Design and Evaluation of Persea, a Sybil-Resistant DHT

Mahdi Nasrullah Al-Ameen
The University of Texas at Arlington
Arlington, TX, USA
mahdi.al-ameen@mavs.uta.edu

Matthew Wright
The University of Texas at Arlington
Arlington, TX, USA
mwright@cse.uta.edu

ABSTRACT

P2P systems are inherently vulnerable to Sybil attacks, in which an attacker creates a large number of identities and uses them to control a substantial fraction of the system. We propose Persea, a novel P2P system that derives its Sybil resistance by assigning IDs through a bootstrap tree, the graph of how nodes have joined the system through invitations. Unlike prior Sybil-resistant P2P systems based on social networks, Persea does not rely on two key assumptions: (1) that the social network is fast mixing and (2) that there is a small ratio of attack edges to honest nodes. Both assumptions have been shown to be unreliable in real social networks. A node joins Persea when it gets an invitation from an existing node in the system. The inviting node assigns a node ID to the joining node and gives it a chunk of node IDs for further distribution. For each chunk of ID space, the attacker needs to socially engineer a connection to another node already in the system. The hierarchical distribution of node IDs confines a large attacker botnet to a considerably smaller region of the ID space than in a normal P2P system. We then build upon this hierarchical ID space to make a distributed hash table (DHT) based on the Kad network. The Persea DHT uses a replication mechanism in which each (key, value) pair is stored in nodes that are evenly spaced over the network. Thus, even if a given region is occupied by attackers, the desired (key, value) pair can be retrieved from other regions. We evaluate Persea in analysis and in simulations with social network datasets and show that it provides better lookup success rates than prior work with modest overheads.

Categories and Subject Descriptors

C.2.4 [**Communication Networks**]: Distributed Systems-Distributed applications

Keywords

Sybil attack; security; social DHT

1. INTRODUCTION

Peer-to-peer (P2P) systems are highly susceptible to Sybil attacks, in which an attacker creates a large number of pseudonymous entities and uses them to gain a disproportionately large influence over the system [6, 7, 9, 13]. A malicious node may present multiple identities to a P2P system that appear and function as distinct nodes. By becoming part of the P2P system, the Sybil attackers can then collude to launch further attacks to subvert the system's operation, such as taking over resources and disrupting connectivity. Researchers have documented this vulnerability in real-world systems, including the Maze P2P file-sharing system [14, 34] and the Vanish data storage system [33].

Many P2P systems employ a distributed hash table (DHT), which provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Well-known DHT-based systems, or simply DHTs, include Chord [27], CAN [22], Pastry [23], and Kademlia [17]. Kademlia was the basis for both the Kad network and Vuze, DHTs used in the popular BitTorrent file-sharing P2P system with millions of users each.

Recent research has focused on leveraging information from social networks to make the system robust against Sybil attackers, resulting in a number of decentralized *community-based* schemes [11, 12, 19, 28, 31, 36, 37]. The key to these approaches is the idea that honest and malicious nodes can be effectively partitioned into two subgraphs in the social network. The link between an honest node and a malicious peer is called an *attack edge*, which represents an act of social engineering to convince the honest node to add the link.

These mechanisms are based on two key assumptions:

- Online social networks are *fast mixing*, meaning that a random walk in the honest part of the network approaches the uniform distribution in a small number of steps.
- The number of attack edges are rather limited in an online social network, as benign users are unlikely to accept friend requests from strangers.

Recent studies [3, 20, 30, 35], however, show that the above assumptions do not hold in real-world social networks. Thus, the effectiveness of these schemes are left as an open question. In particular, Mohaisen et. al. [20] show that in social graphs where edges correspond to strong real-world trust (e.g., Epinions, Physics co-authorship, DBLP, etc.), the mixing-time is not as fast as the community-based schemes assume. Thus, the schemes do not perform well on "trusted" social graphs. Further, Viswanath et al. have shown that a number of Sybil defenses are ineffective for slower-mixing, highly *modular* social networks [30].

At the same time, recent work shows that the probability with which fake identities are accepted as friends is much higher than anticipated [25], with studies reporting that users accept $40-80\%$ of friendship requests from strangers [3, 5]. This implies that that number of attack edges may not be small, even as a fraction of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIA CCS'14, June 4–6, 2014, Kyoto, Japan.
Copyright 2014 ACM 978-1-4503-2800-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2590296.2590326.

the number of honest nodes. As the assumption of few attack edges breaks down, prior schemes do not work well. The Whānau scheme, for example, requires 10-100 times the overhead to get successful lookups when there is one attack edge per honest node.
Contributions. These findings about online social networks mean that it remains an open research problem to design an effective Sybil defense that does not rely on the assumptions of a fast-mixing social network and a small number of attack edges. In this paper, we propose a Persea, a new Sybil-resistant DHT that addresses these problems.

In Persea, existing nodes invite new peers to join the system. When a node joins the DHT it also creates a link with the inviting node. This creates a *bootstrap tree*, linking nodes together through invitation relationships. We then leverage the social relationships among honest nodes for building a robust DHT. We develop a mechanism to distribute hierarchical node IDs based on the bootstrap tree. In this mechanism, when a node joins the system after getting an invitation, the inviting node assigns a node ID to the joining node and gives it a chunk of node IDs for further distribution.[1] ID and chunk assignment are certified in a chain from the root of the bootstrap tree. The use of a bootstrap tree relies heavily on the root nodes in the early stages of system deployment, but by careful design of how IDs are certified, the reliance on these nodes can be minimized. Based on this ID distribution method, we then design a DHT routing table and lookup protocol based on the Kademlia design [17], with modest modifications to enhance robustness.

The Persea approach offers a number of important advantages over existing schemes:

- Persea does not depend on assuming that there are few attack edges compared to the benign nodes. Let g denote the number of attack edges and n be the honest nodes. Our simulation results show that even for $g/n = 1.0$—i.e. one attack edge for every honest node—95.6% of lookups still succeed.
- Although Persea may work better for fast-mixing social networks, our system is also dependable for slow-mixing social networks. For the slowest-mixing social network we evaluated on and $g/n = 0.5$, Persea lookups succeed 96.2% of the time.
- The hierarchical distribution of node IDs limits the attackers to isolated regions in the ID space. Even for $g/n = 1.0$, only 0.7% of the total ID space is occupied by attackers.
- Building a bootstrap tree is more realistic than assuming that the clients have access to authenticated lists of social network connections or activity levels from a system like Facebook; such lists may also bear little resemblence to social connections inside the P2P system.

Additionally, Persea has several other important features:

- Although we test Persea with a DHT routing table design similar to Kademlia [17], which is widely used, it can be adapted to other DHTs.
- IDs are certified, making attacks based on ID forging impossible outside of attacker-controlled ID ranges.
- Varying the number of attackers per attack edge does not significantly affect the lookup success rate in Persea.
- Sybil-resilient system design is inherently probabilistic [28], and thus our system provides resilience against further attacks, including *denial of service*, *node ID hijacking*, and *node insertion*, that may be launched by the existing Sybil nodes in the system.

[1]The name Persea comes from a tree in ancient Egyptian mythology (also called the ished tree) upon whose leaves the Gods wrote the names of the pharaohs (see `http://www.touregypt.net/featurestories/treegoddess.htm`).

2. RELATED WORK

Due to the power and generality of the Sybil attack, a large number of defenses have been proposed [1, 13]. In this section, we examine the use of social networks for Sybil defense.
Random walks over social networks. A number of works have proposed Sybil detection techniques or Sybil resistance based on random walks over a social network [8, 11, 12, 19, 28, 31, 36, 37]. The basic idea is that we can divide the social network into a Sybil region and an honest region connected via a small number of attack edges (a *small cut*). Random walks starting from the honest region have a low probability of ending in the Sybil region. This can be leveraged in a variety of ways, leading to detection mechanisms [8, 30, 31, 36, 37], admission control mechanisms [28], and Sybil-resistant P2P designs [11, 12, 19].

