Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2009-045 September 24, 2009

Whanaungatanga: Sybil-proof routing with social networks

Chris Lesniewski-Laas and M. Frans Kaashoek

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Whanaungatanga: Sybil-proof routing with social networks

Chris Lesniewski-Laas M. Frans Kaashoek

ABSTRACT

Decentralized systems, such as distributed hash tables, are subject to the Sybil attack, in which an adversary creates many false identities to increase its influence. This paper proposes a routing protocol for a distributed hash table that is strongly resistant to the Sybil attack. This is the first solution to this problem with sublinear run time and space usage.

The protocol uses the social connections between users to build routing tables that enable Sybil-resistant distributed hash table lookups. With a social network of n well-connected honest nodes, the protocol can tolerate up to $O(n/\log n)$ "attack edges" (social links from honest users to phoney identities). This means that an adversary has to fool a large fraction of the honest users before any lookups will fail.

The protocol builds routing tables that contain $O(\sqrt{n}\log^{3/2} n)$ entries per node. Lookups take $O(1)$ time. Simulation results, using social network graphs from LiveJournal, Flickr, and YouTube confirm the analytical results.

1 Introduction

Decentralized systems on the Internet are vulnerable to the "Sybil attack", in which an adversary creates numerous false identities to influence the system's behavior [7]. This problem is particularly pernicious when the system is responsible for routing messages amongst nodes, as in the Distributed Hash Tables (DHT) [19] which underlie many peer-to-peer systems, because an adversary can prevent honest nodes from communicating altogether [18].

If a central authority certifies identities as genuine, then standard replication techniques can be used to fortify these protocols [2, 15]. However, the cost of universal strong identities may be prohibitive. Instead, recent work [22, 21, 6, 13, 11, 3] proposes using the weak identity information inherent in a social network to produce a completely decentralized system. This paper resolves an open problem by demonstrating an efficient, structured DHT which enables honest nodes to reliably communicate despite a concerted Sybil attack.

Consider a set of honest people (nodes) who are connected by a network of individual trust relations formed through collaborations and introductions. These social links are assumed to be reflexive (undirected), and each node keeps track of his immediate neighbors, but the set of people have no official leader — there is no central trusted node. Assume that the social network is well-connected (Sections 4.3, 7.1).

An adversary can infiltrate the network by creating many *Sybil nodes* (phoney identities) and gaining the trust of honest people. Nodes cannot directly distinguish Sybil identities from genuine ones (if they could, it would be simple to reject Sybils). However, we assume that most honest nodes have more social connections to other honest nodes than to Sybils; in other words, the network has a *sparse cut* between the honest nodes and the Sybil nodes.

We assume that the adversary cannot prevent immediate friends from communicating, but can try to disrupt the network by spreading misinformation. Consider an honest node u that wants to look up the key x and will recognize the corresponding value (e.g., a signed data block, or the current IP address of another node). In a typical structured DHT, u queries another node which u believes to be "closer" to x, which forwards u to another even-closer node, and so on until x is found. The adversary can disrupt this process by spreading false information (e.g., that its nodes are close to a particular key) and then intercepting honest nodes' routing queries. Unstructured protocols that work by flooding or gossip are more robust against these attacks, but pay a heavy performance price, requiring linear time to find a key.

This paper's main contribution is Wh$\bar{a}$naungatanga,[1] a novel protocol that is the first solution to Sybil-resistant routing that has a sublinear run time and space usage. Wh$\bar{a}$naungatanga builds on recent results on fast-mixing social networks; it constructs routing tables by taking short random walks on the social network. The second contribution is a detailed theoretical analysis which shows that the routing tables contain $O(\sqrt{n}\log^{3/2} n)$ entries per node. Using these routing tables, lookups take $O(1)$ time, like previous (insecure) one-hop DHTs. The third contribution is an evaluation of Wh$\bar{a}$naungatanga using existing social networks. The evaluation shows that social network graphs from LiveJournal, Flickr, and YouTube are sufficiently well-connected

[1] *Wh$\bar{a}$naungatanga* is a M$\bar{a}$ori word which refers to the collective support network of mutual obligations associated with kinship and community relationships. The root word *wh$\bar{a}$nau* is cognate with the Hawai'ian word *'ohana*.

that Whānaungatanga works well. Simulations using these graphs confirm the theoretical analysis.

Section 2 reviews some of the related work. Section 3 informally states our goals. Section 4 explains what properties of the social network we use to design an efficient protocol. Section 5 presents a simple unstructured protocol that is clearly Sybil-proof, but inefficient. Section 6 presents the structured Whānaungatanga protocol, which is both Sybil-proof and efficient. Section 7 proves Whānaungatanga's correctness, and Section 8 confirms its theoretical properties by simulations on social network graphs from popular Internet services. Section 9 briefly outlines how to extend the protocol to dynamic social networks. Section 10 summarizes.

2 Related work

Shortly after the introduction of scalable peer-to-peer systems based on DHTs, the Sybil attack was recognized as a serious security challenge [7, 10, 18, 17]. A number of techniques [2,15,17] have been proposed to make DHTs resistant to a small fraction of Sybil nodes, but all such systems ultimately rely on a certifying authority to perform admission control and limit the number of Sybil identities [7, 16, 1].

Several researchers [11,13,6,3] proposed using social network information to fortify peer-to-peer systems against the Sybil attack. The *bootstrap graph* model [6] introduced a correctness criterion for secure routing using a social network and presented preliminary progress towards that goal, but left a robust and efficient protocol as an open problem.

Recently, the SybilGuard and SybilLimit systems [22, 21] have shown how to use a *fast mixing* social network (see Section 4.3) as a defense against the Sybil attack in general decentralized systems. Using SybilLimit, an honest node can certify other players as "probably honest", accepting no more than $O(\log n)$ dishonest Sybil identities per attack edge. (Each certification costs $O(\sqrt{n})$ bandwidth.) For example, SybilLimit's vetting procedure can be used to check that at least one of a set of storage replicas is likely to be honest.

Applying SybilLimit naïvely to the problem of Sybil-proof DHT routing yields a protocol which uses $O(n^2\sqrt{n})$ bandwidth. Unfortunately, this is more costly than even a simple flooding protocol (see Section 4.1). However, this paper shows how the underlying technique developed for SybilLimit — short random walks on a fast-mixing social network — can be adapted to the Sybil-proof routing problem.

A few papers have adapted the same underlying idea for purposes other than routing. Nguyen *et al.* use it for Sybil-resilient content ranking [20], and Danezis and Mittal use it for Bayesian inference of Sybils [5].

3 Goals

3.1 The setting: informal security definition

At each node, the application provides the DHT with a set of key-value records to store. The aim of the DHT routing protocol is to construct a distributed data structure enabling secure and efficient lookup from any key in the system to its corresponding value.

A DHT's implementation is a pair of procedures SETUP and LOOKUP. SETUP() cooperatively transforms the nodes' local parameters (e.g. key-value records, social connections) into a set of routing table structures stored at each node. (This paper distils the algorithmic content from the the details of inter-node communication by presenting SETUP as if it operated on the state of all nodes at once.) After all nodes complete the SETUP phase, any node s can call LOOKUP(s, key_t) to use these routing tables to find and return the target $value_t$.

We allow adversaries to deviate from the protocol in a Byzantine way: they may make up arbitrary responses to queries from honest nodes, but may not forge messages from honest nodes. The application might enforce this by authenticating messages using public keys, where each node knows its social neighbors' public keys (through, e.g., a physical rendezvous). Adversaries may create any number of pseudonyms (Sybils) which are indistinguishable from honest nodes.

We consider a DHT "Sybil-proof" if LOOKUP has a high probability of returning the correct value, despite arbitrary attacks by the adversary during both the SETUP and LOOKUP phases. Note that we can always amplify the probability of success exponentially by running multiple protocol instances in parallel.

The adversary can always join the DHT normally and insert an arbitrary key-value pair, including a different value for a key already in the DHT. Thus, a Sybil-proof DHT provides availability, but not integrity: LOOKUP must at least find all values inserted by honest nodes for the specified key, but may also return some values inserted by the adversary. Many applications will have some application-specific way to discard bogus key-value pairs. For example, if the key is a content-hash of the value (as in many block storage DHTs),then any node can easily check whether they match.

3.2 Performance goals

In addition to security, we are also concerned about the resource consumption of a DHT protocol. Section 7 examines performance in detail, but informally, our goals are:

- The routing tables should be constructible by an efficient distributed protocol.
- The table size per node should be reasonably bounded. If $n = 5 \times 10^8$, the approximate number of Internet hosts in 2009, or $n = 3 \times 10^9$, the approximate number of mobile phones in the world, then it may be impractical to download and store $O(n)$ table entries to each node. However, $O(\sqrt{n})$ entries may be acceptable.
- The run time of LOOKUP, and the number of messages sent, should be reasonably small — ideally $O(1)$.
- The storage and bandwidth consumption should not be strongly influenced by the adversary's behavior.
- An interactive adversary may force an honest node s to look up a bogus key, i.e. call LOOKUP$(s, \hat{k})$ on some arbitrary key $\hat{k}$. Returning a negative result should not consume too much of the honest nodes' resources.

