cert(x)$ provided by x and verifies that K lies in $(pred(x),x]$; then it obtains copies of $Cert(x)$ from the witnesses, and if a copy $Cert(x)'$ has a more recent issue time, then x fails the test.

The only way a malicious node can claim responsibility for a key using a revoked certificate is by having all its neighbors as colluders. The probability of this case can be adjusted by changing the parameter l.

Node leaves and stop failures are handled by a maintenance protocol that keeps certificates in a consistent state. Nodes periodically try to contact the nodes listed in their own certificate. When a node does not respond it is considered out of the system, the *NA* is contacted, and new certificates are issued so that they reflect the new range for the successor of the left node. It would appear that this is a rather heavy setup in systems that subject to considerable churn.

The lookup procedure in Myrmic is an iterative variant of the standard Chord lookup protocol. Assuming that node y is the next step in the lookup for key K, the querying node x contacts y and requests the certificates of each of its fingers and neighbors. If y or one of the nodes specified in the received certificates is responsible for K, x executes the verification procedure and if the result is correct, the lookup ends. Otherwise, the certificates are added to a circular list. Node x selects the next step from the circular list until the correct node is found. The selection of the next step is different from Chord. In Chord the next step is always the nearest predecessor of the key, while in Myrmic the next step is normally a random neighbor listed in the certificate of the nearest predecessor of the key. An exception is when the distance between K and the nearest successor $succ(x)$ is smaller than a predefined threshold. In this case, a random neighbor listed in the certificate of $succ(x)$ is selected as the next step. The reasoning for the random selection is that it strengthens the protocol against attackers attempting to provide certificates that list colluders as the deterministic next hop.

The lookup procedure fails if all nodes in the circular list are marked as contacted or if the hop count exceeds a limit.

The authors show that Myrmic has three important properties. First, honest nodes always have a correct certificate and a consistent neighborhood view. Second, the probability that the verification procedure fails is small, namely f^{2l} where f is the fraction of malicious nodes. Third, the iterative lookup procedure succeeds with high probability in $O(\log N)$ steps.

An important design goal of Myrmic is to make the *NA* stateless, so that it is easy to replicate. To accomplish this, the *NA* must generate the certificates without maintaining any knowledge about the state of the system. When a new node z joins the network, it contacts the existing node x responsible for the ID of z. Next, z contacts the *NA*. Then, the *NA* obtains x's certificate and builds the definitive neighbor list for z's certificate by contacting x's neighbors.

The authors implemented Myrmic and tested its performance on a PlanetLab [Peterson et al. 2006] testbed with 120 nodes. They used a certificate size of 7 ($l = 3$), and each node sent 500 lookup requests at a rate of 1 message every 3 seconds. The 97th and 90th

percentiles of the lookup times were 346*ms* and 281*ms* respectively and 93% of the lookups were solved within 6 hops plus the verification procedure.

The PlanetLab test assumed that all nodes behaved correctly. In order to evaluate the effect of malicious nodes Myrmic was evaluated in a LAN environment with 1000 nodes and with the assumption that the verification procedure never fails. The results showed that with a fraction with 30% of malicious nodes, only 0.0122% of the lookups failed.

Myrmic has several advantages. From a design perspective, it uses a lookup procedure that seems as reliable as wide paths, and more efficient. From a practical point of view, it is the only system among those reviewed in this survey that has a real implementation tested in a wide-area environment, with reasonable performance results.

The biggest disadvantage of this system is the need for an online central point of trust that, even though it is relatively easy to replicate, in practice would introduce a centralized component. In addition, the real-world evaluation did not study the effect of malicious nodes on latency. The experiments where malicious behavior was studied also did not evaluate the accuracy of the verification procedure, which was assumed to be infallible.

Another important disadvantage is that no mechanisms to prevent storage attacks were evaluated. It is clear that the proposed lookup method securely locates the replica-holding successors of the node responsible for a key, which can be used to determine the true value of the data associated with the key. The implementation of this mechanism would affect the latency results obtained in the experiments.

5.8 Fiat et al.

Fiat et al. [2005] introduce S-Chord, a modification to Chord able to function in a scenario referred to by the authors as *Byzantine join attack*. Under this scenario, $(1/4-\varepsilon)N$ malicious nodes join the network, which has at least N nodes in total, with no more than N^p honest nodes joining and leaving, for some parameter $p < 1$.

In S-Chord, nodes receive a random ID, which is a real number between 0 and 1, when they join the network. A central concept in S-Chord is the notion of a *swarm*. For every point x on the unit circle, its swarm $S(x)$ is the set of nodes with ID's located on the unit circle within a clockwise distance of $(c\ln N)/N$ of x, for some design parameter c. A swarm is considered *good* if it has a fraction of at least 3/4 of honest nodes. If the assumptions mentioned above hold, with c large enough, it can be said that all swarms are good with high probability.

All nodes maintain links to all peers in the following intervals:

—$Center(x)$ is the set of peers in the interval $[x-(2c\ln N)/N, x+(2c\ln N)/N]$.

—$Forward(x,i)$ is the set of peers in the interval $[x-(2c\ln N)/N+2^i/M, x+(2c\ln N)/N+2^i/M]$, for $i = 1,2,\ldots,\log M-1$.

—$Backward(x,i)$ is the set of peers in the interval $[x-(2c\ln N)/N-2^i/M, p+(2c\ln N)/N-2^i/M]$, for $i = 1,2,\ldots,\log M-1$.

where M is a number greater than the number of nodes (in practice, node identifiers are m-bit strings and $M = 2^m$ is the maximum possible number of nodes).

The forward intervals are the equivalent to fingers in standard Chord. The difference is that in Chord, the lookup procedure forwards a message to a single peer responsible for key K, while in S-Chord, the message is forwarded to a whole swarm. The purpose of the backward intervals (nonexistent in Chord) is that in S-Chord, a node does not trust a peer

to tell its identifier. Thus, the only peers a node can receive messages from are those listed in the backward intervals.

When a node x joins the system, it is assumed to contact an honest node y, which notifies its own swarm $S(y)$. The nodes in $S(y)$ come to consensus on a random ID for x, and inform nodes in $Center(x)$ that x has joined the system and send to x the identifiers of and pointers to nodes in $Center(x)$. The nodes in $Center(x)$ send to x the data items for all keys K such that $x \in S(K)$. Finally, all peers in $S(x)$ notify the peers in their respective *Forward* and *Backward* intervals about x's existence, and provide x with pointers to those intervals. The join procedure requires $O(\log^3 N)$ messages.

The lookup procedure in S-Chord is analogous to Chord's version, but exploiting swarms. When a peer x requests a key K, x initially sends the request to all peers in $S(x)$. The peers in this swarm forward the message to all peers in $S(z)$, where z is the point in the forward intervals closest clockwise to K. The message is forwarded repeatedly in the same manner until all peers in $S(z)$ have pointers to all peers in $S(K)$, which are sent backwards along the same path. This lookup procedure has latency $O(\log N)$ and requires $O(\log^3 N)$ messages.

Routing attacks are prevented by making nodes in a swarm S_j forward a lookup request to the nodes in the next swarm S_{j+1} only if the request was received from a majority of the nodes in the previous swarm S_{j-1}.

To reduce the number of messages in a lookup to $O(\log^2 N)$, the procedure for sending a message from a swarm S_{j-1} to swarm S_j is changed so that each node $x \in S_{j-1}$ will send the message to a peer $y \in S_j$ only if $h_1(x) = h_1(y) \bmod \log N$, and each node $y \in S_j$ will accept a message from $x \in S_{j-1}$ only if the same condition holds. The hash function h_1 maps IDs to positive integers. Once the peer $y \in S_j$ has received messages from at least two thirds of the possible peers, y does a majority filtering on the received messages to decide which message it will propagate.