These mechanisms require the absence of small cuts within the honest region in the underlying social network. Equivalently, the honest region should be fast-mixing. Mohaisen et al., however, show that the mixing time of many real social networks is slower than the mixing times assumed by these works [20]. Additionally, many real-world social networks fail to satisfy the other requirements of the systems, either because a significant fraction of nodes are sparsely connected or the users are organized in small tightly-knit communities, which are sparsely interconnected [29]. Finally, even perfect community-based defenses would fail against existing Sybil attacks due to the lack of a Sybil community structure [1, 35].
DHTs Built on Social Networks. Three key related works are the Sybil-resistant DHT, Whānau, and X-Vine, and we compare these with Persea in Table 1.

Danezis et al. propose a Sybil-resistant DHT routing protocol [7] that makes use of latent social information that is present in the bootstrap graph of the network. While they pioneered the bootstrap graph model that inspires our work, we note two major shortcomings of their approach. First, the DHT layer is built on top of the bootstrap graph, creating a second overlay layer. This means that one hop in the DHT layer corresponds to a number of hops in the bootstrap graph, each of which is itself an Internet connection. This adds substantial delay and overhead beyond the DHT. Second, the scheme provides diversity in the bootstrap graph at the expense of following the DHT lookup graph. This further extends the lookup delay and overhead. When there is one attack edge per honest node, a lookup in a network of just 100 honest nodes requires over 50 requests to succeed.

Lesniewski-Laas proposed Whānau [11, 12], in which a node constructs its routing table through independent random walks and recording the final node in each walk as the finger in its routing table. Whānau requires significant routing table state on the order of $O(\sqrt{m}\log m)$, where m is the number of objects stored in the DHT. Mittal et al. point out that the network overhead for maintaining this state can be substantial (e.g. 800 KBps per node) [19]. Further, Whānau overheads increase further if large numbers of attack edges grows, such that for $g/n = 1.0$, either routing table sizes grow by 10-100 times or lookup redundancy must grow to 50 or more compared to the regular protocol [12].

X-Vine [19] is a DHT built by communicating only over social network edges. Honest peers rate-limit the number of paths that are allowed to be built over their adjacent edges, which helps to limit the number of Sybil nodes that can join the system. X-Vine, however, relies on the fast-mixing assumption and was only evaluated with a small number of attack edges (one for every ten honest nodes). Rate-limiting only works when there are few attack edges,

Table 1: Comparison of Sybil-resistant DHTs. Grey squares indicate limitations of the system. [($\star$): varies from less than 20 to over 50 depending on the network and routing table sizes, (†): based on limited results [7]]

	Whānau	X-Vine	Sybil-resistant	**Persea**
Reliance on fast mixing	Yes	Yes	No	**No**
Requires that users provide a full social network	Yes	Yes	No	**No**
Lookups succeed at $g/n = 1.0$	Yes	Not tested	Yes	**Yes**
Total messages for a lookup at $g/n = 1.0$	$\mathrm{O}(1)\star$	Unknown	$\sim 0.53 \times n$†	$\mathbf{O}(\log(n))$
Total messages for a lookup at $g/n = 0.1$	$\mathrm{O}(1)$	$\mathrm{O}(\log(n))$	$\sim 0.44 \times n$†	$\mathbf{O}(\log(n))$
Routing table size	$\Omega(\sqrt{n})$	$\mathrm{O}(\log(n))$	$\mathrm{O}(\log(n))$	$\mathbf{O}(\log(n))$

as an attacker with one attack edge per honest node should have the same rate of paths as the honest nodes in the system. Thus, very high levels of redundancy may be required for successful lookups when the number of attack edges is high.

We note that both Whānau and X-Vine use public keys to provide important security properties.

3. GOALS AND ASSUMPTIONS

In this section, we describe our system design goals and our system and attacker models.

3.1 Design Goals

In designing Persea, we have the following *Design Goals*:

1. Sybil-resistent lookups. The system should maintain high success rates for lookups in the presence of Sybil attackers seeking to undermine the system's operations, even if the number of attack edges grows to high levels (e.g. $g/n = 1.0$). Prior work does not handle large numbers of attack edges.

2. Works on a range of social networks. The system should work well for any reasonable structure in the underlying social network, even if the social network is not fast mixing. Most existing systems rely on the fast-mixing property.

3. Builds its own social network. The system should build its own social network to defeat Sybil attacks and not rely on users or online social networks (OSNs).

4. Moderate overhead. System overheads, such as routing table size, the number of overlay hops to perform a lookup, and the number of redundant lookups, should be kept to reasonable levels.

Regarding *Design Goal #3*, we find that most existing Sybil defenses require a social network without specifying where the network information comes from. One possible source would be from user input, but it is unclear how to motivate users to provide more than a minimal amount of information. Note that the fast-mixing property needed by many systems is particularly hard to obtain from a limited subset of edges. Another possible source would be to get the data from an OSN, but this has many issues. First, a few users may not even have accounts on the selected OSN, while others may not want to share their OSN network with the P2P system for privacy reasons. Also, relationships between users in the P2P system may be vastly different than those in the OSN, as even a popular P2P system would likely have only a fraction of all OSN members. Finally, an OSN like Facebook may attempt to block the P2P system from mining its network information, which is generally protected due to its value for advertising. This is especially true for mining enough information to formulate *interaction graphs* that only include edges between active communication partners and are thus more reliable than friendship graphs for use in Sybil defense [3, 4, 10, 32].

3.2 System Model

The key assumptions about our system involve how we model users and social links. Participating in Persea requires an invitation from an existing user, and new users can request invitations from an existing user if they have her out-of-band contact information, such as an email address. The P2P client UI can integrate a request in the system with a code sent to the email address for verification. We assume that honest users generally accept invitation requests from people they know in real life. Thus, the links created from the set of accepted invitations form a subgraph of the social network known as a *bootstrap graph*. The structure of this network may vary and it is beyond the scope of this paper to identify requirements for either the bootstrap graph or the underlying social network. We instead evaluate our system with a variety of social networks with different properties to show that Persea is effective for a broad range of networks.

The bootstrap graph starts with a small set of *bootstrap nodes*. We make the following assumptions about the early stages of building the system: (1) bootstrap nodes are honest and (2) bootstrap nodes trust each other. The creation of a bootstrap graph starting from a trusted set of nodes allows us to obviate the use of OSN data and thus achieve *Design Goal #3*. The assumptions are reasonable, since the bootstrap nodes together take the initiative to start building the system. As the system passes the initial stage of growth and reaches a modest population size, we can gradually relax these requirements. In particular, the bootstrap nodes no longer need to be honest or trust each other for the system to work well.

As with the prior works on leveraging social networks for Sybil-resistent DHTs [12, 19] (see §2), Persea uses public keys to certify IDs. Also like X-Vine [19], we suggest to ensure the integrity of content or services by using self-certifying identifiers on returned lookup results [2, 18].

3.3 Attacker Model

We assume that the attacker is a single entity, or a small and highly coordinated group, with access to substantial computational and network resources. For example, the attacker can control a botnet. The attacker's goal is to disrupt the operations of the P2P network, and the lookup operation in particular, as lookups are the mechanism for distribution of information and resources in a DHT. To do this, the attacker seeks to add malicious nodes to the system and then have them disrupt the system's activities.