As a matter of policy and fairness, we believe that a node's table size and bandwidth consumption should be proportional

to the node's degree — that is, highly connected nodes should do more work than casual participants. While it would be straightforward to adapt our protocol to a different policy, this paper does not discuss the topic any further.

4 Approach: use a social network

Any Sybil-proof protocol must make use of some externally-provided information [7]; we propose that each node be given a list of its neighbors in the social network. This section gives a simple example to illustrate how these social connections can be useful, and then informally describes the characteristics of social networks important for Wh$\bar{a}$naungatanga.

4.1 Strawman protocol: social flooding

An extremely simple protocol demonstrates that a social network can be used to thwart the Sybil attack [6]. The SETUP phase does nothing, and LOOKUP simply floods queries over all links of the social network:

LOOKUP(u, key_t)

1 **if** $key_t = \text{key}(u)$ ▷ *The locally-stored record*
2 **then return** $\text{value}(u)$ ▷ *at u is $\big(\text{key}(u), \text{value}(u)\big)$.*
3 **for** $v \in \text{neighbors}(u)$ ▷ *Send message to all neighbors.*
4 **do** try LOOKUP(v, key_t) ▷ *Don't wait for reply.*

The above algorithm is secure. The adversary's nodes might refuse to forward queries, or they might reply with bogus values. However, if there exists any path of honest nodes between the source node and the target key, then the adversary cannot prevent each of these nodes from forwarding the query to the next. In this way, the query will always reach the target node, which will reply with the correct value.

Clearly, this illustrative LOOKUP routine has many deficiencies as a practical implementation; for example, the query messages will always continue to propagate even after a result is found. Some of the deficiencies are easily patched using standard techniques, but the flooding approach will always be inefficient: a large fraction of the participating nodes are contacted for every lookup. This paper's goal is to reduce resource consumption while keeping the system's security against the Sybil attack. Reducing the number of messages per lookup — to sublinear, and then constant, in the number of nodes — requires progressively more complex protocols.

4.2 The adversary's attack edges

Figure 1 conceptually divides the social network into two parts, an **honest region** containing all honest nodes and a **Sybil region** containing all Sybil identities. An **attack edge** is a connection between a Sybil node and an honest node. An **honest edge** is a connection between two honest nodes [22]. (An "honest" node whose software's integrity has been compromised by the adversary is considered a Sybil node.)

Our key assumption is that the number of attack edges, g, is small relative to the number of honest nodes, n. We justify this assumption by observing that, unlike creating a Sybil identity, creating an attack edge requires the adversary to expend social-engineering effort: he must convince an honest person to create a social link to one of his Sybil identities.

Crucially, our protocol's behavior does not depend *at all* on the number of Sybil identities, or on the structure of the Sybil region. Therefore, the classic Sybil attack, of creating many identities to swamp the honest identities, is ineffective.

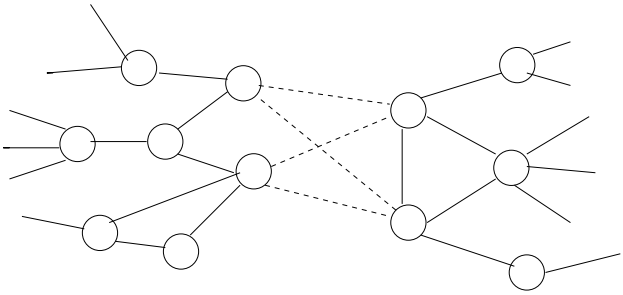


Figure 1: Social network. A sparse cut (the dashed attack edges) separates the honest nodes from the Sybil nodes.

4.3 Fast-mixing social networks

We can restate the above assumption, that $g \ll n$, as "there is a *sparse cut* between the honest region and the Sybil region." To make any use of this assumption, we naturally need to make one more assumption: that there is *no* sparse cut within the honest region. In other words, the honest region must be an *expander graph*.

Expander graphs are **fast mixing**, which means that a short random walk starting from any node will quickly approach the stationary distribution [4]. Roughly speaking, every honest edge is equally likely to be the last edge crossed by the random walk. The **mixing time**, w, is the number of steps a random walk must take to reach this nearly-uniform distribution. For a fast mixing network, $w = O(\log n)$.

Section 8.1 shows that real social graphs appear to be fast mixing, except for a tiny fraction of isolated nodes. This matches our intuition that social networks are highly connected.

4.4 Sampling by random walk

A fast-mixing social network permits us to use random walks as a powerful tool to build Sybil-resistant protocols. Consider a w-step random walk starting at an honest node. If the number of attack edges is small, the random walk is likely to stay entirely within the honest region: Section 7.3 shows that the probability of crossing an attack edge is bounded by $O(gw/n)$. A random walk which doesn't cross an attack edge is entirely unaffected by any behavior of the adversary — the walk exactly follows the distribution of random walks on the honest region. Therefore, the last edge crossed follows a nearly-uniform distribution.

Based on this observation, an honest node can send out a w-step walk to sample a random node from the social network. If it sends out a large number of such walks, the resulting set will contain a large fraction of random honest nodes and a small fraction of Sybil nodes.

This random sampling subroutine is Wh$\bar{a}$naungatanga's main building block, and is the only way our protocol uses the social network. Because the initiating node cannot tell which individual samples are good and which are bad, Wh$\bar{a}$-

	Typical magnitude	Description
n	arbitrary	number of honest nodes
w	$O(\log n)$	mixing time of honest region
g	$O(n/w)$	number of attack edges
ϵ	$O(gw/n)$	fraction of loser nodes

Table 1: Model parameters

naungatanga treats all sampled nodes equally, relying only on the fact that a large fraction will be good samples.

Even if the number of attack edges is small compared to the number of honest nodes, some honest nodes may be near a heavy concentration of attack edges. Such **loser nodes** have been lax about ensuring that their social connections are real people, and random walks starting from those nodes will be much more likely to escape into the Sybil region. As a consequence, loser nodes will have to do more work per lookup than winner nodes, since the adversary can force them to waste resources. Luckily, only a small fraction of honest nodes are losers: a high concentration of attack edges in one part of the network means a low concentration elsewhere. Section 7.2 treats losers and winners in more detail.

A simple unbiased random walk tends to end preferentially at nodes with higher degree, since the final hop is a uniformly chosen edge. If each node u acts as $degree(u)$ independent virtual nodes, then good random samples will be distributed uniformly over all virtual nodes. As a bonus, this technique fulfils the policy goal (Section 3.2) of allocating both workload and trust according to each person's level of participation in the social network.

4.5 Sybil-proofness revisited

The preceding sections have (informally) defined our model parameters (see Table 1), enabling a more precise definition of a "Sybil-proof" DHT:

Definition. A DHT protocol is (g, ϵ)-**Sybil-proof** if, against an active adversary with up to g attack edges, the protocol's LOOKUP procedure succeeds with probability better than $1/2$ for at least $(1 - \epsilon)n$ honest nodes.

The probability $1/2$ above is arbitrary: as noted in Section 3.1, any non-negligible probability of success can be amplified exponentially by running multiple independent instances of the protocol in parallel. Amplifying until the failure probability is less than $O(1/n^3)$ essentially guarantees that all lookups will succeed with high probability (since there are only n^2 possible source-target node pairs).

ϵ represents the fraction of loser nodes, which is a function of the distribution of attack edges in the network. If attack edges are distributed uniformly, then ϵ may be zero; if attack edges are clustered, then a small fraction of nodes (Section 7.3) may be losers.

5 An unstructured DHT

In this section, we show that simple unstructured search using random walks on the social graph is Sybil-proof. However, we will find that it is not as efficient as we would hope. The intuition behind this protocol is straightforward: if an honest node sends out enough random walks, it will eventually hit every other honest node.

5.1 Subroutines for random sampling

The RANDOM-WALK procedure implements a random walk of length w on the social graph:

```
RANDOM-WALK(u_0)
1  for i ← 1 to w
2      do u_i ← RANDOM-CHOICE(neighbors(u_{i−1}))
3  return u_w
```

The protocol uses RANDOM-WALK repeatedly to collect large random samples of nodes or key-value records: [2]

```
SAMPLE-NODES(u, r)
1  for i ← 1 to r
2      do v_i ← RANDOM-WALK(u)
3  return {v_1, ..., v_r}
```

```
SAMPLE-RECORDS(u, r)
1  {v_1, ..., v_r} ← SAMPLE-NODES(u, r)
2  for i ← 1 to r
3      do record_i ← (key(v_i), value(v_i))
4  return {record_1, ..., record_r}
```

Recall that the sets returned by SAMPLE-NODES contain a large fraction of good samples (uniformly distributed over honest nodes) and a small fraction of bad samples (adversarially-chosen Sybil nodes), and honest nodes cannot distinguish between Sybil and honest nodes. Let $\mathbf{1/\alpha}$ be the minimum expected fraction of good samples returned by SAMPLE-NODES. Section 7.3 will specify α more precisely in terms of the model's parameters, but typically, $1 < \alpha < 4$. Note that to expect k good samples, a node must perform αk random walks; thus, in general, α acts like a (small, constant) multiplier on the amount of work an honest node must do.