Another characteristic of S-Chord is that it requires only a constant increase of the bandwidth consumption in a lookup operation with respect to Chord. This is achieved by encoding the message in multiple pieces using an erasure code, sending each piece only to nodes whose ID hash to the same hash value of the piece, and having the nodes in the destination swarm execute a Byzantine agreement protocol to reach a consensus. If consensus is not reached, a resend request is sent to the previous swarm. A separate hash function h_2 is used to map pieces to positive integers.

This theoretical scheme gives Chord tolerance to routing and storage attacks by using wide paths and majority voting. One aspect that is not clear in the join algorithm is how the new node ensures that the node used to join the overlay is correct. Moreover, if there is a set of known correct nodes in the system, perhaps the use of a consensus algorithm to generate random identifiers is unnecessary.

Since the system is designed to be optimized asymptotically, one may expect some of the proposed techniques to have a negative effect on the performance of a practical implementation. For example, the use of erasure codes and Byzantine agreement protocols to make the bandwidth increase a constant with respect to Chord may have a negative overall effect, as in practice it has been found that using erasure codes results in similar bandwidth consumption than replication [Blake and Rodrigues 2003], and the processing burden may significantly increase the latency.

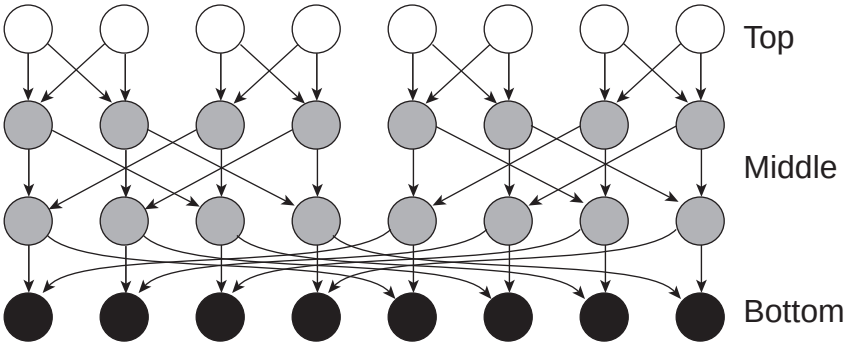


Fig. 2. A butterfly network of supernodes. Each supernode has multiple nodes, and the connections between supernodes are expander graphs between their constituent nodes.

5.9 Fiat and Saia

Fiat and Saia [2007] describe an approach to building a DHT capable of storing N data items in N nodes. Their system is based on a butterfly network (see Figure 2), and is able to resist the random deletion of a fraction of less than $N/2$ nodes. The vertices of the butterfly network are called supernodes. Every supernode is associated with a set of real nodes. Supernodes at the topmost level of the network are referred to as top supernodes, those at the bottommost level are bottom supernodes, and the rest are middle supernodes. The butterfly network has a depth of $\log N - \log\log N$.

Nodes are mapped uniformly and independently at random to c top supernodes, c bottom supernodes, and $c\log N$ middle supernodes. The two sets of nodes associated with two connected supernodes in the butterfly network, are connected by a random expander graph of constant degree d. In addition, every node chooses c_{top} random top supernodes and points to all nodes in those supernodes. Each key is hashed to c_{bottom} random bottom supernodes, and is stored in all the nodes of the bottom supernodes to which it has been hashed. The constants c, d, c_{top} and c_{bottom} are determined by an error parameter $\varepsilon > 0$.

To perform a lookup starting from node x, multiple requests start in parallel in all top supernodes pointed to by x. Each of these requests moves through links between the supernodes until it ends in a bottom supernode to which the key is mapped or until all paths between the starting top supernode and all bottom supernodes to which the key is mapped are completely searched. In case of a successful search the results are returned backwards along the same path that the query came in.

The authors show that this system has the following properties, given the error parameter ε:

—Each node requires $c_1(\varepsilon)\log N$ memory.

—Lookups take $c_2(\varepsilon)\log N$ time and $c_3(\varepsilon)\log^2 N$ messages.

—At any time, all but εN nodes can find all but εN keys.

A proposed modification that can make the system resistant to nodes reporting false versions of the data items is to make sure that there is a full bipartite graph between any pair of connected supernodes instead of a constant degree expander graph. The search procedure is modified so that a node will send a data item or request to the following

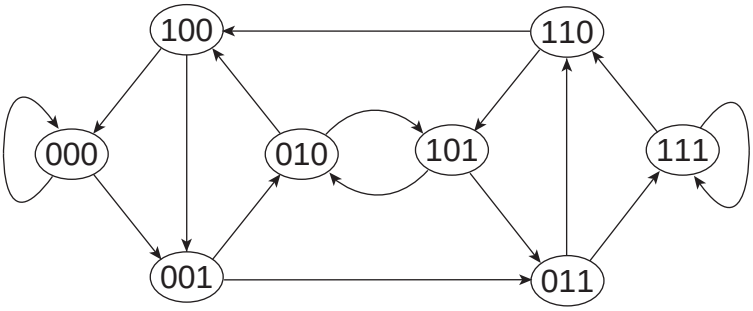


Fig. 3. Classical de Bruijn graph over $\{0,1\}^3$. It can be seen that every node $b_0b_1b_2$ is connected to nodes b_1b_20 and b_1b_21.

supernode only if the majority of the items received from the previous supernodes have the same value. This modification changes the properties of the system such that:

—Each node requires $c_1(\varepsilon)\log^2 N$ memory.

—Lookups take no more than $c_2(\varepsilon)\log N$ time and $c_3(\varepsilon)\log^3 N$ messages.

—All but εN nodes can find all but εN of the true data items.

This system is a revision of previous work on butterfly networks [Fiat and Saia 2002; Saia et al. 2002].

We note that this scheme is an interesting theoretical work backed with proofs of desirable properties. However, some aspects are not clearly defined such as node join and leave protocols, or a generalization for storing an arbitrary number of data items.

5.10 Awerbuch and Scheideler

Awerbuch and Scheideler [2006] propose a DHT based on a dynamic variant of a de Bruijn graph. In a classical de Bruijn graph, $\{0,1\}^m$ represents nodes. Each node x, identified by the string $(x_1x_2\cdots x_m)$ is connected either to nodes $(0x_1x_2\cdots x_{m-1})$ and $(1x_1x_2\cdots x_{m-1})$, or nodes $(x_2x_3\cdots x_m0)$ and $(x_2x_3\cdots x_m1)$. That is, nodes that result either in a right or left shift of the bits of x. Figure 3 shows a classical de Bruijn graph over $\{0,1\}^3$. If the bit string identifiers are considered as binary representations of fractional part of the real numbers in the interval $[0,1)$, then this is equivalent to saying that a node x is connected to nodes $x/2$ and $(1+x)/2$, or $2x \bmod 1$ and $(2x-1) \bmod 1$.

This system uses the real interval $[0,1)$ as its identifier space. A region is an interval of size $1/2^m$, where m is an integer. For any integer m, the identifier space is divided in 2^m regions, as required by a de Bruijn graph. A *k-region* is a region of size closest to k/N, with N being the maximum number of honest nodes. For any point $x \in [0,1)$, $R_k(x)$ is the unique k-region containing x. When a node joins, it is mapped to a random point y in $[0,1)$ using a verifiable secret sharing scheme. Every time a node joins the system with identifier x, all the nodes in $R_k(x)$ are moved to random points in $[0,1)$. This is known as the *cuckoo rule* (see Section 4.5) and guarantees that the number of nodes in $k-$regions is balanced and that each $k-$region contains a majority of honest nodes, as long as the ratio ε of malicious nodes to honest nodes satisfies $\varepsilon < 1-1/k$. This holds even under the presence of a join-leave attack, where malicious nodes continuously join and leave the system until they get the identifier they want.