Adding malicious nodes. The attacker can first attempt to find users of the P2P system and socially engineer them into giving up an invitation. Such an invitation is called an *attack edge* in the social network. We note that targeting specific users may be desirable when attacking Persea, but this is difficult without external information about exactly who the key users are and how to contact

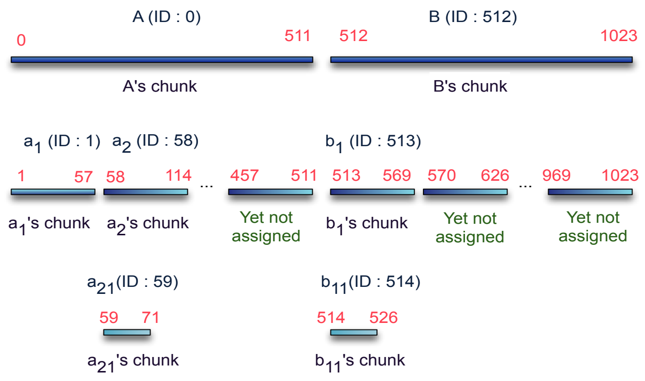


Figure 1: Hierarchical Distribution of Node IDs

them. This information is not meant to be publicly available. Instead, the attacker gathers invitations from anyone using the system who he can identify and trick. We model this as getting invitations via uniform selection from the set of honest users. We have not evaluated Persea for the case that targeted attacks are feasible.

In our evaluations, we limit the number of attack edges that an attacker can get via social engineering to a ratio of at most one per honest node on average. We note that prior schemes face very high overheads at such ratios, which are realistic given studies showing that OSN users accept fake and duplicate accounts at a rate of $40 - 80\%$ [3, 4]. Unlike most OSNs, we attempt to limit attack edges via three methods: (i) Invitation requests should require that a code be sent through an out-of-band channel; (ii) When using such a code to issue an invitation, the client UI can warn users to not invite strangers; and (iii) Each user has a limited number of invitations (the amount used would be shown to users) due to our system design (see §4). Thus, there is an incentive for users to not be overly promiscuous with invitations and to only invite peers based on actual social connections.

Once an invitation is accepted, the attacker may proceed to invite an arbitrary number of malicious peers. This is unlike X-Vine and detection approaches like SybilLimit that use rate limits to effectively keep the attacker to a small number of Sybil nodes per attack edge [19, 36]. In such approaches, an attacker can use a large number of attack edges (e.g. one per honest node) to overcome the rate limits and overwhelm the system.

The attacker does not invite honest nodes, as this provides him with no advantage in our system. An attacker node could leave the system and join again under a different identity, but doing so provides no advantages, as we do not employ detection or reputation mechanisms. Further, the attacker might lose an attack edge, and so we do not model attacker churn in our evaluation.

Attacking lookups. An attacker with a large number of malicious peers then proceeds to attack the system. Rather than simply dropping lookup requests, the attacker nodes respond with the closest other attacker nodes to the requested keys. This gives the attacker nodes the best chance to appear later in the lookup path and thus manipulate the final lookup results. If, at the end of the lookup, an attacker node is finally asked for the requested information, it will then drop the request.

We also assume that the attacker attempts to manipulate DHT routing tables. The routing tables in our system, which is based on Kad, are opportunistically modified based on intermediate query results, so the attacker's strategy of returning other attacker nodes benefits him in this way as well. We assume that the attacker does not make any lookups.

4. SYSTEM DESIGN

We now describe the Persea design. Persea consists of two layers: a social network layer (the *bootstrap graph*) and a DHT layer. In this paper, an *edge* refers to a link between two nodes in the DHT layer. The social network and DHT are simultaneously built starting with a set of bootstrap nodes. The bootstrap nodes assign IDs to themselves such that they are evenly spaced over the circular ID space. Thus, the ID space of the DHT is divided into one region for each bootstrap node.

A new peer must join the Persea system through an invitation from an existing node in the network. In general, it is expected that a node that is invited is socially known to the inviting peer. Thus, inviting a new node is a feature of the social network layer of Persea. More specifically, when a node is invited, it becomes a child of the inviting peer in the bootstrap graph. The inviting peer also gives it an ID and a *chunk* of IDs that it can use to invite more nodes to join the network.

The number of nodes that a peer can invite is limited by the number of IDs in its chunk. Thus, there is an incentive for peers to only invite other peers based on actual social connections so that it does not run out of IDs.

The DHT layer of Persea is based largely on Kademlia [17], a DHT that is widely adopted for the BitTorrent P2P file-sharing system. In particular, we use Kademlia's XOR distance metric to perform routing and k-buckets to store contacts. The main difference in Persea is that IDs are replicated evenly around the ID space for greater resiliency given our ID distribution scheme.

4.1 Hierarchical ID Space

We now describe how IDs are distributed in Persea. Each bootstrap node has a contiguous range of IDs called a *chunk*, which includes the bootstrap node's ID. A bootstrap node divides its chunk of IDs into sub-chunks based on the *chunk-factor*, a system parameter.

When a bootstrap node invites a peer to join the system, it assigns the new node an ID from one of its sub-chunks and gives it control over the rest of the sub-chunk. The newly joined node becomes the authority for distributing IDs from the given sub-chunk and uses this to invite more nodes to join the system. Based on the invitation-relationship among peers, a *bootstrap tree* is formed, in which an inviter node is the parent of its invited peers. If we have more than one bootstrap node, then we would have a forest of trees, where each bootstrap node is the root of a tree. The chunk-factor and size of the ID space define the maximum possible height and width of the trees.

This hierarchical ID distribution mechanism features the advantage that even if the attacker compromises a node in the system, and through it a large number of malicious nodes join the network, they will still be confined to a particular region of the ID space.

We briefly explain the mechanism with an example illustrated in Fig. 1. Let A and B be two bootstrap nodes that initiate the system. If we consider a b-bit ID space, then the total number of IDs in the DHT n_{max} would be 2^b. In this toy example, we consider a 10-bit ID space, so $n_{max} = 2^{10}$. If Z is the number of bootstrap nodes, each bootstrap nodes has $\lfloor \frac{n_{max}}{Z} \rfloor$ IDs in its chunk. To simplify the discussion, we ignore issues of uneven division of IDs. In this example, both node A and node B have 512 IDs. The lowest ID in a chunk is assigned to the bootstrap node itself and the remaining IDs are for further distribution to new nodes. In this example, node A's ID is 0 and the interval $[1, 511]$ is its chunk of IDs for further distribution.

```
Input: $S_c$: An integer, denoting the number of sub-chunks
Output: B[ ]: The integer array of size $S_c$
Variables: i, p, q: Integers
Operations: a=0
for $i = 1$ to $S_c$ do
    if $i == 2^a$ then
        a=a+1 B[i]=floor($S_c$/$2^a$)
    end
    else
        B[$i$] = B[$i-1$] + floor($S_c$/$2^{a-1}$)
    end
end
```

Algorithm 1: Computing the balanced ordering of sub-chunks for distribution

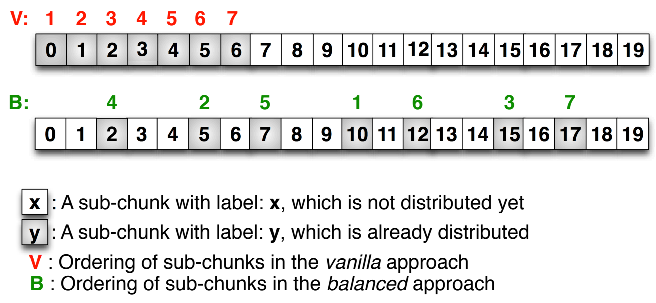

V: 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
B: 4 2 5 1 6 3 7
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
x : A sub-chunk with label: x, which is not distributed yet
y : A sub-chunk with label: y, which is already distributed
V : Ordering of sub-chunks in the vanilla approach
B : Ordering of sub-chunks in the balanced approach

Figure 2: Ordering of sub-chunks for distribution

Each node divides its chunk into sub-chunks based on the chunk-factor. Let n_c be the number of IDs in a chunk and n_s represent the number of IDs in each of its sub-chunks. In
a chunk, the lowest ID is assigned to the owner node and the remaining IDs are for further distribution. Thus $n_c - 1$ represents the number of IDs in a chunk available for distribution by the owner node. If the chunk factor is c_f $(0 \leq c_f \leq 1)$ then n_s would be $\lfloor (n_c - 1)^{c_f} \rfloor$. The number of sub-chunks (S_c) that can be created from a chunk is $\lfloor \frac{n_c - 1}{n_s} \rfloor + 1$. This also represents the maximum number of nodes that can be invited by a node having chunk of size n_c.