5.2 Setup and lookup

The SETUP procedure simply uses SAMPLE-RECORDS to construct a database of r_u samples at each node. (r_u is a protocol configuration parameter which can be statically defined or dynamically adjusted based on the size of the network.)

In the SETUP procedure, the notation "for each node u" means that all honest nodes will run the subsequent code in parallel. Of course, Sybil nodes may act arbitrarily.

```
SETUP (key(·), value(·), neighbors(·); w, r_u)
1  for each node u
2      do database(u) ← SAMPLE-RECORDS(u, r_u)
3  return database
```

The LOOKUP procedure chooses a random intermediate node v and sends it a query message. It repeatedly queries different nodes until it finds one with key_t in its local database. [3]

[2] If RANDOM-WALK is implemented as a recursive request to a node's immediate neighbor, many such requests can be batched up into a single message. This improvement reduces the number of messages sent, but does not change the total bandwidth consumed.

[3] There's no need to throw away the random intermediate nodes v from LOOKUP; instead, it makes sense to add them to the database, saving work on future lookups.

```
LOOKUP(s, key_t)
1  repeat  v ← RANDOM-WALK(s)
2          try value_t ← QUERY(v, key_t)
3    until found valid value_t

QUERY(v, key_t)
1  if (key_t, value_t) ∈ database(v) for some value_t
2    then return value_t
3  elseif key_t = key(v)            ▷ Check whether it's the local record
4    then return value(v)
5  else return "not found"
```

LOOKUP keeps initiating fresh random walks until it finds the correct value associated with the target key. Since it will eventually query every single honest node in the system this way, LOOKUP is guaranteed to eventually succeed. Therefore, this protocol is Sybil-proof.

The parameter r_u tunes the relative workload of SETUP and LOOKUP. Observe that if we set $r_u = 0$ or 1, then this protocol is similar to existing unstructured ones such as Gnutella [14]. In this mode, we expect LOOKUP to send roughly αn queries before finding the correct target record: each walk has a $1/\alpha$ chance of returning a good sample, and each good sample has a $1/n$ chance of being the target.

On the other end of the scale, if $r_u = \alpha n$, we expect each node's database to contain the majority of the honest records in the system. Therefore, LOOKUP should find the target after querying about 2 good nodes, or sending approximately 2α messages in total. Compared with the Gnutella mode, this "prefetching" mode has slow SETUP and fast LOOKUP.

To balance the running times of SETUP and LOOKUP, we would choose an intermediate value of r_u in the neighborhood of $\alpha\sqrt{n}$. Section 7.4 proves a general bound on the maximum number of queries required for a given r_u.

6 The Whanaungatanga protocol

Lookups are slow in the unstructured protocol because each node's local table is a completely random sample of the key-value records. Distributed hash tables solve this problem by adding structure to the routing tables, so that queries can be directed to nodes which are more likely to know about the target key. The structured DHT literature offers an enormous variety of design alternatives. However, naïvely applying an existing design creates an opportunity for the adversary to exploit the DHT's structure to disrupt routing [18, 17]. As a consequence, we have carefully crafted Whānaungatanga's structure to avoid introducing such attacks.

Section 6.1 describes Whānaungatanga's global structure, and Section 6.2 explains how each of its idiosyncratic features relates to specific potential attacks against the structure. Sections 6.3 and 6.4 define SETUP and LOOKUP in detail. Section 7 will prove Whānaungatanga's correctness and analyze its performance.

6.1 Global structure

Whānaungatanga's structure resembles other DHTs such as Chord [19], SkipNet [9], and Kelips [8]. Like SkipNet and Chord, Whānaungatanga assumes a given, global, circular ordering $\prec$ on keys. The notation $x_1 \prec x_2 \prec \cdots \prec x_z$ means that for any indexes $i < j < k$, the key x_j is on the arc (x_i, x_k). For many applications, lexical ordering will be the natural choice for $\prec$.

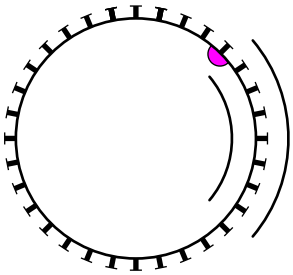


Figure 2: Each node is responsible for a subset of the keys starting at a random point on the ring.

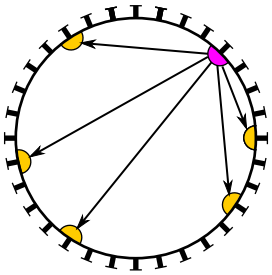


Figure 3: Finger pointers are distributed evenly on the ring.

Like SkipNet, but unlike Chord and many other DHTs, Whānaungatanga does not embed the keys into a metric space using a hash function. Therefore, the "distance" between two keys has no *a priori* meaning.

To distribute the records evenly, each node chooses a random **ID** from the set of keys in the system, and stores a table of its **successors**, those key-value records following the ID on the ring (see Figure 2). In addition, each node chooses $k = O(\log n)$ **layered IDs**, each picked randomly from the IDs in the previous layer; a separate successor table is stored for each layer. Finally, each node stores a table of pointers to random **finger** nodes, whose IDs will be distributed evenly around the ring (see Figure 3).

Whānaungatanga's structure is designed for one-hop lookups, like Kelips, and unlike Chord and SkipNet (which have smaller tables but $O(\log n)$-hop lookups). The lookup procedure is simple in principle: send a query message to the nearest finger node to the target key. If the finger and successor tables are sufficiently large, and if the chosen finger is honest, then the finger node will have the target key in its successor table. All k layers are searched independently.

6.2 Why this particular structure?

Several features of Whānaungatanga's structure, such as layers, one-hop lookups, and the lack of a hash function, are

relatively idiosyncratic compared to typical DHTs. This section briefly justifies these decisions.

Most DHTs apply some hash function to keys in order to distribute them randomly over some metric space. However, given the hash function, an active adversary can easily use trial and error to construct many keys which fall between any two neighboring honest keys. Since this warps the distribution of keys in the system, it completely defeats the purpose of the hash function. Therefore, Wh$\bar{a}$naungatanga relies only upon an arbitrary ordering $\prec$ on keys. The adversary may choose his keys to fall anywhere in the ordering; this does not affect Wh$\bar{a}$naungatanga's security.

Organizing Wh$\bar{a}$naungatanga's routing tables according to the given ordering $\prec$ enables fast lookups, but it also enables a new class of attacks on the ordering. The ordering of the honest keys is fixed, but the attacker can choose where to place his keys and his IDs. He may choose to distribute them evenly, which would be easy for a DHT to handle; or he may cluster them, attacking a particular honest key or keys [17].

A key-clustering attack would be effective if each node used a deterministic value (e.g., its IP address or public key [19], or a locally-stored key [9]) as its ID. By inserting many bogus keys immediately before a targeted key, the adversary could keep the target key out of the honest nodes' successor tables. Wh$\bar{a}$naungatanga prevents this class of attack by choosing IDs randomly from the set of keys in the system.

Just as the adversary may attack the successor tables by clustering his keys, he may attack the finger tables by clustering his IDs. If the adversary chooses his IDs to fall near a targeted key, then honest nodes may have to waste many query messages to Sybil nodes before eventually querying an honest finger. Layered IDs prevent this class of attack: if the adversary chooses to cluster his IDs within a small range, then the honest nodes will naturally cluster their next-layer IDs within the same range. As a result, the adversary cannot dominate any small range in every layer.

Finally, Wh$\bar{a}$naungatanga is a one-hop DHT, unlike many DHTs which trade longer lookup routes for smaller routing tables. This is because every level of recursion amplifies the adversary's influence by a factor of α. For example, consider a hypothetical subroutine that sends out a random walk and queries the resulting node for a record that was, itself, found using a random walk. If a walk is 50% likely to return a good node, then this subroutine would only be 25% likely to succeed. Extending the recursion to $O(\log n)$ hops would allow the adversary to turn an initially small toehold into an overwhelming advantage. Therefore, our SETUP and LOOKUP procedures specifically avoid any deeply recursive queries.

6.3 Setup

The SETUP procedure (Figure 4) takes the locally stored key-value record and the social connections as input and constructs four routing tables:

- $\mathit{fingers}(u)$: u's finger nodes, sorted by their IDs.
- $\mathit{database}(u)$: a sample of records used to construct succ.
- $\mathit{ids}(u, \ell)$: u's layer-ℓ ID, a random key.
- $\mathit{succ}(u, \ell)$: u's layer-ℓ successor records.