In the dynamic variant of the de Bruijn graph used by the system, the quorum region R_x of a node identified by $x \in [0,1)$ is defined as a unique region of size closest to $(\gamma \log N)/N$ that contains x, where $\gamma > 1$ is a constant. For any set of nodes $V \subset [0,1)$, every node $x \in V$ maintains connections to all nodes whose quorum regions contain a point in $\{x, x/2, (1+x)/2, 2x \bmod 1, (2x-1) \bmod 1\}$. When a node x joins or leaves, only quorum regions containing a point in $\{x, x/2, (1+x)/2, 2x \bmod 1, (2x-1) \bmod 1\}$ are affected, which according to the cuckoo rule, amount to $O(\log N)$ regions, with high probability.

To send a message from $x \in [0,1)$ with binary representation $(x_1 x_2 \cdots x_{\log N})$ to $y \in [0,1)$ with binary representation $(y_1 y_2 \cdots y_{\log N})$, the message must be forwarded along the quorum regions containing the points $(x_2 x_3 \cdots x_{\log N} y_1)$, $(x_3 x_4 \cdots x_{\log N} y_1 y_2)$, and so on, until the quorum region containing the point $(y_1 y_2 \cdots y_{\log N})$.

The basic strategy to prevent storage attacks is to use $2c-1 = \Theta(\log N)$ hash functions to map data items to $2c-1$ points in $[0,1)$. Each replica i of a data item x must be stored in all nodes of the quorum region containing the point $h_i(x)$, where h_i is one of the $2c-1$ hash functions.

The lookup protocol proceeds in rounds and allows several attempts for each lookup request and uses a mechanism to prevent congestion possibly caused by malicious nodes generating bogus messages. In each round, a packet is sent to each of the $2c-1$ destinations. The packets are routed level by level to each route according to the simple routing protocol. If a region has more than $\gamma c \log^2 N$ packets for some level l, all of them are discarded. Otherwise they are routed to the next level. The answers are sent backwards along the same route.

The insertion protocol is similar, but for the destination level the bound is $\gamma' c \log N$, which is more restrictive.

The authors show that for any set of n lookup or insert requests, the lookup protocol can serve all requests in $O(\log n)$ attempts and $O(c \log^4 n)$ communication rounds. For any set of n insert requests, every node has to store at most $O(c \log^2 n)$ copies.

This scheme is an interesting theoretical work backed with proofs of desirable properties, including a congestion control mechanism that may help defend against denial-of-service attacks, and a mechanism to guarantee that malicious nodes are spread over the identifier space. Some of the protocols seem complex, although this may be due to the fact that this work provides detailed protocols for more operations than other theoretical solutions. Still, some lower-level protocols need to be defined to cover all aspects of a DHT operation, such as routing table maintenance.

5.11 Naor and Wieder

Naor and Wieder [2003] propose a DHT that tolerates an adversary able to remove random peers in the system or to make them produce arbitrary false versions of the data items requested.

This system is designed using a so-called *continuous-discrete* approach. The topology of the system is initially defined as a continuous graph G_C which is later discretized.

The vertex set of G_C is the set of real numbers $[0,1)$, and the edge set is defined such that for each vertex x there exist the edges $l(a) = (x, x/2)$ and $r(a) = (x, (x+1)/2)$. It should be noted that this is essentially a de Bruijn graph, similar to the one described in Section 5.10.

The authors show that it is possible to get within a distance of 2^{-p} of any node y from any node x if one has a prefix of length p of the binary representation σ_p of y. The procedure

consists of doing a walk that starts in x and proceeds by processing σ_p from right to left with 0 meaning that the edge $l(x)$ should be traversed, and 1 traversal of $r(x)$. For example, if $x = (.100)_2$ and $y = (.110)_2$, the walk would successively pass points $(.0110)_2$, $(.00110)_2$ and would end at $(.100110)_2$ which is indeed within a distance of 2^{-3} of x.

The discretization of the system proceeds as follows. When a node joins the system, it chooses a random identifier $x_i \in [0,1)$ and estimates a bound $q_i \approx \log N/N$. The node is said to cover the segment $[x_i, x_i + q_i]$, and will have links to another node if the segments covered by both nodes overlap or if they contain vertices connected by edges in G_C. The authors show that each data item is covered by $\Theta(\log N)$ nodes.

The authors provide two lookup methods. The first one consists of an emulation of a walk in the continuous graph towards a node that covers the requested item. It can tolerate random fail-stop failures, but not Byzantine behavior. The second method is tolerant to faulty nodes that produce false data items. It is also based on continuous walk emulation, but the message is passed to all $\Theta(\log N)$ nodes covering the requested item. At each step in the walk, each node receives $\Theta(\log N)$ messages, one from each node covering the previous step. Each intermediate node continues sending the message only if it was received from a majority of nodes in the previous step.

This scheme is another interesting theoretical work. It is defined at a high level of abstraction and makes many assumptions about critical aspects of DHT operation such as join and leave protocols, node identifier assignment, and routing table maintenance.

5.12 Discussion

Defenses against storage and routing attacks are based on two mechanisms: redundant storage and redundant routing.

The most common way to achieve redundant storage is to replicate data, which allows for relatively simple implementation of maintenance and verification algorithms. An alternative method is erasure coding, as proposed by Fiat et al. [2005], Dabek et al. [2004] and Mills and Znati [2008]. In theory, coding has the advantage of requiring less storage and bandwidth at the cost of extra system complexity and possibly higher latencies. However, a study [Rodrigues and Liskov 2005] has shown that while coding does provide storage savings, the bandwidth required to maintain appropriate redundancy levels under various degrees of churn is approximately the same for both coding and replication. This study does not consider mutable data nor takes into account malicious nodes, which would most likely increase the complexity of coding. It is clear that secure DHT designers have considered this and have preferred to implement data redundancy using replication.

There are also several approaches to data replication. The first consists of storing replicas in nodes that are numerically close in the identifier space. This is the method used by Pastry, Chord, Kademlia, and many of the reviewed approaches in this section [Castro et al. 2002; Hildrum and Kubiatowicz 2003; Fiat et al. 2005; Naor and Wieder 2003; Wang et al. 2007]. The second method is to store replicas at locations spread over the identifier space. This is the method used by Tapestry, which stores replicas at random locations, and Harvesf and Blough [2006] who propose storing replicas at equally spaced locations. Others combine both approaches and select locations spread over the identifier space, and store multiple replicas in nodes that are in the vicinity of each location [Fiat and Saia 2007; Awerbuch and Scheideler 2006].

One of the main advantages of having replicas at numerically close locations is that it may be easier to maintain the desired replication degree and to keep replicas consistent

in case of mutable data, since nodes keep links to numerically close peers in most DHT topologies. However, this approach requires malicious nodes to be spread over the identifier space. Otherwise a small number of malicious nodes could concentrate on a specific region of the overlay and easily take control of all replicas in that region. Replicas spread over the identifier space are not necessarily a big improvement in this respect, since DHT placement algorithms are public knowledge, and malicious nodes may attack all relevant locations for a specific key.

This shows that data redundancy is not enough to prevent storage attacks. It is also necessary to guarantee that nodes are unable to select their own location in the identifier space. In previous sections we have seen that the most straightforward way to achieve this is by using random identifiers issued by a trusted certification authority able to limit the fraction of malicious nodes [Castro et al. 2002]. Another possibility is to have a set of participating nodes generate identifiers using a Byzantine-fault-tolerant consensus algorithm [Awerbuch and Scheideler 2006] plus induced churn to prevent concentration of malicious nodes at specific regions by means of a join-leave attack [Awerbuch and Scheideler 2006; Condie et al. 2006]. However, this does not limit the number of attackers, and should be coupled with other techniques to limit Sybil attacks.