Let $c_f = 0.65$ in this example. Node A divides its chunk into nine sub-chunks where each sub-chunk has 57 IDs (the last sub-chunk has 55 IDs). Node a_1 joins the network after getting an invitation from node A and A assigns a sub-chunk to a_1. The lowest ID in this sub-chunk is 1, which is assigned as a_1's ID, and the interval $[2, 57]$ represents the remaining IDs of the sub-chunk that are for further distribution by a_1. Another node a_2 joins the network through an invitation from A. Using $c_f = 0.65$, both a_1 and a_2 divide their chunks into five sub-chunks, where each sub-chunk has 13 IDs (the last one has four). Node a_{21} joins the network through the invitation from node a_2, which assigns it the ID 59. Nodes b_1 and b_{11} join the network through the invitations from nodes B and b_1, respectively.

We assume that each node knows the value of Z, b, and c_f. Thus, if a node assigns an ID from its chunk to a joining node, the joining node can easily verify it, because any node in Persea can calculate the chunk distribution.

Balanced ordering of sub-chunks. As shown at the top of Fig. 2, if chunks are assigned in consecutive order, the distribution of ID space will be uneven. To maintain reasonable load distribution, we instead space out the chunk distribution as shown in the lower part of Fig. 2.

More specifically, a node doles out each sub-chunk so as to divide its chunk's ID space into the most balanced distribution possible at each moment. Using Fig. 2, in which there are 20 sub-chunks, the node will first allocate sub-chunk 10, which divides its ID space in half between itself and its first child. Further allocations will be to cut the total ID space into quarters (chunks 5 and 15), eighths (chunks 2, 7, 12, and 17), and so on. More formally, a node issues its chunks according to Algorithm 1.

4.2 Certification of IDs and Chunk Allocations

Persea employs a simple public key infrastructure to protect the advantages of hierarchical ID distribution from fraudulent ID and chunk ownership claims. In Persea, each node has a certificate, signed by its parent in the bootstrap tree, containing its ID, its public key, the parent's ID, and the last ID of its chunk. The ID of the node itself is the first ID of its chunk, and thus the chunk ranges from the node's ID to the last ID. The information in the certificate helps to prevent attacks based on fraudulent node creation. We discuss the resilience of Persea to such attacks in §5.

To ensure that its certificate can be found, each node takes the hash of its ID to get a target ID and publishes the certificate to the Persea DHT, using replication (see §4.3). When node P contacts node Q in the DHT, it validates Q's ID as follows. First, it uses the DHT to find Q's certificate, which is signed by Q's parent Q_1. Q_1's certificate is also obtained from the DHT, and the public key it contains is used to verify Q's certificate. Each replica that stores Q's certificate will also look up Q_1's certificate in the DHT to prevent an attacker from storing fake certificates. Note that Q_1's certificates have already been verified, and so on logically up to the bootstrap node. The bootstrap node certificates themselves can be distributed along with the P2P software or during the invitation process.

Note that the Persea lookup for Q_1's certificate obtains all R replicas. If there are any discrepancies between the replicas, P can continue to look up additional ancestors in the bootstrap tree up to the bootstrap node itself, if necessary. This provides resilience in the face of content poisoning attacks. If such attacks are widespread as an annoyance, the lookups for ancestors can be made probabilistic to avoid excessive overhead while still ensuring that fraudulent ID claims are detected quickly.

By placing certificates in the DHT, we avoid relying too much on the bootstrap nodes. When the system is first deployed, the bootstrap nodes are highly trusted as being in charge of large chunks of the ID space. After a number of peers have joined, however, much of the ID space is allocated. If a bootstrap or other node is compromised, it may attempt to issue certificates for malicious peers in space that has already been allocated. To prevent this, we propose the following mechanism: Nodes who store a certificate for a peer will not allow this certificate to be replaced by another node's information. We expect that this rule can be made more flexible by establishing consensus that a node has left the system or should be expelled, but protocols for doing so are beyond the scope of this paper.

4.3 Replication Mechanism

In this section, we describe our replication mechanism, which is the key difference between the Persea DHT layer and Kademlia.

As with other DHTs, we calculate the key for a given (key, value) pair by taking the consistent hash (e.g. SHA-1) of a search key, such as a file name. The node with the closest ID to the key in terms of the XOR of the key and the ID, interpreted as an integer, is deemed to be the owner of the key and should store the (key, value) pair. A lookup for the key should return this owner. In Persea, we replicate each (key, value) pair over R owners that are evenly

Table 2: Topologies [MT: mixing time]

Network	Abbrv.	Nodes	Edges	MT
advogato	**adv**	6551	51332	2.3
hamsterster	**ham**	2426	16631	3.0
youtube	**ytub**	15088	5574249	3.0
ca-AstroPh	**astro**	18772	396160	8.0
flickr	**flic**	80513	5899882	11.5
catster	**cat**	149700	5449275	8.0

spaced over the circular ID space. Thus, even if a region is occupied by the malicious peers, the redundant lookup operations can retrieve the desired (key, value) pair from owners in other regions of the network.

More specifically, when the initiator intends to store or retrieve a (key, value) pair in Persea, it calculates the ID of the target nodes as follows. Assume a b-bit ID space, such that $n_{max} = 2^b$. We virtually divide the ID space into R regions where each region (except the last one) accommodates at most $D = \lfloor \frac{n_{max}}{R} \rfloor$ IDs, and the last region has $n_{max} - D \times (R-1)$ IDs. The interval $[D \times r, D \times (r+1) - 1]$ for $0 \leq r < R-1$ represents the IDs that are in the rth region; the last region spans $[D \times (R-1), n_{max} - 1]$. A node ID i is replicated to each other region by taking $(i + D \times r) \bmod n$ for $1 \leq r < R$.

4.4 Routing Table Organization and Lookup

Our routing table organization and lookup mechanism follow closely to the model of Kademlia [17], though we note that other models could be followed. Here, we briefly describe the mechanisms for completeness.

Routing table organization. In the DHT layer, each node maintains a routing table consisting of b node lists for a b-bit ID space. Each list, or *k-bucket*, contains up to k entries, each of which contains the IP address, port, ID, and public key of another node. The ID of a node in the b^{th} k-bucket of a node with ID i should share the first $b-1$ bits of i and have a different b^{th} bit from i.

Lookup mechanism. To initiate a **lookup**(key) request, the querying node contacts the α nodes in its k-buckets that are the closest ones to the desired key. Each of the α nodes sends the initiator β IDs from its k-bucket closest to the target node. If any of the β nodes is found not alive, the next closest node to the desired key, which is alive, is returned to the initiator. From the set of returned IDs, the initiator selects α nodes for the next iteration. This process is iterated until the target is found or no nodes are returned that are closer than the previous best results.