The global parameters r_f, r_d, r_s, and k determine the sizes of these tables; SETUP also takes the mixing time w as a parameter. Section 7 will show how all these parameters relate to Wh$\bar{a}$naungatanga's performance.

The SETUP pseudocode constructs the routing tables in three separate phases. When implemented as a distributed protocol, this simply means that each node finishes the current phase before responding to the next phase's queries. Honest nodes which respond slowly may simply be ignored.

The $\mathit{fingers}$ and $\mathit{database}$ tables are easy to construct. SETUP's first phase simply sends out r_f random walks and collects the resulting nodes into the finger table. This ensures that each node's good fingers are uniformly distributed.

The second phase sends out r_d random walks to collect a sample of the records in the social network and stores them in the $\mathit{database}$ table. These samples are used to build the successor tables, and then the $\mathit{database}$ table is discarded at the end of the setup procedure. The $\mathit{database}$ table has the good property that each honest node's key-value record is frequently represented in the other honest nodes' tables.

The final phase, and the most complex, chooses each node's IDs and constructs its successor table. The layer-zero ID is chosen by picking a random key from a random node's $\mathit{database}$. This has the effect of distributing honest layer-zero IDs evenly around the key space.

Higher-layer IDs are defined recursively: the $i + 1^{\text{th}}$ ID is chosen by picking a random finger from a random node's finger table, and using that finger's i^{th} ID. As explained above, this causes honest IDs to cluster wherever Sybil IDs have clustered, ensuring a rough balance between good fingers and bad fingers in every range of keys.

Once a node has its ID for a layer, it must collect the successor list for that ID; this is the hard part of the setup procedure. It might seem that we could solve this the same way Chord does, by bootstrapping off LOOKUP to find the ID's first successor node, then asking it for its own successor list, and so on. However, this approach is deeply recursive (Section 6.2) and would allow the adversary to fill up the successor tables with bogus records. To avoid this, Wh$\bar{a}$naungatanga fills each node's succ table without using any other node's succ table; instead, it uses only the $\mathit{database}$ tables.

The information about any node's successors is spread around the $\mathit{database}$ tables of many other nodes, so the SUCCESSORS subroutine must contact many nodes and collect little bits of the successor list together. The obvious way to do this is to ask each node v for the closest record in $\mathit{database}(v)$ following the ID. We chose another way because it is easier to prove correct in Section 7.5.1.

The SUCCESSORS subroutine repeatedly calls SUCCESSORS-SAMPLE r_s times, each time accumulating a few more potential-successors. SUCCESSORS-SAMPLE works by contacting a random node and sending it a query containing the ID. The queried node v, if it is honest, sorts all of the records in its local $\mathit{database}(v)$ by key, and then returns a small random

```
SETUP (key(·), value(·), neighbors(·); w, r_f, r_d, r_s, k)
  ▷ 1. Collect a finger list of random nodes.
1 for each node u
2     do fingers(u) ← SAMPLE-NODES(u, r_f)

  ▷ 2. Collect random records into the database table.
3 for each node u
4     do database(u) ← SAMPLE-RECORDS(u, r_d)

  ▷ 3. Choose an ID and collect a successor list in each layer.
5 for ℓ ← 0 to k
6     do for each node u
7            do ids(u, ℓ) ← CHOOSE-ID(u, ℓ)
8               succ(u, ℓ) ← SUCCESSORS(u, ℓ, r_s)

  ▷ Post-processing: Query all fingers for their IDs.
    Pre-sort the successor list by key and the finger list by ID.
9 return fingers, ids, succ
```

```
CHOOSE-ID(u, ℓ)
1 v ← RANDOM-WALK(u)
2 if ℓ = 0
3   then Choose a random (key, value) ∈ database(v)
4        return key
5   else Choose a random node f ∈ fingers(v)
6        return ids(f, ℓ − 1)

SUCCESSORS(u, ℓ, r_s)
1 R ← {}
2 for i ← 1 to r_s
3     do R ← R ∪ SUCCESSORS-SAMPLE(u, ids(u, ℓ))
4 return R

SUCCESSORS-SAMPLE(u, key_0)
1 v ← RANDOM-WALK(u)
2 {(key_1, value_1), ..., (key_{r_d}, value_{r_d})} ← database(v)
    (sorted so that key_0 ≺ key_1 ≺ ··· ≺ key_{r_d} ≺ key_0)
3 R ← {}
4 for i ← 1 to r_d
5     do With probability 1/i: R ← R ∪ {(key_i, value_i)}
6 return R
```

Figure 4: Pseudocode for Whānaungatanga's SETUP procedure to construct routing tables.

sample of the records biased towards those closer to the ID. Specifically, the record in $database(v)$ closest to the ID is always sent, the second-closest is chosen with 50% probability, the third-closest with 33% probability, and so on: if there are δ intervening records in $database(v)$ between a record and the ID, then a biased coin is flipped and the record is chosen with odds 1-to-δ. [4] This procedure ends up returning approximately $\sum_{i=1}^{r_d} 1/i \approx \log r_d + 0.577 = O(\log n)$ candidate successors for each query.

Since each SUCCESSORS-SAMPLE query is independent and random, there will be substantial overlap in the result sets, and some of the records returned will be far away from the ID and thus not really successors. Nevertheless, Section 7.5.1 will show that, for appropriate values of r_d and r_s, the union of the repeated queries will contain all the desired successor records.

After all $succ$ tables have been constructed, the $database$ tables may be discarded (although in a dynamic implementation, they may be reused). In order to quickly process lookup requests, each node should sort its successor table by key and its finger table by ID. Note that, since each node has k IDs, each finger will appear k times in the sorted table.

6.4 Lookup

The basic goal of the LOOKUP procedure is to find a finger node which is honest and which has the target key in its successor table. The SETUP procedure ensures that any honest finger f which is "close enough" to the target will have it in $succ(f)$; and, since every finger table contains many random honest nodes, it is likely to have an honest finger which is "close enough" (if r_f is big enough). However, if the adversary clusters his IDs near the target key, then LOOKUP might have to waste many queries to Sybil fingers before finding this honest finger. LOOKUP's pseudocode (Figure 5) is complex because it must foil this category of attack.

To prevent the adversary from focusing its attack on a single node's finger table, LOOKUP tries once to find the target using its own finger table, and, if that fails, repeatedly chooses a random delegate and retries the search from there.

The TRY subroutine searches the finger table for the closest layer-zero ID x to the target key key_t. It then chooses a random layer ℓ to try, and a random finger f whose ID in that layer, $ids(f, \ell)$, lies between x and the target key. TRY queries f for the target key; as an optimization, if the query fails, it may choose a new finger and retry.

If there is no clustering attack, then the layer-zero ID x is likely to be an honest ID; if there is a clustering attack, then x can only become closer to the target key. Therefore, in *either* case, any honest finger found between x and key_t will be close enough to have key_t in its successor table.

The only question remaining is: how likely is CHOOSE-FINGER to pick an honest finger? Recall that, during SETUP, if the adversary clusters his IDs between x and key_t in some layer, then the honest nodes will tend to cluster in the same range in the next layer. Thus, the adversary's fingers cannot dominate the range in the majority of layers. Now, the layer chosen by CHOOSE-FINGER is random — so, probably not dominated by the adversary — and therefore, a finger chosen randomly from that layer is likely to be honest.

In conclusion, for a random honest node's finger table, CHOOSE-FINGER has a good probability of returning an hon-

[4] The pseudocode seems inefficient, iterating over all the records in the database. However, since the returned set size is $O(\log n)$ WHP, the same effect can be achieved more efficiently by choosing $O(\log n)$ records from the database according to a probability distribution proportional to $1/i$. (Returning extra records cannot harm correctness.) Also, note that if a Sybil node tries to reply to a SUCCESSORS-SAMPLE query with a larger number of records, it can safely be ignored.

```
LOOKUP(s, key_t)
1  u ← s
2  repeat value_t ← TRY(u, key_t)
3         u ← RANDOM-WALK(s)
4    until TRY found valid value_t, or hit retry limit
5  return value_t

TRY(u, key_t)
1  {x_1, ..., x_{r_f}} ← {ids(f, 0) | f ∈ fingers(u)}
                          (sorted so key_t ⪯ x_0 ⪯ ··· ⪯ x_{r_f} ≺ key_t)
2  i ← r_f
3  repeat (f, ℓ) ← CHOOSE-FINGER(u, x_i, key_t)
4         value_t ← QUERY(f, ℓ, key_t)
5         i ← i − 1
6    until QUERY found valid value_t, or hit retry limit
7  return value_t

CHOOSE-FINGER(u, x, key_t)
1  for ℓ ← 1 to k
2       do F_ℓ ← {f ∈ fingers(u) | x ⪯ ids(f, ℓ) ⪯ key_t}
3  Choose a random ℓ ∈ {0, ..., k} such that F_ℓ is nonempty
4  Choose a random node f ∈ F_ℓ
5  return (f, ℓ)

QUERY(f, ℓ, key_t)
1  if (key_t, value_t) ∈ succ(f, ℓ) for some value_t
2    then return value_t
3  error "not found"
```

Figure 5: Pseudocode for Whānaungatanga's LOOKUP procedure to search for a key.

est finger which is close enough to have the target key in its successor table. Therefore, LOOKUP should almost always succeed after only a few calls to TRY.