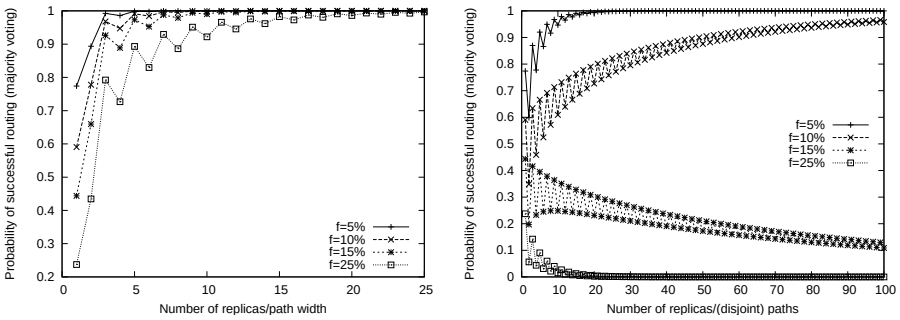
Another dimension of secure storage concerns data verification, which refers to how a querier ensures that the results from a lookup are correct. The reviewed papers do not concentrate most of their efforts on this issue as it is application dependent. The most important aspects to consider are the use of self-certifying data, and whether data is mutable or not.

Read-only self certifying data is the most favorable case as it requires access to only one valid replica, which increases the probability of successful routing even without redundant routing. Redundant routing, however is needed both for inserting data and to determine if a given key exists or not. If data is mutable, it may be possible to only retrieve metadata from multiple replicas, full data from a single replica, and verify the data according to an application defined rule (e.g., highest version number stored in the metadata or majority voting). Castro et al. [2002] outlines the use of a Byzantine-fault-tolerant replication algorithm [Castro and Liskov 2002] to provide strong consistency guarantees for mutable data, but many other replication algorithms could be used as long as they are aware of the overlay structure and the possibility of malicious behaviour, and provide the consistency guarantees required by the application, which may not necessarily be strong.

If self-certifying data cannot be used, the most straightforward method to verify read-only data is to retrieve multiple replicas and use majority voting. Storing mutable data without any verification capability at the application level leaves the system open to data poisoning attacks that may be executed by external entities that are not necessarily part of the DHT.

Another storage issue that may arise is concurrent conflicting modification of mutable data. Solving this problem may require adding transactional capabilities to the DHT. Such systems have been proposed [Antony et al. 2008; Schütt et al. 2008; Plantikow et al. 2007; Mesaros et al. 2005], but we are not aware of any transactional DHT design that tolerates malicious nodes.

Finally, in order to reliably locate nodes responsible for a given key, it is necessary to have some form of redundant routing. Redundant routing can be implemented using multiple paths [Castro et al. 2002; Sánchez Artigas et al. 2005; Harvesf and Blough 2006],



(a) Wide Paths (b) Multiple Paths

Fig. 4. Probability of success for redundant routing approaches using majority voting and assuming that malicious nodes are randomly spread over the ID space. In both cases, the number of replicas equals the width or the number of paths, and the number of hops is set to 5. The multiple paths are assumed to be disjoint.

wide paths [Hildrum and Kubiatowicz 2003; Fiat et al. 2005; Naor and Wieder 2003; Wang et al. 2007; Maymounkov and Mazières 2002], or multiple wide paths [Fiat and Saia 2007; Awerbuch and Scheideler 2006]. Wide paths are suitable to replicas stored at numerically close nodes, while multiple paths are a better match for replicas spread over the identifier space.

In a majority voting scenario, wide paths are much more reliable than multiple paths, since they require only one good node at each intermediate routing step, while multiple paths require a majority of paths to consist exclusively of honest nodes. If malicious nodes are spread over the identifier space, the probability of successful routing using wide paths and majority voting is

$$(1-f^p)^{h-1} \sum_{i=\left\lceil \frac{r+1}{2} \right\rceil}^{r} \binom{r}{i} (1-f)^i f^{r-i}$$

while the probability of successful routing using multiple paths is no more than

$$\sum_{i=\left\lceil \frac{p+1}{2} \right\rceil}^{p} \binom{p}{i} (1-f)^{hi} \left[1-(1-f)^h\right]^{p-i}$$

where f is the fraction of malicious nodes, r is the number of replicas, p is the number of paths or path width ($p = r$ for multiple paths), and h is the number of hops. Figure 4 shows that the multiple path approach requires a low fraction of malicious nodes and a larger number of replicas to achieve a high probability of success. Probably for this reason, Castro et al. [2002] use multiple paths, but introduce a protocol that exploits the numerical closeness of replicas to construct a definitive replica set, requiring approximately only one correct path. This increases the probability of successful routing, but has the disadvantage that it introduces additional complexity, and may reduce the probability that the multiple paths are disjoint.

Multiple wide paths combine both approaches, by trying wide paths successively. A disadvantage of multiple wide paths is that maintaining consistency when replicas are mutable may be more expensive in most DHT topologies.

The approach by Wang et al. [2007] is similar to wide paths, but instead of trying multiple nodes simultaneously at each step, it tries only one node, and resorts to an alternative only in case of failures.

Making a fair comparison of solutions for routing and storage attacks is difficult because not all approaches try to solve the same problems or make the same assumptions regarding the attack model. Some proposals pay little or no attention to storage attacks and are mostly concerned with securely routing a message to a single node responsible for a key [Ganesh and Zhao 2005; Sánchez Artigas et al. 2005; Wang et al. 2007], while others address both problems with either a unified approach [Hildrum and Kubiatowicz 2003; Harvesf and Blough 2006; Fiat and Saia 2007; Fiat et al. 2005; Awerbuch and Scheideler 2006; Naor and Wieder 2003] or with separate protocols that complement redundant routing [Castro et al. 2002].

Perhaps the clearest distinction between the different solutions is that some are based on purely theoretical constructions defined at a very high level [Fiat and Saia 2007; Fiat et al. 2005; Awerbuch and Scheideler 2006; Naor and Wieder 2003], while others take a more engineering approach and have their properties measured with experiments, but also with mathematical analysis in many cases [Castro et al. 2002; Hildrum and Kubiatowicz 2003; Harvesf and Blough 2006; Ganesh and Zhao 2005; Sánchez Artigas et al. 2005; Wang et al. 2007]. Engineered solutions have the advantage that they may be reasonably easy to implement in practice, while the main objective of theoretical solutions is to study the feasibility of countering attacks while maintaining provable scalability properties, though this may result in schemes that are difficult or impossible to implement in a secure way, or that exhibit poor performance in practice. Nevertheless, both types of approach are subject to assumptions that may be difficult to realize in real-world applications. Table III summarizes and provides a comparison of experimentally tested defense against routing and storage attacks, while Table IV does the same for theoretical defenses.

Table III: Comparison of experimentally tested defenses against routing and storage attacks.

Technique	**Authors**	**Advantages**	**Disadvantages**
Use of two routing tables: one optimized with network measurements, and one constrained used in case of a test failure. The constrained table is used with redundant routing over multiple paths. Replicas are placed at numerically close locations, and there is a protocol to determine the replica set if at least one of the multiple paths is correct.	Castro et al. [2002]	In principle, it allows the use of proximity routing. Apart from the offline CA, it does not assume the existence of services that may be difficult to implement and administrate.	The frequency of redundant routing is increased due to attacks against the routing failure test and the progressive poisoning of the optimized routing tables. The number of required paths may be large because they are not guaranteed to be disjoint and a single malicious node completely invalidates all the paths in which it is included.