In Persea, the initiator performs R such independent parallel lookup operations and calculates the ID of the target nodes according to the mechanism described in §4.3. When an owner is found from any of R independent lookups, the initiator sends the owner a message for either the store (**put**$(key, value)$) or retrieval (**get**(key)) operation.

5. SECURITY ANALYSIS

In this section, we examine the possible avenues of attack against Persea. We focus primarily on examining the resilience of Persea against attacks on ID control.

5.1 Attacks on ID Control

An attacker could attempt to undermine Persea's restrictions on ID space. We now discuss these attacks and how Perseea is resilient to them.

Note that our arguments for security against these attacks are inductive: Starting from a small base of honest nodes, in which lookups will succeed, additional nodes can be added according to the rules of protocol without substantial risk that lookups will be subverted as long as the ratio of attack edges to nodes remains moderate (i.e. $g/n \leq 1.0$) at all times. This assumption would only be violated if the early adopters were particularly vulnerable to social engineering.

Node insertion. In a node insertion attack, the attacker responds to a **lookup**(key) request by pretending that there is a malicious node with an ID that matches very closely to the key currently being searched for and returning this node's information. The attacker creates a certificate with this ID and signs it with a plausible key for a parent.

The certificate checks in Persea will cause this attack to fail. In particular, the node that initiated the **lookup**(key) request will also perform a lookup for the certificate of the returned node. The attacker will not be able to store the certificate in the DHT, as the replicas for the given ID will look up the parent's certificate. Note that the parent's certificate similarly cannot be stored in the DHT by the attacker. Additionally, the node ID must fit the fixed values determined by Z, b, and c_f (see §4.1). Each node can very simply calculate whether a returned node ID fits these system-wide values. Thus, the attacker's certificate will fail to validate and the user can use the other returned lookup results.

Node ID hijacking. In this attack, the malicious peer M falsely claims to have the node ID of an existing honest peer H and sends a request to a victim node V to insert this node ID into its k-bucket. If effective, this attack would allow widespread poisoning of k-buckets.

Persea is also resilient against this attack. A node cannot falsely claim a node ID in Persea because of the certification mechanism (§4.2). When node P requests node Q to add it to Q's k-bucket, node P has to show its certificate, which shows the actual node ID.

5.2 Other Attacks on DHTs

P2P systems, and DHTs in particular, are subject to a wide range of attacks. As most of these are orthogonal to our approach, we limit our discussion to the most salient issues.

Persea, being based on Kad, is subject to the *eclipse attack*, in which routing tables are filled the attacker [26]. Any P2P system must address this attack, but it is orthogonal to our design and can be solved using existing techniques, such as the approach of Singh et al. [24].

Index poisoning attacks, in which the attacker adds a large volume of invalid information into the DHT [15], are no more problematic in Persea than other DHTs, and can also be addressed with existing approaches [16].

Denial of service (DoS) is another possible attack on DHTs. DoS has minimal additional impact on our system beyond the impact on other DHTs. Notably, DoS on bootstrap nodes does not prevent nodes from joining, except through the targeted nodes, and does not stop nodes from verifying any certificates, as these are stored in a replicated manner in the DHT.

6. SIMULATION AND RESULTS

To evaluate Persea, we built a custom simulation of the protocol, including building the bootstrap tree and filling the ID space, adding malicious peers via attack edges, and performing lookups over the modified Kad overlay. In this section, we describe the design of our simulation and present the results of our experiments.

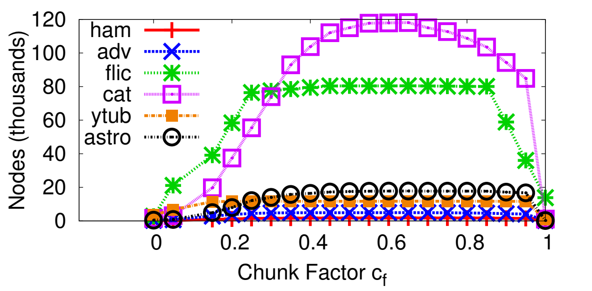


Figure 3: Effect of parameter c_f on the number of nodes in the system

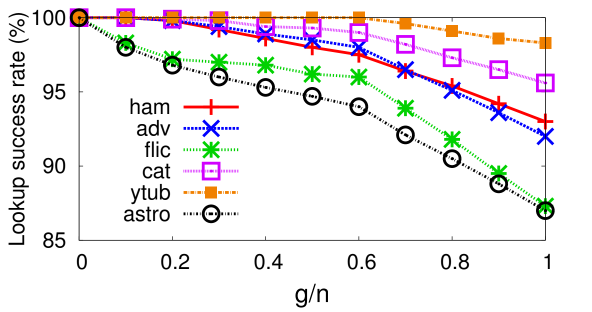


Figure 4: Lookup success rate for varying attack ratio g/n]

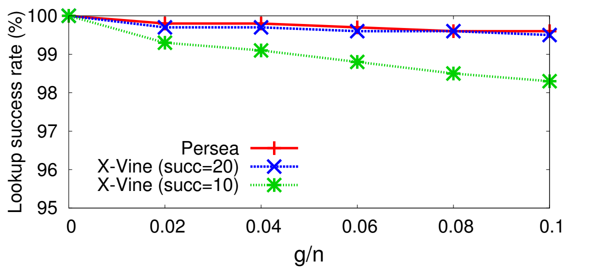


Figure 5: Comparison with X-Vine for $R = 5$

In all of our experiments, we simulate for a 31-bit ID space. Unless otherwise specified, we use default system values of chunk-factor $c_f = 0.65$, redundancy $R = 7$, and Kad parameters $\alpha = 5$, $\beta = 7$, and bucket size $k = 7$. Although the redundancy parameters (R, α, and β) may seem high, we show in Section 6.2.4 that the overheads for lookups are quite reasonable.

6.1 Building the Network and Joining of Attackers

To build the bootstrap tree, we emulate the process of nodes joining via existing connections in a social network graph. Although Persea does not rely on the structure of the social graph for its security properties, we use real social network graphs to provide a realistic basis for the choices that nodes make in building the tree.

To show that Persea will work for a variety of underlying social networks, we evaluate over six different social networks[2] with a range of mixing times. We evaluate Persea in simulations for five social network datasets: advogato (**adv**), hamsterster (**ham**), youtube (**ytub**), flickr (**flic**), and catster (**cat**), and one collaboration network: ca-AstroPh (**astro**). **ham**, **flic**, **ytub**, and **cat** are social networks drawn from users on the Hamsterster.com, Flickr.com, Youtube.com, and Catster.com Websites, respectively. Table 2 shows the sizes and mixing times of the network datasets. We measure the mixing times of these networks by using the code and procedures from Mohaisen et al. [20]; details are in the Appendix. Larger values show slower mixing times.

As we construct the initial boostrap graph, the nodes in these datasets are considered to be honest. We build our system starting with seven bootstrap nodes. In deployment, bootstrap nodes would be the users who take initiative to build the system. In our experiments, we choose seven highly connected nodes from the social network to start building the network. We then use breadth-first-search over the social graph to add other nodes. A link between node P and node Q in the dataset is interpreted as an invitation from node P to node Q. Thus, P becomes Q's parent in the boostrap graph. Also, P and Q add each other to their k-buckets.

After adding all of the honest nodes, we add Sybil nodes by creating attack edges to randomly selected honest peers. An attack edge represents an invitation from the honest node, providing the attacker with a certified ID and chunk of ID space through which it could invite more Sybil nodes. An attack edge can be created with a benign peer from any level of the hierarhcial ID space.