7 Analysis

The preceding sections described Whānaungatanga's setting and design, and intuitively argued for its correctness. This section rigorously defines concepts which we have only informally defined above, analyzes Whānaungatanga's correctness and performance, and sketches the proofs for our main theorems. Unfortunately, space considerations preclude us from including fully detailed proofs.

7.1 Mixing time

Definition. Let P_u^i be the probability distribution of an i-step random walk starting at u, and let π be its stationary distribution. [5] The **mixing time** of a graph is the smallest w such that for all node pairs (u, v):

$$\left|P_u^w(v) - \pi(v)\right| \leq \frac{\pi(v)}{2}$$

The graph is said to be **fast mixing** if $w = O(\log n)$.

A graph's mixing time is connected with its edge expansion via the Cheeger constant; all expander graphs are fast mixing and vice versa [4].

For convenience, assume that the social graph is regular, so that the stationary distribution is uniform over nodes (i.e., $\pi(v) = 1/n$ for all v). This assumption is without loss of generality, because a standard tranformation, replacing each node with an expander, converts a general fast-mixing graph into a regular fast-mixing graph.

7.2 Escape probability and loser nodes

The key underlying observation of our protocol is that a fast mixing graph has no sparse cuts, but the attack edges form a sparse cut between the honest region and the Sybil region. Therefore, a random walk starting in the honest region is less likely to cross into the Sybil region.

Definition. Call a walk starting in the honest region an **escaping walk** if it crosses an attack edge. Let the **escape probability** p_v be the probability that a random walk starting at the node v will escape. Order the honest nodes $\{v_1, \ldots, v_n\}$ so that $p_{v_1} \geq p_{v_2} \geq \cdots \geq p_{v_n}$ (i.e., a random walk starting at v_1 is the most likely to escape). Fix an *arbitrary* constant fraction $0 < \epsilon < 1$. Call the ϵn nodes with highest escape probability $(v_1, \ldots, v_{\epsilon n})$ the **loser nodes**, and call the rest of the honest nodes the **winner nodes**.

The arbitrary fraction ϵ dividing "winners" from "losers" appears only in the analysis and not in the protocol. Its appearance reflects the reality that, due to attack edge clustering, some nodes' escape probability may be so high that random walks are useless. Naturally, the fraction of losers is related to the number of attack edges g, as we show next.

Theorem. *Order the nodes as above. For all k, $p_{v_k} \leq \frac{gw}{k}$.*

Proof. Consider choosing a random honest node and then taking one random step away from that node. This step has a g/m chance of crossing an attack edge, where m is the number of honest plus attack edges, and a $1 - g/m$ chance of crossing an honest edge, in which case the next node is a random honest node, exactly the same as the starting distribution. By repeating this process, observe that a length-w random walk starting at a random honest node has a $(1 - g/m)^w > 1 - gw/m$ chance of staying within the honest region. Thus by algebraic rearrangement we have:

$$\text{Prob}\,[\,\text{escape}\,] \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} p_{v_i} \quad < \quad \frac{gw}{m} < \frac{gw}{n}$$

$$kp_{v_k} < \sum_{i=1}^{k} p_{v_i} < \sum_{i=1}^{n} p_{v_i} \quad < \quad gw$$

Thus $p_{v_k} < \frac{gw}{k}$. □

Corollary. *If $g \ll n/w$, then for all winner nodes v, the escape probability is small: $p_v \leq \frac{gw}{\epsilon n} = O\left(\frac{gw}{n}\right) \ll 1$.*

[5] We use the standard textbook definitions for P_u^i and π,

$$P_u^i(v) \stackrel{\text{def}}{=} \sum_{v' \sim v} \frac{1}{\deg v'} P_u^{(i-1)}(v') \qquad \pi(v) \stackrel{\text{def}}{=} \lim_{i \to \infty} P_u^i(v)$$

Conversely, if the winners' escape probability must be no more than p_v, the number of losers may be up to $\epsilon n \leq \frac{gw}{p_v}$.

7.3 Effectiveness of random-walk sampling

Let $\hat{P}_u(v)$ be the probability that RANDOM-WALK(u) returns v. We now have the tools to analyze this distribution.

Escaping walks are controlled by the adversary and are distributed arbitrarily. However, for non-escaping walks, $\hat{P}_u$ is identical to the standard random walk's distribution P_u^w. Since the walk length is finite, P_u^w is nearly but not exactly uniform over honest nodes. We conservatively treat this non-uniformity as adversarially controlled, rolling both this effect and that of escaping walks into a single parameter α.

Definition. The adversary's **advantage** $\alpha \geq 1$ is defined by:

$$\alpha^{-1} \stackrel{\text{def}}{=} \min_{\text{winner } u,\ \text{honest } v} \frac{\hat{P}_u(v)}{\pi(v)} = \min_{\text{winner } u,\ \text{honest } v} n\hat{P}_u(v)$$

We can always rewrite RANDOM-WALK's distribution as a uniform component plus an adversarial component $\hat{\pi}$:

$$\hat{P}_u(v) = \frac{1}{\alpha}\pi(v) + \left(1 - \frac{1}{\alpha}\right)\hat{\pi}_u(v) = \frac{1}{\alpha}\frac{1}{n} + \left(1 - \frac{1}{\alpha}\right)\hat{\pi}_u(v)$$

In other words, RANDOM-WALK(u) can be safely treated as having a $1/\alpha$ chance of returning a uniform honest node, and a $1 - 1/\alpha$ chance of returning a node chosen by the adversary. Therefore, if a winner node sends out α random walks, it may expect to get (on average) one good sample.

Definition. A **good sample** is a uniformly random honest node. We will treat RANDOM-WALK as a process which flips a weighted coin, and with probability $1/\alpha$ returns a good sample. Otherwise, it returns an arbitrary node (honest or Sybil). The caller of RANDOM-WALK cannot distinguish between the two cases, so it cannot filter out bad samples.

Because α acts as a multiplier on the amount of work each node must do, we want α to be a small constant.

Theorem. *If $g \ll \epsilon n/w$, then $\alpha = O(1)$.*

Proof. From the definition of mixing time in Section 7.1, a non-escaping walk has at least a 1/2 chance of returning a good sample. Thus $\text{Prob}[\text{ good sample }] \geq (1 - p_v)/2$.

Recall from Section 7.2 that winner nodes have escape probability $p_v \leq gw/\epsilon n \ll 1$. Thus

$$\alpha \leq \left[\frac{1}{2}\left(1 - \frac{gw}{\epsilon n}\right)\right]^{-1} \approx 2\left(1 + \frac{gw}{\epsilon n}\right) < 4 = O(1) \quad \square$$

Observe that when g is small, α is a small constant, but when $g \gg n/w$, then α grows very rapidly. This is as expected: large g means that most walks will escape.

Lemma. *Let X be the number of good samples in the output of* SAMPLE-NODES$(u, 2\alpha r)$. *Then the expectation is $E[X] \geq 2r$, and $X \geq r$ almost certainly:* $\text{Prob}[\ X < r\ ] \leq e^{-r/4}$.

Proof. Follows immediately from standard expectation and Chernoff bound of the binomial distribution $B(r, 1/\alpha)$. $\square$

7.4 The unstructured protocol's performance

The unstructured protocol (Section 5) obviously always eventually succeeds, because some random walk will hit the node which originally stored the target key (see lines 3-4 of QUERY). However, this takes an expected αn queries. The point of the *database* table is to reduce this to $O(n/r_u)$.

Theorem. *If $r_u \leq \alpha n$, then* LOOKUP *succeeds with probability $> 1/2$ after fewer than $\frac{24\alpha^2 n}{(1-\epsilon)r_u} = O(\frac{n}{r_u})$ queries.*

Proof. Consider two cases. If $r_u < 12\alpha/(1-\epsilon)$, then we can ignore the *database*: after $\frac{24\alpha^2 n}{(1-\epsilon)r_u} > 2\alpha n$ queries, the probability that no random walk has hit the target node is at most $(1 - 1/\alpha n)^{2\alpha n} \approx e^{-2} \approx 0.135$.