Continued on next page

Table III – Continued

Technique	Authors	Advantages	Disadvantages
Use of redundant routing table entries based on network proximity. Routing uses wide paths, which take advantage of the redundant table entries.	Hildrum and Kubia-towicz [2003]	Uses proximity routing and wide paths.	Depends on secure and stable network distance measurements, which may not be readily available in practice. The grid topology used in the experiments may produce distance measurements that are not realistic.
Multiple independent paths constructed by partitioning nodes using an equivalence relation.	Sánchez et al. [2005]	Simple design that leverages existing protocols. Independent paths are expected to be more reliable than non-independent paths.	Addresses only routing attacks, but the topology is suitable for the implementation of traditional DHT replication methods.
Certificates that prove the existence of nodes responsible for ID prefixes. These proofs are placed at random locations in a Tapestry-style DHT.	Ganesh and Zhao [2005]	Very effective at detecting attacks.	Does not prevent or mitigate storage attacks. Online distribution of key pairs may lead to further security issues. Can be seen as a complicated way to implement multiple routes.
Replicas placed at equally-spaced locations in a Chord ring. Routing uses multiple paths.	Harvesf and Blough [2006]	The number of disjoint paths is increased with respect to other approaches based on multiple paths.	Tested only with self verifying data, which is not a realistic model.
Modification to Chord that introduces a verification procedure based on certificates issued by an online neighborhood authority.	Wang et al. [2007]	Routing looks more efficient than routing using wide or multiple paths, and as reliable as using wide paths. Tested on a real implementation and with simulations.	The neighborhood authority introduces an online centralized component and administrative costs. Experiments did not test the verification procedure or the effect of malicious nodes on latency. Does not address data replication, but the topology is suitable for the implementation of traditional DHT replication methods.

6. IMPLEMENTATIONS

DHTs have been used in numerous popular peer-to-peer systems in the real world, such as the KAD network (used by eMule and other compatible programs), BitTorrent and LimeWire.

Table IV: Comparison of theoretical approaches to defend against routing and storage attacks.

Technique	Authors	Advantages	Disadvantages
Topology based on a variant of a de Bruijn graph. Routing uses wide paths.	Naor and Wieder [2003]	The topology is simple and makes it relatively easy to support mutable data and defend against storage attacks.	Makes more assumptions about lower level protocols such as node joins/leaves, routing table maintenance, and node ID assignment than other theoretical proposals.
Modification to Chord that uses swarms of nodes instead of single nodes as the basic construct. Each step in routing is a swarm, resulting in wide paths.	Fiat et al. [2005]	Based on a well known topology that makes it straightforward to protect against storage attacks and support mutable data.	The assumption that there are well-known honest nodes might make the distributed ID generation scheme unnecessary. Use of erasure codes makes the protocol more complex, although it provides good asymptotical properties for bandwidth.
Topology based on a dynamic de Bruijn graph. Routing uses multiple wide paths.	Awerbuch and Scheideler [2006]	Provides a decentralized node ID assignment protocol that does not depend on trusted nodes. The lookup procedure introduces a congestion control mechanism. Provides algorithms to defend against storage attacks.	Supporting mutable data would require non-trivial protocols to efficiently keep data in multiple regions consistent. Algorithms are complex, though this may be due to this work making fewer high-level assumptions than other theoretical proposals.
Topology based on a butterfly network of supernodes. Each supernode contains multiple real nodes, and connections between supernodes are expander graphs between their constituent sets of nodes, which makes paths among supernodes similar to wide paths. Replicas are placed at several supernodes and are looked up sequentially until a correct replica supernode is found.	Fiat and Saia [2007]	Provides algorithms to defend against storage attacks.	The model assumes that only N data items are going to be stored. It is not explained how the model can be extended to support an arbitrary number of items. The topology may make it difficult to support mutable data. Makes more assumptions about lower level protocols such as node ID and supernode assignment, node joins/leaves and maintenance, than other theoretical proposals.

It is interesting to see that all of these use the Kademlia protocol. We believe that the main reason for this choice is Kademlia's combined properties of performance and relative security. For one, it is difficult to affect the routing tables of a Kademlia node, as each node tends to keep only highly available peers in its routing table. This increases the required costs for an attacker to convince honest nodes to link to the compromised nodes. Similarly, Kademlia uses iterative routing, exploring multiple nodes at each step, making routing less dependent on specific nodes and thus less vulnerable to attacks.

However, Kademlia is still vulnerable to Sybil and Eclipse attacks as nodes can generate their own identifiers. In addition, because the protocol is based on UDP, spoofing other nodes becomes relatively easy. Steiner et al. [2007] show that it is indeed very easy to launch Sybil and Eclipse attacks in KAD, which is probably the largest DHT currently deployed, with estimates between 1.5 million [Steiner et al. 2007] and 4 million [Crosby and Wallach 2007] nodes. Moreover, they also show how honest nodes can be enlisted against their will to participate in DDoS attacks against other systems.

Crosby and Wallach [2007] study two Kademlia implementations used by BitTorrent clients. They concentrate on performance issues, but they also point out the security problems related to node identifier assignment. They argue that for this application, having a central authority would not be politically feasible, and that NATs make it difficult to use hashed IP addresses.

Kademlia is also used by the Storm botnet, which is used by its operators for criminal activities such as DDoS attacks and sending of spam. Holz et al. [2008] study the botnet and show that early versions of the bot code use the Kademlia-based Overnet network[1], which also contains benign peers, while newer versions connect to a separate DHT formed exclusively by bots. The Storm operators use keys that depend on the current date to store files whose contents encapsulate the IP addresses of nodes that provide commands to the bots, and use data replication spread over the identifier space. The authors of the study attack the botnet DHT with Sybil and Eclipse attacks similar to those performed against KAD [Steiner et al. 2007] and found out that the Eclipse attack is less effective than in KAD since they need to corrupt more routes as data is replicated over the identifier space. They opted to mitigate the effects of the botnet by performing a successful DoS attack where they rewrite existing keys by republishing them using bogus data. We note however, that this mitigation technique could be easily defeated by Storm operators with the use of digital signatures.

Another well documented DHT in operation is OpenDHT [Rhea 2005], which runs on approximately 200 PlanetLab hosts, uses the Bamboo protocol [Rhea et al. 2004], and is used for many academic applications that require a remote storage service. OpenDHT is not designed to tolerate malicious nodes, but it assigns node identifiers using a hash of the node IP address and port, uses redundant storage, and resorts to alternative routing paths in case of failures. It supports mutable data and uses gossiping algorithms based on anti-entropy [Demers et al. 1987] to maintain consistency.

It can be seen that real-world deployments, even though they are not optimized for security, employ protocols that use some form of redundant storage and routing, which improves reliability and security. The nature of the applications that use the DHT dictates

[1]Overnet is a file sharing network used by the eDonkey program. In principle, it was shutdown in 2006 after legal action by the recording industry [Los Angeles Times 2006], but due to its decentralized design, it is still in operation.

the identifier assignment method. File-sharing applications KAD, LimeWire and BitTorrent, as well as the Storm botnet, choose maximum flexibility over security and let nodes choose their own ID, probably hoping that their sheer size will be enough to keep the fraction of malicious nodes low; while OpenDHT, which runs on the more closed environment of PlanetLab and does not have to deal with NATs, uses the stronger though imperfect method of hashed IP addresses. Moreover, the Storm botnet, which publishes read-only data, makes use of replicas spread over the ID space, while OpenDHT uses numerically close replicas to easily support consistency protocols for mutable data.

This shows that there is no single best approach to implement a secure DHT, but that the assumptions that can be made by the application using the DHT are what dictates which techniques to employ. It is also clear that achieving security in a DHT application is difficult since it requires using complex techniques and solving difficult trade-offs, especially in the node ID assignment procedures.

7. CONCLUSIONS

We have discussed some well-known security threats faced by distributed hash tables and have reviewed several techniques proposed to solve or mitigate them. The variety of the proposed solutions, and the trade-offs they introduce, show how difficult it is to secure a DHT system in a hostile environment.