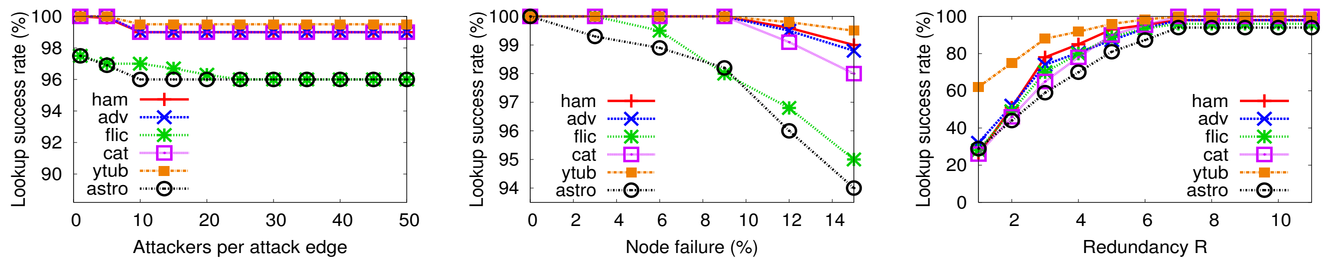
One may think that the attacker is at a disadvantage by being added after honest nodes build a bootstrap tree. This is not the case. The attackers have an equal chance to get attack edges at all levels of the tree, and there are always chunks to be given out at the highest levels of the tree. To demonstrate this, we examined the ratio of attack edges to honest nodes (g/n) in our simulations for all levels of the ID space (see Table 3 for the results for $g/n = 1.0$ overall). The ratios are roughly equal across the levels. We note that the ratios are actually greater than the overall ratio at the top level for the two networks (**cat** and **ytub**) for which Persea has the best lookup performance.

Maximizing the number of nodes. As the bootstrap graph is built, Persea should allow any new node to join through any existing node who is a real social connection. This means that both the number of IDs in each node's chunk and the depth of the tree should be large to prevent significant limitations on legitimate invitations. These two numbers are balanced by the chunk factor c_f—a larger c_f means larger chunks for each node, while a smaller c_f means a larger maximum tree depth. In Figure 3, we show this trade-off for our ID space and our social network graphs in terms of the maximum number of nodes in the system. Based on these results, we use $c_f = 0.65$ in our simulations to maximize the size of the experiment.

Table 3: g/n in each level of hierarchical ID space [overall $g/n = 1.0$]

Level	**ham**	**adv**	**flic**	**cat**	**ytub**	**astro**
1	0.91	0.95	1.17	1.10	1.15	0.95
2	0.96	1.10	0.96	0.97	0.93	0.98
3	1.01	1.03	0.97	0.97	0.97	0.99
4	0.98	1.04	0.95	0.97	0.95	1.03
5	1.00	0.93	0.97	0.97	1.00	1.03
6	1.16	0.95	0.98	0.98	–	1.04
7	0.94	–	–	0.98	–	1.05
8	1.04	–	–	1.11	–	0.93

[2] http://snap.stanford.edu/data, http://konect.uni-koblenz.de, http://socialcomputing.asu.edu/pages/datasets



(a) Varying attackers per attack edge ($g/n = 0.10$)

(b) Varying percentage of node failure

(c) Varying R ($g/n = 0.50$)

Figure 6: Lookup success rates in Persea. Note the different y-axis ranges.

6.2 Results

In this section, we justify our claims through the results of simulation. We also analyze the effects of changing parameters and report the results for overhead.

6.2.1 Justification of claims

In this paper, we make the following claims about Persea:

- **Claim I:** Persea provides a high lookup success rate, even when the ratio of attack edges to honest nodes is not small.
- **Claim II:** Persea works well for both fast- and slow-mixing social networks.
- **Claim III:** The hierarchical node ID distribution confines the attackers in considerably smaller regions of ID space.
- **Claim IV:** The lookup success rate in Persea is not significantly affected by varying the number of attackers per attack edge.
- **Claim V:** Even when a significant fraction of nodes fail simultaneously, lookups should still succeed.

Claims I and III-V all speak to *Design Goal #1* of offering Sybil-resistent lookups. Claim II concerns *Design Goal #2* of not depending on the structure of the underlying social network. We justify our claims through our simulation experiments.

Claim I. In Persea, we get a 100% lookup success rate for up to $g/n = 0.45$ in a network of $149,700$ honest nodes (Figure 4). When $g/n = 1.0$ in the largest network in our evaluation, Persea's lookup success rate is 95.6%. In comparison, X-Vine only shows results for g/n up to 0.1, where the success rate is already below 100% [19]. We directly compare Persea with X-Vine in Figure 5, which is limited to at most $g/n = 0.07$ using results from Mittal et al. [19]. See the Appendix for details. Persea performs similarly to X-Vine using 20 successors per node. Whānau has significantly lower success rates for $g/n \geq 0.15$ [11] and can be expanded to improve success rates at the cost of greatly increased overheads [12].

Persea's DHT is based on Kad, which originally did not have any protections against the Sybil attack. Cholez et al. report that recent versions of Kad clients (aMule and eMule) attempt to protect against Sybil attacks by restricting IP addresses from appearing more than once in a node's k-buckets [6]. This does not affect our attacker, who we assume to have a large number of IP addresses, e.g. by running a botnet. We evaluated Kad in our experiments as well. For $g/n = 0.1$, the lookup success rate in Kad is 30% in **cat** and 40% in **flic**, both much worse than Persea.

Claim II. For these experiments, we select six networks with mixing times varying from 2.3 steps (fast-mixing) to 11.5 steps (slow-mixing) (Table 2). Using the lookup results when $g/n = 1.0$, we do find an inverse correlation between mixing time and lookup success ($r(4) = 0.60, p = 0.1$). Though it is not statistically significant, we do note that Persea performed the worst on two of the slowest mixing graphs (**flic** and **astro**). It is possible that faster mixing leads to better distribution of ID space among honest nodes, though we have not investigated this in detail. Nevertheless, the lookup results remain reasonable for these social networks. When $g/n = 0.5$, Persea has a success rate of 96.2% on **flic**, the slowest mixing graph, and 94.7% on **astro**.

In tables showing lookup overhead (Table 5, 6, and 7) and the amount of attacker-controlled ID space (Table 4), we show the mixing time in parentheses after the network name. Slower mixing networks also have higher overheads, though all overheads are quite reasonable for all six networks. The amount of attacker-controlled ID space does not seem to be correlated with mixing time.

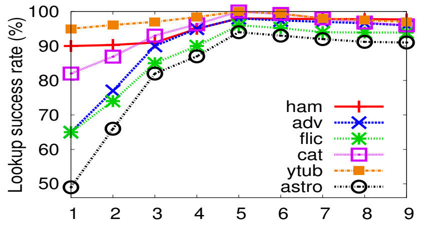
Claim III. In Persea, the hierarchical node ID distribution limits the attacker nodes to a small fraction of the ID space, even for a significant number of attack edges. Table 4 shows the fraction of ID space controlled by the attacker. Even for $g/n = 1.0$, no more than 0.9% of the ID space is controlled by attacker nodes.

Claim IV. When an attacker joins the system, it can invite more malicious nodes. However, the lookup success rate in Persea should not be significantly affected by increasing the number of attackers per attack edge, since the hierarchical node ID distribution limits the attackers to isolated regions in the ID space. To test this, we increase the number of attackers per attack edge from one to fifty, which makes the number of attackers become 60% of total nodes in the network. We keep $g/n = 0.10$ fixed. For example, the lookup success rate is 99.5% in **ytub** and 99% in **ham** and **cat** when the number of attackers per attack edge is 50 (Figure 6(a)). We see no drops in performance in any network when the number of attackers per attack edges grows above 25.