If $r_u \geq 12\alpha/(1-\epsilon)$, then we focus on the *database*. A call to QUERY is useful if the random walk chooses a winner node which has not been previously queried. If fewer than half the winners have been queried yet, then each walk has at least a $(1-\epsilon)/2\alpha$ chance of reaching a new winner node. By a Chernoff bound, this means that after $\frac{24\alpha^2 n}{(1-\epsilon)r_u}$ queries, the number of unique winners queried is at least $q = 6\alpha n/r_u$, with failure probability less than $e^{-6\alpha n/4r} < e^{-3/2} < 1/4$. (Observe that $q < (1-\epsilon)n/2$, validating the assumption that the number of winners queried is less than half.)

Since each winner's *database* table is constructed independently, querying q unique winners is like sending out qr_u separate random walks, Therefore, the probability that the target key is in none of the tables is at most $(1-1/\alpha n)^{qr_u} = (1 - 1/\alpha n)^{6\alpha n/4} \approx e^{-6/4} \approx 0.223$.

Combining these results: there is a less than 1/4 chance of querying fewer than q unique winners, and less than 1/4 chance that no winner has the target key. Thus, the total probability of failure is less than 1/2. $\square$

The constant factor 24 is a very loose bound, chosen to make the proof easy: the expected number of queries is closer to $\frac{\alpha^2 n}{(1-\epsilon)r_u}$. The failure probability shrinks exponentially with the table size r_u and the number of queries.

7.5 Whanaungatanga's performance

For the same reason as the unstructured protocol, Whānaungatanga's LOOKUP will always eventually succeed if it runs for long enough: some random walk (LOOKUP, line 3) will find the target node. However, the point of Whānaungatanga's added complexity is to improve lookup performance beyond the trivial $O(n)$ algorithm. This section shows that LOOKUP uses only $O(1)$ messages to find any target key.

There are three prerequisites for a successful lookup:

1. SETUP must correctly build complete successor tables.
2. SETUP must correctly build finger tables which contain an honest finger close enough to every target key.
3. LOOKUP must be able to find that honest finger.

We consider each of these factors in turn.

7.5.1 Successor tables are correctly constructed

Definition. Let **the database** $\mathcal{D}$ be the disjoint union of all the honest nodes' $database(u)$ tables:

$$\mathcal{D} \stackrel{\text{def}}{=} \biguplus_{\text{honest } u} database(u)$$

Intuitively, we expect honest nodes to be heavily represented in the database. $\mathcal{D}$ has exactly $r_d n$ elements; we expect at least $\frac{1-\epsilon}{\alpha} r_d n$ of those to be honest nodes' records, so randomly sampling from the database is likely to get good records.

Recall that SETUP (Figure 4) uses the SUCCESSORS subroutine, which calls SUCCESSORS-SAMPLE r_s times, to find all the honest records in $\mathcal{D}$ immediately following an ID x. Consider an arbitrary successor key $y \in \mathcal{D}$, and define Δ be the number of records (honest and Sybil) in $\mathcal{D}$ between x and y. We will show that if the $database$ and $succ$ tables are sufficiently large, and Δ is sufficiently small, then y will almost certainly be collected into the successors table. Thus any winner node u's table $succ(u, \ell)$ will ultimately contain all records y close enough to the ID $x = ids(u, \ell)$.

Lemma. *For honest $y \in \mathcal{D}$ and $\Delta = \big|\{z \in \mathcal{D} | x \prec z \prec y\}\big|$,*

$$\text{Prob}\,[\, y \in \text{SUCCESSORS-SAMPLE}(x)\,] \gtrsim \frac{1-\epsilon}{\alpha^2}\frac{r_d}{n+\Delta}$$

Proof sketch. SUCCESSORS-SAMPLE begins by walking to a random node v (line 1), which is a winner with probability at least $(1-\epsilon)/\alpha$. If v is a winner, then $database(v)$ has about r_d/α good samples. Each good sample has a $1/n$ chance of being y. Thus, if $r_d \ll n$, the total probability that $y \in database(v)$ is about $\frac{1-\epsilon}{\alpha}\frac{r_d}{\alpha}\frac{1}{n} = \frac{1-\epsilon}{\alpha^2}\frac{r_d}{n}$.

Now, on average, there will approximately $\delta = \Delta/n$ records in $database(v)$ between x and y. (The distribution of this distance can be manipulated by the adversary, but not to his benefit.) After sorting (line 2), y's index will therefore be approximately $1+\delta$. So, y will be included in the result set (line 5) with probability about $\frac{1}{1+\delta} = \frac{n}{n+\Delta}$. Combine this with the probability that $y \in database(v)$ above, yielding:

$$\text{Prob}\,[\, y \in R\,] \gtrsim \left(\frac{1-\epsilon}{\alpha^2}\frac{r_d}{n}\right)\left(\frac{n}{n+\Delta}\right) = \frac{1-\epsilon}{\alpha^2}\frac{r_d}{n+\Delta} \quad \square$$

Corollary. *After calling* SUCCESSORS-SAMPLE(x) *r_s times,*

$$\text{Prob}\,[\, y \notin succ(u,\ell)\,] \lesssim \left[1 - \frac{1-\epsilon}{\alpha^2}\frac{r_d}{n+\Delta}\right]^{r_s} < e^{\frac{1-\epsilon}{\alpha^2}\frac{r_d r_s}{n+\Delta}}$$

Thus, we have $\text{Prob}\,[\, y \notin succ(u,\ell)\,] \lesssim e^{-c}$ *if*

$$r_d r_s \gtrsim c\frac{\alpha^2}{1-\epsilon}(n+\Delta) \tag{1}$$

We can intuitively interpret this result in two ways. First, regardless of Δ, we must have $r_d r_s = \Omega(n \log n)$ to ensure a complete successor table. This makes sense in the context of the Coupon Collector's Problem: the SUCCESSORS subroutine examines $r_d r_s$ random elements from $\mathcal{D}$, and it must collect the entire set of n honest records. Second, the fraction, $\Delta/|\mathcal{D}|$, of records y likely to be in $succ(u,\ell)$, grows as $\Delta/|\mathcal{D}| = O(r_s/n)$. In other words, the range of "close enough" records is proportional to r_s, as we would hope.

7.5.2 Finger tables: layer zero is evenly distributed

The previous section showed that winner nodes' successor tables are correct; we must still show that SETUP constructs correct finger tables.

Lemma. *Honest IDs are distributed evenly: a winner's layer-zero ID is, with probability $1/\alpha$, a random key from $\mathcal{D}$.*

Proof. CHOOSE-ID line 1 performs a random walk, getting a uniform honest node v with probability at least $1/\alpha$. Line 3 picks a random key from $database(v)$. Since all honest nodes contribute the same number of keys to $\mathcal{D}$, this is equivalent to picking a random element of $\mathcal{D}$. $\square$

Consider an arbitrary winner's finger table. Approximately $\frac{1-\epsilon}{\alpha} r_f$ of the fingers will themselves be winners, and so approximately $r'_f = \frac{1-\epsilon}{\alpha^2} r_f$ fingers will have layer-zero IDs drawn randomly from $\mathcal{D}$. Pick an arbitrary Δ and an arbitrary key $y \in \mathcal{D}$: we expect to find at least

$$\frac{\Delta}{|\mathcal{D}|} r'_f = \frac{1-\epsilon}{\alpha^2}\frac{\Delta}{n}\frac{r_f}{r_d}$$

of these honest fingers in the range of Δ keys in $\mathcal{D}$ before y.

Corollary. *The probability that there is no finger in the range:*

$$\text{Prob}\,[\,\textit{no finger within } \Delta \textit{ of } y\,] \lesssim \left[1 - \frac{\Delta}{|\mathcal{D}|}\right]^{r'_f} < e^{\frac{1-\epsilon}{\alpha^2}\frac{\Delta}{n}\frac{r_f}{r_d}}$$

Thus, we have $\text{Prob}\,[\,\textit{no finger}\,] \lesssim e^{-c}$ *if*

$$\frac{r_f}{r_d} \gtrsim c\frac{\alpha^2}{1-\epsilon}\frac{n}{\Delta} \tag{2}$$

We can intuitively interpret this result by observing that $\Delta/|\mathcal{D}| = O(1/r_f)$. In other words, with a large finger table, we may expect to find fingers in a small range $\Delta \ll |\mathcal{D}|$.

7.5.3 Finger tables: layers are immune to clustering

The adversary may attack layer zero of the finger tables by clustering his IDs. CHOOSE-ID lines 5–6 cause honest nodes to respond by clustering their IDs on the same keys. Line 1's random walk prevents the adversary from focusing his attack on one node's finger table: the honest nodes' clustering follows the distribution of IDs across all finger tables.

Fix an arbitrary range of keys in $\mathcal{D}$, large enough that at least one layer-zero ID is expected to fall in the range. Let β_i ("bad fingers") be the average (over winners' finger tables) of the number of Sybil fingers with layer-i IDs in the range. Likewise, let γ_i ("good fingers") be the average number of honest fingers in the range. Finally, define $\mu \stackrel{\text{def}}{=} \left(\frac{1-\epsilon}{\alpha}\right)^2$.