We can conclude that securing a DHT requires secure assignment of node identifiers, a low fraction of malicious nodes, malicious nodes spread over the identifier space, data replication, and a routing mechanism that provides a high probability of reaching a correct replica set. We have reviewed techniques that aim to solve these problems, usually under different assumptions. We consider that the best solution for a given application depends on what assumptions can be made; it may require the combination of several of the reviewed approaches.

Current DHT deployments are not specifically designed to tolerate the presence malicious nodes. However, most of them are based on Kademlia, which provides relative security by using data replication and a redundant routing mechanism similar to wide paths. However, it is still vulnerable to Sybil attacks, as nodes can generate their own identifiers.

The most challenging problem for securing DHTs and decentralized systems in general is robust and secure assignment of node identifiers. This is crucial to guarantee that malicious nodes represent a small fraction and that they cannot choose their location in the overlay, thus preventing Sybil and Eclipse attacks. This conclusion is also drawn in other studies [Baumgart and Mies 2007; Cerri et al. 2005] in which secure identifier assignment is proposed.

In any case, although it may be clear that we may be able to reach a level of security that is practically acceptable for various DHT-based applications, our survey shows that much more work is needed if security requirements are demanding.

REFERENCES

ANDERSON, D. P. AND FEDAK, G. 2006. The Computational and Storage Potential of Volunteer Computing. In *Proc. 6th International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Press, Los Alamitos, CA., 73–80.

ANTONY, S., AGRAWAL, D., AND ABBADI, A. E. 2008. P2P Systems with Transactional Semantics. In *Proc. 11th International Conference on Extended Database Technology* (Nantes, France). ACM Press, New York, NY, 4–15.

AWERBUCH, B. AND SCHEIDELER, C. 2004. Group Spreading: A Protocol for Provably Secure Distributed Name Service. In *Proc. 31st International Coll. on Automata, Languages and Programming (ICALP)* (Turku, Finland). Lecture Notes on Computer Science, vol. 3142. Springer-Verlag, Berlin, 183–195.

AWERBUCH, B. AND SCHEIDELER, C. 2006. Towards a Scalable and Robust DHT. In *Proc. 18th Symposium on Parallel Algorithms and Applications* (Cambridge, Massachusetts, USA). ACM Press, New York, NY, 318–327.

AWERBUCH, B. AND SCHEIDELER, C. 2007. Towards Scalable and Robust Overlay Networks. In *Proc. 6th International Workshop on Peer-to-Peer Systems*. Bellevue, WA.

BAUMGART, I. AND MIES, S. 2007. S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing. In *Proc. 13th International Conference on Parallel and Distributed Systems* (Hsinchu, Taiwan). IEEE Computer Society Press, Los Alamitos, CA., 2:1–8.

BAZZI, R. A. AND KONJEVOD, G. 2005. On the Establishment of Distinct Identities in Overlay Networks. In *Proc. 24th Symposium on Principles of Distributed Computing* (Las Vegas, NV). ACM Press, New York, NY, 312–320.

BAZZI, R. A., RI CHOI, Y., AND GOUDA, M. G. 2006. Hop Chains: Secure Routing and the Establishment of Distinct Identities. In *Proc. 10th International Conference on Principles of Distributed Systems* (Bordeaux, France). Lecture Notes on Computer Science, vol. 4305. Springer-Verlag, Berlin, 365–379.

BLAKE, C. AND RODRIGUES, R. 2003. High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. In *Proc. 9th Hot Topics in Operating Systems*. USENIX, Berkeley, CA, 1–6.

BORISOV, N. 2006. Computational Puzzles as Sybil Defenses. In *Proc. 6th International Conference on Peer-to-Peer Computing*. IEEE Computer Society Press, Los Alamitos, CA., 171–176.

CAI, M., CHERVENAK, A., AND FRANK, M. 2004. A Peer-to-Peer Replica Location Service Based on a Distributed Hash Table. In *Proc. 18th International Conference on Supercomputing*. IEEE Computer Society Press, Los Alamitos, CA., 56.

CASTRO, M., COSTA, M., AND ROWSTRON, A. 2004. Performance and Dependability of Structured Peer-to-Peer Overlays. In *Proc. International Conference on Dependable Systems and Networks*. IEEE Computer Society Press, Los Alamitos, CA., 9–18.

CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., AND WALLACH, D. S. 2002. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Proc. 5th Symposium on Operating System Design and Implementation* (Boston, MA). ACM Press, New York, NY, 299–314.

CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., AND ROWSTRON, A. 2002. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communication 20,* 8 (Oct.), 100–110.

CASTRO, M. AND LISKOV, B. 2002. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems 20,* 4 (Nov.), 398–461.

CERRI, D., GHIONI, A., PARABOSCHI, S., AND TIRABOSCHI, S. 2005. ID Mapping Attacks in P2P Networks. In *Proc. Global Telecommunications Conference (GLOBECOM)* (St. Louis, MO). IEEE Computer Society Press, Los Alamitos, CA.

CONDIE, T., KACHOLIA, V., SANKARARAMAN, S., HELLERSTEIN, J., AND MANIATIS, P. 2006. Induced Churn as Shelter from Routing Table Poisoning. In *Proc. 13th Symposium on Network and Distributed System Security* (San Diego, CA). The Internet Society, San Diego, CA.

CROSBY, S. AND WALLACH, D. S. 2007. An Analysis of BitTorrent's Two Kademlia-Based DHTs. Tech. Rep. TR-07-04, Department of Computer Science, Rice University, Houston, TX, USA.

DABEK, F., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. 2001. Wide-area Cooperative Storage with CFS. In *Proc. 18th Symposium on Operating System Principles* (Baniff, Canada). ACM Press, New York, NY.

DABEK, F., LI, J., SIT, E., ROBERTSON, J., KAASHOEK, M. F., AND MORRIS, R. 2004. Designing a DHT for Low Latency and High Throughput. In *Proc. 1st Symposium on Networked Systems Design and Implementation* (San Francisco, CA). USENIX, Berkeley, CA, 85–98.

DAHAN, S. AND SATO, M. 2007. Survey of Six Myths and Oversights about Distributed Hash Tables' Security. In *Proc. 27th International Conference on Distributed Computing Systems Workshops*. IEEE Computer Society Press, Los Alamitos, CA., 26.

DANEZIS, G., LESNIEWSKI-LAAS, C., KAASHOEK, M. F., AND ANDERSON, R. J. 2005. Sybil-Resistant DHT Routing. In *Proc. 10th European Symposium on Research in Computer Security*. Lecture Notes on Computer Science. Springer-Verlag, Berlin, 305–318.

DASWANI, N. 2004. Denial-OF-Service (DOS) Attacks and Commerce Infrastructure in Peer-to-Peer Networks. Ph.D. thesis, Dept. of Computer Science, Stanford University.

DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. 1987. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. 6th Symposium on Principles of Distributed Computing* (Vancouver). ACM Press, New York, NY, 1–12.

DINGER, J. AND HARTENSTEIN, H. 2006. Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration. In *Proc. 1st International Conference on Availability, Reliability and Security* (Vienna, Austria). IEEE Computer Society Press, Los Alamitos, CA., 756–763.

DOUCEUR, J. R. 2002. The Sybil Attack. In *Proc. 1st International Workshop on Peer-to-Peer Systems*. Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 251–260.

FIAT, A. AND SAIA, J. 2002. Censorship Resistant Peer-to-Peer Content Addressable Networks. In *Proc. 13th Symposium on Discrete Algorithms* (San Francisco, CA). Society for Industrial and Applied Mathematics, Philadelphia, PA, 94–103.

FIAT, A. AND SAIA, J. 2007. Censorship Resistant Peer-to-Peer Networks. *Theory of Computing 3,* 1, 1–23.