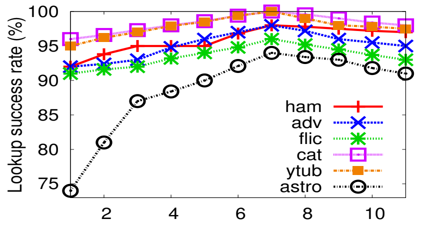
Claim V. We evaluate the performance of Persea under node failure. We are interested in the static resilience of our system, i.e., the lookup success rate after a percentage of nodes in the system fail simultaneously. This is a worst-case version of a system with churn

Table 4: Fraction of ID space occupied by the attackers for varying g/n

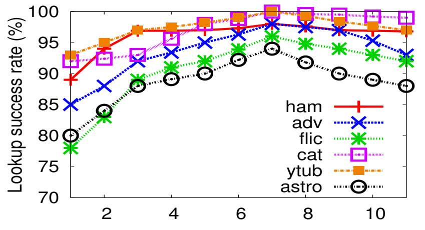
g/n	0.10	0.50	0.80	1.0
adv (2.3)	0.0004	0.001	0.002	0.003
ham (3.0)	0.0005	0.0006	0.001	0.002
ytub (3.0)	0.001	0.004	0.006	0.007
astro (8.0)	0.0003	0.001	0.002	0.003
flic (11.5)	0.001	0.005	0.007	0.009
cat (8.0)	0.0007	0.003	0.006	0.007



(a) Nodes queried α



(b) Nodes returned β



(c) k-bucket size

Figure 7: Lookup success rate for varying Kad-based system parameters [$g/n = 0.50$]

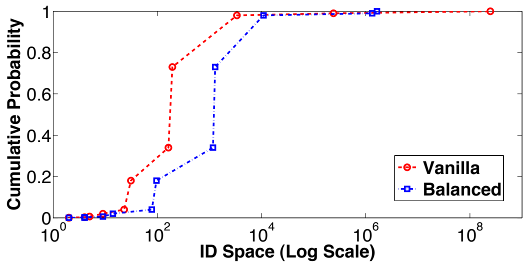


Figure 8: Empirical CDF of ID ownership in the catster network

in which the nodes fail and routing updates have not yet propagated to any peers. Figure 6(b) depicts the lookup success rate as a function of the percentage of nodes that fail simultaneously, averaged over $10,000$ lookups. The results show that the lookup success rate remains 100% in the largest network of our evaluation, even when 10% of nodes fail. We do not add any malicious peers during this experiment. In Figure 9 (in the Appendix), we present the results of an experiment in which we vary the percentage of node failure and have attackers, with $g/n = 0.1$.

6.2.2 *Varying system parameters*

We now examine the effect of changing the key system parameters. When we increase redundancy R, nodes will store each (key, value) pair in more nodes and consequently lookup the key in more nodes. Thus, a larger value of R will result in better lookup performance at the cost of greater overheads. In Figure 6(c), we see that success rates appear to level off at $R = 7$. When β is increased, more nodes are returned to the source by each of α nodes in an iteration. We get the maximum lookup success rates for $\alpha = 5$ (Figure 7(a)) and $\beta = 7$ (Figure 7(b)). For a large number of attackers, higher values of α and β may increase the probability of selecting more malicious nodes in each iteration. We also examine settings for the bucket size k. We get seven as the optimal value for k with these settings (Figure 7(c)). The lookup success rate may decrease for higher values of k, since it means that routing tables can hold more attackers.

Table 5: Lookup Overhead [$g/n = 1.0$]

Network	Msgs (R lookups)	hop count (1 lookup)
adv (2.3)	21.65	3.09
ham (3.0)	18.87	2.69
ytub (3.0)	19.27	2.75
astro (8.0)	26.63	3.80
flic (11.5)	25.90	3.70
cat (8.0)	24.36	3.48

6.2.3 *ID ownership distribution*

We now examine the distribution of IDs across nodes. Figure 8 shows the distribution of IDs when using **cat**, the largest network on which we tested. In *vanilla Persea*, with in-order ID distribution, some nodes own 11.4% of the ID space. With our balanced approach, however, no node owns more than 0.077% of the ID space. The balanced approach provides much more even distribution of IDs to the peers.

6.2.4 *Overheads*

We now discuss how our system meets *Design Goal #4* of having moderate overheads. One overhead in DHTs is maintaining routing tables due to churn, e.g. by using *heartbeat messages*. Mittal et al. report that Whānau has an average routing table size of 20,000 in a network of 100,000 nodes, leading to very high maintenance costs [19]. In our experiments, however, Persea systems had between 10.1 and 68.1 routing table entries per node for different networks. This suggests moderate maintenance costs and is in line with other DHTs, including X-Vine.

Another overhead is in lookup costs, as measured by the *number of messages* and the *hop count*, the number of nodes required to reach a lookup target. In Persea, the certificate of a peer is stored in R nodes in the DHT, and the source retrieves them for validation by using the lookup mechanism (§4.4). Thus, the lookup overhead shown in Table 5 also includes the overhead for the retrieval of the certificates of a node. The number of messages is quite reasonable at less than 27 in all cases. As for hop count, Table 5 shows that the average hop count in Persea is low, e.g., 3.48 for $g/n = 1.0$ in the 149,700-node **cat** network. In contrast, for topologies with $100,000$ nodes, X-Vine requires 10-15 hops for routing [19]. Whānau has a fixed hop count of two.

Increasing the rate of node failure does not greatly increase the average hop count and messages during lookup in Persea. Results (shown in Table 6 in the Appendix) show that for 15% node failure in the largest network in our evaluation, the average hop count per lookup increases by only 0.31. Finally, we find that increasing the number of attackers per attack edge has a small impact on the lookup overhead in Persea, as shown in Table 7 (in the Appendix).

7. CONCLUSION

Persea leverages the social relationships among honest peers to build a Sybil-resistant DHT that does not depend on the assumptions of a fast-mixing social network and a small number of attack edges. Persea also provides resilience against attacks (denial of service, node ID hijacking, and node insertion) launched by attacker nodes.

We see Persea as a first step in exploring mechanisms for using one's social network as an identifier and leveraging that for security. It is well known that social network information is strongly identifying [21], but it is non-obvious as to how we can convert this information into a usable identifier. The bootstrap tree is one approach to this, and the effectiveness of Persea suggests to us that this broader idea has significant potential and should be further explored.

8. ACKNOWLEDGMENTS

We thank our shepherd Aziz Mohaisen and the anonymous reviewers for their detailed comments and feedback. We also thank Aziz Mohaisen for providing code and assistance for measuring mixing times of social networks. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1117866 and CAREER award number CNS-0954133.

9. REFERENCES

[1] L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi. SoK: The evolution of Sybil defense via social networks. In *IEEE S&P*, 2013.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). *ACM SIGCOMM Computer Communication Review*, 38(4):339–350, 2008.

[3] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: Automated identity theft attacks on social networks. In *WWW*, 2009.

[4] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: when bots socialize for fame and money. In *ACSAC*, 2011.

[5] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. Design and analysis of a social botnet. *Computer Networks*, 57(2):556–578, 2013.

[6] T. Cholez, I. Chrisment, and O. Festor. Evaluation of Sybil attacks protection schemes in KAD. In *AIMS*, 2009.

[7] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson. Sybil-resistant DHT routing. In *ESORICS*, 2005.

[8] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil nodes using social networks. In *NDSS*, 2009.

[9] J. R. Douceur. The Sybil attack. In *IPTPS*, 2002.

[10] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu. Reverse social engineering attacks in online social networks. In *DIMVA*, 2011.