Lemma. *The number of good fingers in a range is at least μ times the total number of fingers in the previous layer:*

$$\gamma_{i+1} \gtrsim \mu(\gamma_i + \beta_i)$$

Proof. The average winner's finger table contains $\frac{1-\epsilon}{\alpha} r_f$ winner nodes. Each of those winner nodes chose its layer-$(i+1)$

ID by walking to a random node and querying it for a random layer-i finger. The random walk reached another winner with probability at least $\frac{1-\epsilon}{\alpha}$. By the definition of γ_i and β_i, a random layer-i finger from a random winner's table fell in our range with probability $\frac{\gamma_i+\beta_i}{r_f}$. Thus, the total average number of good layer-$(i+1)$ IDs in our range is at least

$$\gamma_{i+1} \gtrsim \left(\frac{1-\epsilon}{\alpha} r_f\right)\left(\frac{1-\epsilon}{\alpha}\right)\left(\frac{\gamma_i+\beta_i}{r_f}\right) = \mu(\gamma_i+\beta_i) \quad \square$$

Lemma. *Define the density of good fingers in a range of layer i as $\rho_i \stackrel{\text{def}}{=} \gamma_i/(\gamma_i+\beta_i)$. Then $\prod_{i=0}^{k} \rho_i \gtrsim \mu^k/r_f$.*

Proof. By the previous lemma, $\rho_i = \frac{\gamma_i}{\gamma_i+\beta_i} \gtrsim \mu\frac{\gamma_{i-1}+\beta_{i-1}}{\gamma_i+\beta_i}$. By cancelling numerators and denominators, $\prod \rho_i \gtrsim \mu^k \frac{\gamma_0}{\gamma_k+\beta_k}$. Because each finger table contains no more than r_f fingers, $\gamma_k+\beta_k \le r_k$. Also, if the range is big enough to have an honest layer-zero finger, $\gamma_0 \ge 1$. Therefore, $\prod \rho_i \gtrsim \mu^k \frac{1}{r_f}$. $\square$

This result means that the adversary's scope to affect the density of good nodes is limited. The adversary is free to choose any values of β_i between zero and r_f. However, the adversary's strategy is intuitively limited by the rule that if it halves the density of good nodes in one layer, it will necessarily double the density of good nodes in another layer. It thus turns out that the adversary's optimal strategy is to attack all layers equally, increasing its clustering β_i by the same factor from each layer to the next.

Theorem. *The average layer's density of good fingers is*

$$\bar{\rho} = \frac{1}{k+1}\sum_{i=0}^{k} \rho_i \gtrsim \frac{\mu}{e}(\mu r_f)^{-\frac{1}{k+1}} \tag{3}$$

Proof. By multiplying out terms, $\left(\sum \rho_i\right)^{k+1} \ge (k+1)! \prod \rho_i$. Substituting in Stirling's approximation $(k+1)! > \left(\frac{k+1}{e}\right)^{k+1}$ and the lemma's bound for $\prod \rho_i$ yields $\left(\sum \rho_i\right)^{k+1} \gtrsim \left(\frac{k+1}{e}\right)^{k+1} \frac{\mu^k}{r_f}$. Thus $\sum \rho_i \gtrsim (k+1)\frac{\mu}{e}(\mu r_f)^{-\frac{1}{k+1}}$. $\square$

Observe that as $k \to \infty$, the average layer's density of good fingers asymptotically approaches the limit μ/e, and that as we decrease $k \to 0$, the density of good fingers shrinks exponentially. We can get a density reasonably close to the ideal bound, $\bar{\rho} \gtrsim \mu/e^2$, by choosing the number of layers to be

$$k+1 = \log \mu r_f \tag{4}$$

7.5.4 Main result: lookup is fast

The preceding sections' tools enable us to prove that Whānaungatanga uses a constant number of messages per lookup. Define a "handicap" $\eta \stackrel{\text{def}}{=} \frac{\alpha^2}{1-\epsilon}$. Our main theorem will show that if the parameters satisfy (4) and

$$\eta^{-1} r_s \gtrsim \frac{n}{r_d} + \frac{n}{\eta^{-1} r_f} \tag{5}$$

then lookups will only need $O(1)$ queries to succeed. The formula (5) may be interpreted to mean that both $r_s r_d$ and $r_s r_f$ must be $\Omega(n)$: the first so that SUCCESSORS-SAMPLE is called enough times to collect every successor, and the second so that successor lists are longer than the distance between fingers. These would both need to be true even with no adversary; the handicap factor $\eta = O(1)$ represents the extra work required to protect against Sybil attacks.

Theorem (Main theorem)**.** *A single iteration of* TRY *succeeds with probability better than $\frac{\mu}{4e^2} = \left(\frac{1-\epsilon}{2e\alpha}\right)^2 = \Omega(1)$ if the table size parameters satisfy (4) and (5).*

Proof. Consider a range of Δ records in $\mathcal{D}$ preceeding the target key, setting $\Delta = \frac{|\mathcal{D}|}{\eta^{-1} r_f}$ to correspond to the average distance between winner fingers. First, substitute Δ and (5) into (2). Since (2) is satisfied, with probability at least $1 - 1/e$, there is an honest finger within the range Δ of the target key. TRY line 1 finds x_{r_f}, the closest layer-zero finger to the target key. This may be an honest finger or a Sybil finger, but in either case, it must be at least as close to the target key as the closest honest finger. Thus, x_{r_f} is closer than Δ with probability at least $1 - 1/e > 1/2$.

Second, recall that CHOOSE-FINGER first chooses a random layer, and then a random finger from that layer with ID between x_{r_f} and key_t. The probability of choosing any given layer i is $1/(k+1)$, and the probability of getting an honest finger from the range is ρ_i. Thus, the total probability that CHOOSE-FINGER returns an honest finger is simply the average layer's density of good nodes $\bar{\rho} = \frac{1}{k+1}\sum \rho_i$. Since we assumed $k+1 = \log \mu r_f$, the previous section showed that we have a probability of success at least $\bar{\rho} \gtrsim \mu/e^2$.

Finally, if the finger is honest, the only question remaining is whether the target key is in the finger's successor table. Substituting Δ and (5) into (1) shows that (1) is satisfied by this choice of parameters. Therefore, when QUERY checks the finger's successor table, there is an at least $1 - 1/e > 1/2$ chance that $key_t \in succ(f, \ell)$.

An iteration of TRY will succeed if three conditions are met: (1) x_{r_f} is within distance Δ of key_t; (2) CHOOSE-FINGER returns an honest finger f; (3) key_t is in f's successor table. Combining the probabilities of each of these events yields a total probability of success $\ge \frac{1}{2}\frac{\mu}{e^2}\frac{1}{2} = \frac{\mu}{4e^2}$. $\square$

Corollary. *The expected number of queries sent by* LOOKUP *is bounded by $\frac{4e^2}{\mu} = \left(\frac{2e\alpha}{1-\epsilon}\right)^2 = O(1)$. With high probability, the number of queries is bounded by $O(\log n)$.*

7.5.5 Routing tables are small

Any table size parameters which satisfy (4) and (5) will, by the previous section's proof, provide fast lookups. However, it makes sense to balance the parameters to use the minimum resources to achieve a given lookup performance.

Recall that SUCCESSORS-SAMPLE returns a set of size $\approx \log r_d$. The total number of table entries is

$$S \approx r_d + (k+1)(r_f + r_s \log r_d)$$

Approximating liberally, we take $k+1 \approx \log \mu r_f \approx \log r_d \approx \frac{1}{2}\log n \gg 1$ and minimize S subject to (5), obtaining

$$S = \frac{1}{\sqrt{2}}\eta\sqrt{n}\log^{3/2} n$$

with parameter settings $r_d \approx S/\sqrt{2\eta \log n}$, $r_f \approx S/\log n$, and $r_s \approx 2S/\log^2 n$. This makes the finger tables and the successor tables about the same size.

Optimizing for SETUP's bandwidth usage gives broadly similar results with slightly different parameter settings. The minimum bandwidth usage is $O(\eta\sqrt{n}\log^{3/2} n)$, the same as the minimum routing table size.

8 Results

We implemented the Wh$\bar{a}$naungatanga protocol (Section6) in simulation and tested its performance against graphs extracted from several social networking services.

8.1 Experimental setup

We used the social network graphs from Mislove *et al.*'s study in IMC 2007 [12]. We tested Wh$\bar{a}$naungatanga against the downloaded LiveJournal, Flickr, and YouTube social networks, and we present our results from the two larger graphs, LiveJournal and Flickr. These graphs have degree distributions following a power law (coefficient ≈ 1.7) and short average path lengths (≈ 5.8). The LiveJournal graph is estimated to cover 95.4% of users in Dec 2006, and the Flickr graph 26.9% in Jan 2007.