FIAT, A., SAIA, J., AND YOUNG, M. 2005. Making Chord Robust to Byzantine Attack. In *Proc. 13th Annual European Symposium on Algorithms*. Lecture Notes on Computer Science, vol. 3669. Springer-Verlag, Berlin, 803–814.

FONSECA, R., RATNASAMY, S., ZHAO, J., EE, C. T., CULLER, D. E., SHENKER, S., AND STOICA, I. 2005. Beacon Vector Routing: Scalable Point-to-Point Routing in Wireless Sensornets. In *Proc. 2nd Symposium on Networked Systems Design and Implementation*. ACM Press, New York, NY, 1–11.

GANESH, L. AND ZHAO, B. Y. 2005. Identity Theft Protection in Structured Overlays. In *Proc. 13th International Conference on Network Protocols* (Boston, MA). IEEE Computer Society Press, Los Alamitos, CA., 49–54.

GODFREY, B., LAKSHMINARAYANAN, K., SURANA, S., KARP, R. M., AND STOICA, I. 2004. Load Balancing in Dynamic Structured P2P Systems. In *Proc. 23rd INFOCOM Conference*. Vol. 4. IEEE Computer Society Press, Los Alamitos, CA., 2253– 2262.

GODFREY, P. B., SHENKER, S., AND STOICA, I. 2006. Minimizing Churn in Distributed Systems. In *Proc. SIGCOMM* (Pisa, Italy). ACM Press, New York, NY, 147–158.

GUMMADI, K., GUMMADI, R., GRIBBLE, S., RATNASAMY, S., SHENKER, S., AND STOICA, I. 2003. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proc. SIGCOMM* (Karlsruhe, Germany). ACM Press, New York, NY, 381–394.

GUPTA, A., LISKOV, B., AND RODRIGUES, R. 2004. Efficient Routing for Peer-to-Peer Overlays. In *Proc. 1st Symposium on Networked Systems Design and Implementation*. USENIX, Berkeley, CA, 113–126.

GUPTA, I., BIRMAN, K. P., LINGA, P., DEMERS, A. J., AND VAN RENESSE, R. 2003. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. In *Proc. 2nd International Workshop on Peer-to-Peer Systems* (Berkeley, CA). Lecture Notes on Computer Science, vol. 2735. Springer-Verlag, Berlin, 160–169.

HARVESF, C. AND BLOUGH, D. M. 2006. The Effect of Replica Placement on Routing Robustness in Distributed Hash Tables. In *Proc. 6th International Conference on Peer-to-Peer Computing*. IEEE Computer Society Press, Los Alamitos, CA., 57–6.

HILDRUM, K. AND KUBIATOWICZ, J. 2003. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks. In *Proc. 17th International Symposium on Distributed Computing (DISC)* (Sorrento, Italy). Lecture Notes on Computer Science, vol. 2848. Springer-Verlag, Berlin, 321–336.

HOLZ, T., STEINER, M., DAHL, F., BIERSACK, E., AND FREILING, F. 2008. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *Proc. 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (San Francisco, CA). USENIX, Berkeley, CA, 1–9.

KARGER, D. R. AND RUHL, M. 2006. Simple Efficient Load-Balancing Algorithms for Peer-to-Peer Systems. *Theory of Computing Systems 39,* 6, 787–804.

KLEINBERG, J. 2000. The Small-World Phenomenon: An Algorithm Perspective. In *Proc. 32nd Symposium on Theory of Computing* (Portland, OR). ACM Press, New York, NY, 163–170.

LESNIEWSKI-LAAS, C. 2008. A Sybil-proof one-hop DHT. In *Proc. 1st Workshop on Social Network Systems (SocialNets)* (Glasgow, Scotland). ACM Press, New York, NY, 19–24.

LEVINE, B. N., SHIELDS, C., AND MARGOLIN, B. N. 2006. A Survey of Solutions to the Sybil Attack. Tech. rep., University of Massachusetts, Amherst, Amherst, MA, USA.

LI, J., STRIBLING, J., MORRIS, R., KAASHOEK, M. F., AND GIL, T. M. 2005. A Performance vs. Cost Framework for Evaluating DHT Design Tradeoffs under Churn. In *Proc. 24th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 225–236.

LOS ANGELES TIMES. 2006. EDonkey File-Sharing Network Is Shut Down. [Online; accessed 19-November-2008]. `http://articles.latimes.com/2006/sep/13/business/fi-donkey13`.

MAHAJAN, R., RODRIG, M., WETHERALL, D., AND ZAHORJAN, J. 2004. Experiences Applying Game Theory to System Design. In *Proc. Workshop on Practice and Theory of Incentives in Networked Systems (PINS)* (Portland, Oregon, USA). ACM Press, New York, NY, 183–190.

MARGOLIN, N. B. AND LEVINE, B. N. 2007. Informant: Detecting Sybils Using Incentives. In *Proc. 11th International Conference on Financial Cryptography and Data Security*. Lecture Notes on Computer Science, vol. 4886. Springer-Verlag, Berlin, 192–207.

MAYMOUNKOV, P. AND MAZIÈRES, D. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. 1st International Workshop on Peer-to-Peer Systems* (Cambridge, MA). Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 53–65.

MESAROS, V., COLLET, R., GLYNN, K., AND VAN ROY, P. 2005. A Transactional System for Structured Overlay Networks. Tech. Rep. RR2005-01, Universit Catholique de Louvain (UCL). March.

MILLS, B. N. AND ZNATI, T. F. 2008. SCAR - Scattering, Concealing and Recovering Data within a DHT. In *Proc. 41st Annual Simulation Symposium*. IEEE Computer Society Press, Los Alamitos, CA., 35–42.

NAOR, M. AND WIEDER, U. 2003. A Simple Fault Tolerant Distributed Hash Table. In *Proc. 2nd International Workshop on Peer-to-Peer Systems* (Berkeley, CA). Lecture Notes on Computer Science, vol. 2735. Springer-Verlag, Berlin, 88–97.

PETERSON, L., BAVIER, A., FIUCZYNSKI, M. E., AND MUIR, S. 2006. Experiences Building PlanetLab. In *Proc. 7th Symposium on Operating System Design and Implementation* (Seattle, WA). USENIX, Berkeley, CA.

PIERRE, G. AND VAN STEEN, M. 2006. Globule: a Collaborative Content Delivery Network. *IEEE Communications Magazine 44,* 8 (Aug.), 127–133.

PLANTIKOW, S., REINEFELD, A., AND SCHINTKE, F. 2007. Transactions for Distributed Wikis on Structured Overlays. In *Proc. 18th Workshop on Distributed Systems: Operations and Management*. Lecture Notes on Computer Science, vol. 4785. Springer-Verlag, Berlin, 256–267.

POUWELSE, J., GARBACKI, P., WANG, J., BAKKER, A., YANG, J., IOSUP, A., EPEMA, D., REINDERS, M., VAN STEEN, M., AND SIPS, H. 2007. Tribler: A social-based peer-to-peer system. *Concurrency & Computation: Practice and Experience 20,* 2 (Feb.), 127–138.

RAMASUBRAMANIAN, V. AND SIRER, E. G. 2004a. Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In *Proc. 1st Symposium on Networked Systems Design and Implementation* (San Francisco, CA). USENIX, Berkeley, CA, 99–112.

RAMASUBRAMANIAN, V. AND SIRER, E. G. 2004b. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proc. SIGCOMM* (Portland, OR). ACM Press, New York, NY.

RAO, A., LAKSHMINARAYANAN, K., SURANA, S., KARP, R. M., AND STOICA, I. 2003. Load Balancing in Structured P2P Systems. In *Proc. 2nd International Workshop on Peer-to-Peer Systems* (Berkeley, CA). Lecture Notes on Computer Science, vol. 2735. Springer-Verlag, Berlin, 68–79.

RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SCHENKER, S. 2001. A Scalable Content-Addressable Network. In *Proc. SIGCOMM* (San Diego, CA). ACM Press, New York, NY, 161–172.

REIDEMEISTER, T., BOHM, K., WARD, P. A. S., AND BUCHMANN, E. 2005. Malicious Behaviour in Content-Addressable Peer-to-Peer Networks. In *Proc. 3rd Annual Communication Networks and Services Research Conference*. IEEE Computer Society Press, Los Alamitos, CA., 319–326.

RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. 2004. Handling Churn in a DHT. In *Proc. USENIX Annual Technical Conference* (Boston, MA). USENIX, Berkeley, CA, 127–140.

RHEA, S. C. 2005. OpenDHT: A Public DHT Service. Ph.D. thesis, University of California at Berkeley. Adviser-John Kubiatowicz.

RODRIGUES, R. AND LISKOV, B. 2005. High Availability in DHTs: Erasure Coding vs. Replication. In *Proc. 4th International Workshop on Peer-to-Peer Systems*. Lecture Notes on Computer Science, vol. 3640. Springer-Verlag, Berlin, 226–239.

ROWAIHY, H., ENCK, W., MCDANIEL, P., AND LA PORTA, T. 2007. Limiting Sybil Attacks in Structured Peer-to-Peer Networks. In *Proc. 26th INFOCOM Conference* (St. Louis, MO). IEEE Computer Society Press, Los Alamitos, CA., 2596–2600.

ROWSTRON, A. AND DRUSCHEL, P. 2001a. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. Middleware*. Lecture Notes on Computer Science, vol. 2218. Springer-Verlag, Berlin, 329–350.

ROWSTRON, A. AND DRUSCHEL, P. 2001b. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proc. 18th Symposium on Operating System Principles* (Banff, Canada). ACM Press, New York, NY, 188–201.

SAIA, J., FIAT, A., GRIBBLE, S. D., KARLIN, A. R., AND SAROIU, S. 2002. Dynamically Fault-Tolerant Content Addressable Networks. In *Proc. 1st International Workshop on Peer-to-Peer Systems* (Cambridge, MA). Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 270–279.

SÁNCHEZ ARTIGAS, M., GARCÍA LÓPEZ, P., AND GÓMEZ SKARMETA, A. F. 2005. A Novel Methodology for Constructing Secure Multipath Overlays. *IEEE Internet Computing 9,* 6, 50–57.

SCHEIDELER, C. 2005. How to Spread Adversarial Nodes?: Rotate! In *Proc. 37th Symposium on Theory of Computing* (Baltimore, MD). ACM Press, New York, NY, 704–713.

SCHÜTT, T., SCHINTKE, F., AND REINEFELD, A. 2008. Scalaris: Reliable Transactional P2P Key/Value Store. In *Proc. 7th ACM SIGPLAN Workshop on ERLANG (ERLANG)* (Victoria, BC, Canada). ACM Press, New York, NY, 41–48.

SINGH, A., NGAN, T.-W., DRUSCHEL, P., AND WALLACH, D. S. 2006. Eclipse Attacks on Overlay Networks: Threats and Defenses. In *Proc. 25th INFOCOM Conference* (Barcelona, Spain). IEEE Computer Society Press, Los Alamitos, CA., 1–12.

SIT, E. AND MORRIS, R. 2002. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *Proc. 1st International Workshop on Peer-to-Peer Systems* (Cambridge, MA). Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 261–269.

SKYPE LIMITED. 2008. Skype. `http://www.skype.com`.

SRIVATSA, M. AND LIU, L. 2004. Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. In *Proc. 20th Annual Computer Security Applications Conference* (Tucson, AZ). IEEE Computer Society Press, Los Alamitos, CA., 252–261.

STADING, T., MANIATIS, P., AND BAKER, M. 2002. Peer-to-Peer Caching Schemes to Address Flash Crowds. In *Proc. 1st International Workshop on Peer-to-Peer Systems* (Cambridge, MA). Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 203–213.

STEINER, M., BIERSACK, E. W., AND ENNAJJARY, T. 2007. Towards Scalable and Robust Overlay Networks. In *Proc. 6th International Workshop on Peer-to-Peer Systems*. Bellevue, WA.

STEINER, M., EN-NAJJARY, T., AND BIERSACK, E. W. 2007. Exploiting KAD: Possible Uses and Misuses. *ACM Computer Communications Review 37,* 5, 65–70.

STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D. R., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. 2003. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking 11,* 1, 17–32.

SZYMANIAK, M., PIERRE, G., AND VAN STEEN, M. 2004. Scalable Cooperative Latency Estimation. In *Proc. 10th International Conference on Parallel and Distributed Systems* (Newport Beach, CA). IEEE Computer Society Press, Los Alamitos, CA., 367–376.

SZYMANIAK, M., PRESOTTO, D., PIERRE, G., AND VAN STEEN, M. 2008. Practical Large-Scale Latency Estimation. *Computer Networks 52,* 7, 1343–1364.

TANG, C., XU, Z., AND MAHALINGAM, M. 2003. pSearch: Information Retrieval in Structured Overlays. *ACM Computer Communications Review 33,* 1, 89–94.

WALLACH, D. 2002. A Survey of Peer-to-Peer Security Issues. In *Proc. Int'l Symp. Softw. Security* (Tokyo, Japan). Lecture Notes on Computer Science, vol. 2609. Springer-Verlag, Berlin, 42–57.

WANG, H., ZHU, Y., AND HU, Y. 2005. An Efficient and Secure Peer-to-Peer Overlay Network. In *Proc. 30th Local Computer Networks*. IEEE Computer Society Press, Los Alamitos, CA., 764–771.

WANG, P., OSIPKOV, I., HOPPER, N., AND KIM, Y. 2007. Myrmic: Secure and Robust DHT Routing. Tech. rep., Digital Technology Center, University of Minnesota at Twin Cities.

YANG, K.-H. AND HO, J.-M. 2006. Proof: A DHT-Based Peer-to-Peer Search Engine. In *Proc. International Conference on Web Intelligence (WI)*. IEEE Computer Society Press, Los Alamitos, CA., 702–708.

YU, H., GIBBONS, P. B., KAMINSKY, M., AND XIAO, F. 2008. SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks. In *Proc. International Symposium on Security and Privacy*. IEEE Computer Society Press, Los Alamitos, CA., 3–17.

YU, H., KAMINSKY, M., GIBBONS, P. B., AND FLAXMAN, A. 2006. SybilGuard: Defending Against Sybil Attacks via Social Networks. In *Proc. SIGCOMM* (Pisa, Italy). ACM Press, New York, NY, 267–278.

YU, Y., LEE, S., AND ZHANG, Z.-L. 2005. Leopard: A Locality Aware Peer-to-Peer System with No Hot Spot. In *Proc. 4th International Networking Conference (NETWORKING 2005)*. Lecture Notes on Computer Science, vol. 3462. Springer-Verlag, Berlin, 27–39.

ZHAO, B., HUANG, L., STRIBLING, J., RHEA, S., JOSEPH, A., AND KUBIATOWICZ, J. 2004. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communication 22,* 1 (Jan.), 41–53.

ZHOU, L. AND VAN RENESSE, R. 2004. P6P: A Peer-to-Peer Approach to Internet Infrastructure. In *Proc. 3rd International Workshop on Peer-to-Peer Systems* (La Jolla, CA). Lecture Notes on Computer Science, vol. 3279. Springer-Verlag, Berlin, 75–86.

ZHU, Y. AND HU, Y. 2005. Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. *IEEE Transactions on Parallel and Distributed Systems 16,* 4, 349–361.