[11] C. Lesniewski-Laas. A Sybil-proof one-hop DHT. In *Workshop on Social Network Systems*, 2008.

[12] C. Lesniewski-Laas and M. F. Kaashoek. Whānau: A Sybil-proof distributed hash table. In *USENIX NSDI*, 2010.

[13] B. Levine, C. Shields, and N. Margolin. A survey of solutions to the Sybil attack. Technical Report 2006-052, University of Massachusetts Amherst, 2006.

[14] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li. An empirical study of collusion behavior in the Maze P2P file-sharing system. In *IEEE ICDCS*, 2007.

[15] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in P2P file sharing systems. In *INFOCOM*, pages 1–12, 2006.

[16] X. Lou and K. Hwang. Prevention of index-poisoning DDoS attacks in peer-to-peer file-sharing networks. Technical Report TR-2006-5, USC Internet and Grid Computing Lab, November 2006.

[17] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information sytem based on the XOR metric. In *IPTPS*, 2002.

[18] D. Mazieres, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. *ACM SIGOPS Operating Systems Review*, 33(5):124–139, 1999.

[19] P. Mittal, M. Caesar, and N. Borisov. X-Vine: Secure and pseudonymous routing in DHTs using social networks. In *NDSS*, 2012.

[20] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. In *ACM IMC*, 2010.

[21] A. Narayanan and V. Shmatikov. De-anonymizing social networks. In *IEEE S&P*, 2009.

[22] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[24] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *ACM SIGOPS European Workshop*, page 21. ACM, 2004.

[25] M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *USENIX NSDI*, 2012.

[26] M. Steiner, T. En-Najjary, and E. W. Biersack. Exploiting kad: possible uses and misuses. *ACM SIGCOMM Computer Communication Review*, 37(5):65–70, 2007.

[27] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[28] N. Tran, J. Li, L. Subramanian, and S. S. Chow. Optimal Sybil-resilient node admission control. In *IEEE INFOCOM*, 2011.

[29] B. Viswanath, M. Mondal, A. Clement, P. Druschel, K. P. Gummadi, A. Mislove, and A. Post. Exploring the design space of social network-based Sybil defenses. In *COMSNETS*, 2012.

[30] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based Sybil defenses. In *ACM SIGCOMM*, 2010.

[31] W. Wei, F. Xu, C. C. Tan, and Q. Li. SybilDefender: Defend against Sybil attacks in large social networks. In *IEEE INFOCOM*, 2012.

[32] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Eurosys*, 2009.

[33] S. Wolchoky, O. S. Hofmanny, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. In *NDSS*, 2010.

[34] M. Yang, Z. Zhang, X. Li, and Y. Dai. An empirical study of free-riding behavior in the Maze P2P file-sharing system. In *IPTPS*, 2005.

[35] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai. Uncovering social network Sybils in the wild. In *ACM IMC*, 2011.

[36] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. In *IEEE S&P*, 2008.

[37] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending against Sybil attacks via social networks. In *ACM SIGCOMM*, 2006.

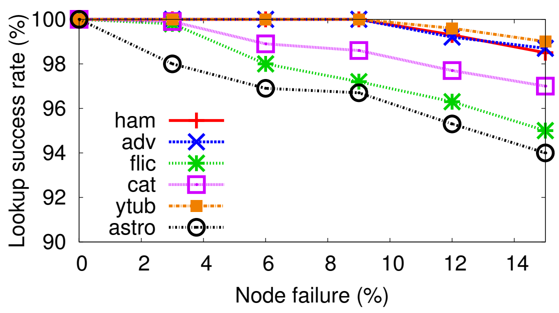


Figure 9: Lookup success rate for varying percentage of node failure [$g/n = 0.10$]

APPENDIX

A. ADDITIONAL RESULTS

Measuring Mixing time. Mixing time represents the number of steps required by a random walk to approach the uniform distribution [20]. We measure the mixing time of networks used in our simulation by using the codes and methodology provided by the authors of [20], where the walk length at total variation distance 0.1 represents the lower bound of the mixing time of a network. To measure the mixing time for large networks (e.g., **flic** and **cat**), we adopt the sampling technique described in [20] to get $25,000$ nodes for each network. For other networks, we use the largest connected component to measure mixing time. Our results (see Table 2) show that **flic**, **cat**, and **astro** are relatively slow-mixing compared to **ham**, **adv**, and **ytub** networks.

Results for node failure ($g/n = 0.1$**).** We evaluate for varying percentage of node-failure having $g/n = 0.1$, where n represents the number of total benign peers (alive+failed). For this experiment, we consider that only benign peers fail in the system. Figure 9 illustrates the results. We find that 99% lookups still succeed in Persea, when 10% fail in a network of $149,700$ nodes.

Results for the distribution of g/n**.** For $g/n = 1.0$ in the whole network, Table 3 shows the distribution of this ratio over different levels of hierarchical ID space. Here, **flic** has six levels and other networks have eight levels in their hierarhcial ID space (see §6 for a detailed explanation).

Additional overhead results. Table 6 compares the amount of overhead for a Persea system with no failures and one with 15% node failures, which should be considered high. We show both total messages and hop count. The number of messages increases in the high failure rate scenario by an average of 2.8 for $R = 7$ redundant lookups across our social networks. The hop count increases by an average of 0.41.

Table 7 compares the amount of overhead for a Persea system with one attacker per attack edge and 10 attackers per attack edge. When averaged across our social networks, the total number of messages increases by 1.4 for $R = 7$ redundant lookups and the hop count increases by 0.2.

Comparison with X-Vine. We compare Persea and X-Vine in terms of lookup success rate and overhead, where the results for X-Vine are taken from [19] and then we perform simulation on Persea for the same dataset (New Orleans Facebook Friendship Graph) and parameter ($R = 5$). The number of nodes and edges in this interaction graph are 63731 and 1545686, respectively. However, in X-Vine [19] the number of nodes and edges are reduced to 50150 and 661850, respectively through pre-processing. So, while comparing Persea and X-Vine (see Figure 5), for each value of g/n the number of attack edges are different for the two systems. In our comparison, for X-Vine we consider two values (10 and 20) as the number of successors.

The results for lookup success rate are shown in Figure 5 (see §6.2), where we find that Persea performs very similarly to X-Vine for the given parameters. The mean lookup path length in Persea is found less than that in X-Vine, where for $R = 5$ the mean lookup path lengths for X-Vine are 13.7 (succ=10) and 10.7 (succ=20), and for Persea the mean lookup path length is 4.4.

Table 6: Overhead for Node Failure

Overhead	Avg. Messages in R Lookups		Avg. Hop-Count in each of R Lookups	
Failure (%)	No Failure	15%	No Failure	15%
adv (2.3)	18.61	20.78	2.65	2.96
ham (3.0)	15.32	20.83	2.19	2.98
ytub (3.0)	17.43	18.25	2.49	2.61
cat (8.0)	22.54	24.71	3.22	3.53
astro (8.0)	23.65	26.55	3.37	3.79
flic (11.5)	23.2	25.04	3.31	3.57

Table 7: Overhead for Varying Attackers per Attack Edge [$g/n = 0.10$]

Overhead	Avg. Messages in R Lookups		Avg. Hop-Count in each of R Lookups	
Attackers per attack edge	1	10	1	10
adv (2.3)	18.72	19.95	2.67	2.85
ham (3.0)	15.41	17.04	2.20	2.43
ytub (3.0)	18.25	18.40	2.60	2.63
astro (8.0)	23.73	26.02	3.39	3.71
flic (11.5)	23.3	25.29	3.32	3.61
cat (8.0)	22.66	23.87	3.23	3.41