We performed several preprocessing steps on the input graphs. We transformed directed edges into undirected edges (the majority of links were already symmetric), and we discarded everything except the giant component of each graph. After this step, the LiveJournal graph contained 5,189,809 nodes and 48,688,097 links (average degree: 18.8), and the Flickr graph contained 1,624,992 nodes and 15,476,835 links (average degree: 19.0).

In addition, to compare with SybilLimit [21], we prepared versions of the graphs with nodes with degree less than 5 removed. After this step, the remaining LiveJournal graph has 2,735,140 nodes and 43,906,710 edges (average degree: 32.1). The remaining Flickr graph has 332,696 nodes and 13,566,771 edges (average degree: 81.6). The results were broadly similar between the truncated and the full graphs: as might be expected, the protocol performs better when more nodes are high-degree, but the change is essentially equivalent to adding more table entries to each node by increasing r. Because our simulator is memory-limited, we were not able to obtain as many data points for the full graphs; for this reason, we present data from the truncated graphs here.

We measured the mixing properties of our data sets by picking random source nodes and directly calculating the probability distributions of short random walks from those nodes. Each line in Figure 6 shows the CDF from a different source node; the thick line shows the ideal case of perfect mixing. We observe that after only 10 steps, the vast majority of nodes are near the average probability density, and that as the walk length increases, the random walks' distribution approaches the ideal. Barely visible at the bottom-left corner is the tiny fraction (≈.01%) of nodes which are relatively isolated and are reached $< 1/10$ as often as the average.

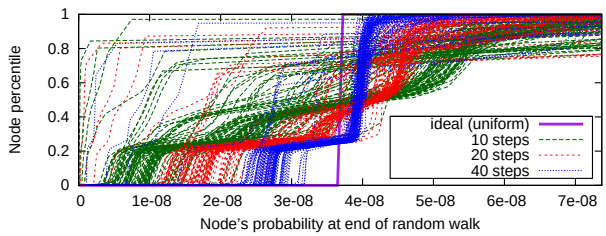


Figure 6: CDFs of random walks on the Flickr network

8.1.1 Simulated adversary behavior

To generate an instance with g attack edges, the simulator marked random nodes as "evil" until there were at least g edges between marked nodes and non-marked nodes. For example, for the Flickr graph, in the instance with $g = 6,000,050$, there are $n = 230,560$ honest nodes (with $m = 6,423,242$ honest edges) and 102,136 Sybil nodes. This method means that the number of honest nodes and honest edges actually decreases as the number of attack edges increases: therefore, our graphs tend to understate the performance of the protocol with respect to the ratio g/m of attack edges to honest edges. Also, because the number of attack edges actually decreases when the simulator marks more than $n/2$ nodes, it is not possible to test the protocol against g/m ratios substantially greater than 1.

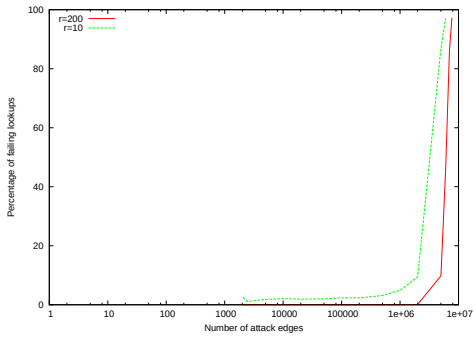
The simulated adversary does not attempt to target a specific key with a clustering attack. It swallows all random walks and returns bogus replies to all requests, which is an optimal non-targeted attack.

8.1.2 Simulated protocol and simulator behavior

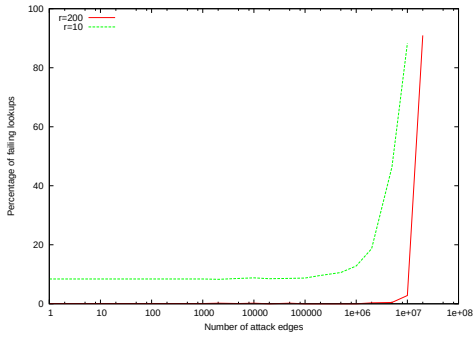
For simplicity, we chose to test the parameter settings $r_d = r_f = r_s = r$ and $k = 0$. We implemented the natural generalization of Wh$\bar{a}$naungatanga to nodes with variable degree; this differed from the protocol described in Section 6 only in that each node's r is multiplied by its degree. Therefore, when interpreting our results, one should generally multiply the given r by the average degree to get an effective average table size, and replace n with m (number of honest edges) in formulas from the analysis.

In the simulator, TRY gives up after querying 20 fingers. LOOKUP gives up after trying 20 delegate nodes. Therefore, the maximum number of messages sent by LOOKUP is 420. When r is sufficiently large that the first finger always has the target in its successor list, only one query will normally be needed.

The simulator generated a single sample of 1000 random source-target node pairs and varied the protocol parameters against the same set of pairs. When the simulator marked either the source or target of a pair as evil, the pair was removed from the sample. As a result, the highest g values for each dataset have only roughly 300 sample pairs. Therefore, our simulation could not measure failure probabilities smaller than $\sim 0.1\%$.

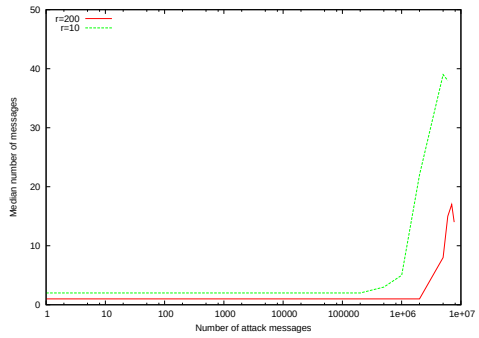


(a) Flickr

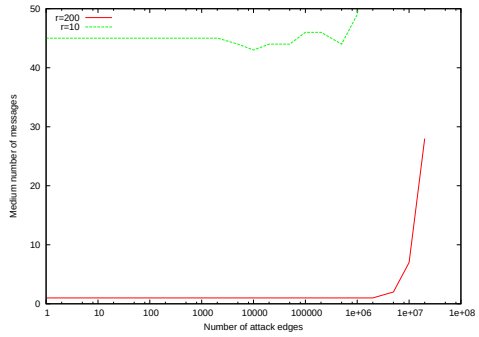


(b) Livejournal

Figure 7: Routing failure rate versus attack edges



(a) Flickr



(b) Livejournal

Figure 8: Median # of messages versus attack edges

8.2 Failure rate

Figure 7 plots the rate of lookup failures versus the number of attack edges g for two values of r (10 and 200), using $w = 10$. Recall that the protocol provides a strong guarantee only while $g \ll m/w$. The transition point, where $g \approx m/w$, is approximately 1,300,000 attack edges for the Flickr graph, and approximately 4,300,000 attack edges for the LiveJournal graph. Our graphs plot data beyond this point only to illustrate how the protocol degrades if it is used outside its intended parameters.

As the graphs illustrate, in the intended range of attack edges, the protocol has a failure rate of 0 for $r = 200$, as expected. The graph shows the protocol's graceful degradation when the adversary is very powerful. In practice, the number of failures is largely unaffected by the adversary until g becomes substantial compared to the number of honest edges. For example, for the Flickr graph, with $g = 5,000,219$ and $m = 7,871,073$, the failure rate is 9.8%. For the LiveJournal graph, the failure rate starts increasing only at approximately 10,000,000 attack edges.

The $r = 10$ lines show that it is important to chose an appropriate table size parameter r. The choice of $r = 10$, a small value, is only acceptable when there are few attack edges with the Flickr graph. With the larger LiveJournal graph, $r = 10$ is too small for efficient lookup even when there are no attack edges.

8.3 Message overhead

Figure 8 plots the median number of queries per LOOKUP for the same parameters as the graphs in figure 7. (Failing lookups are not included, so the maximum value is 420.) When $r = 200$, the median number of messages is 1 when the failure rate is 0, exactly as one would expect for a one-hop routing protocol. When the number of attack edges is larger than the range for which the protocol is intended, then the number of messages increases because LOOKUP repeatedly tries to query more predecessors and delegate the search to other nodes.

For $r = 10$, the median number of messages is 2 for the Flickr graph until the number of attack edges becomes large. This small value of r yields table sizes of roughly 800 entries on average, while $\sqrt{n} \approx 580$; therefore, the tables are just barely large enough to route about half of attempts on the first try. For the LiveJournal graph $r = 10$ is simply too small to route effectively, and so the message counts are consistently near the maximum. When $r \ll \sqrt{n}$, Wh$\bar{a}$naungatanga essentially degrades into unstructured search.

8.4 Choosing sufficiently large table size r

The pronounced "knee" in the previous graphs indicates that our analysis in Section 7 is correct: Wh$\bar{a}$naungatanga provides consistently low probability of failures and low lookup latency as long as table sizes are sufficiently large and the number of attack edges is below the breaking point at $g \approx m/w$.

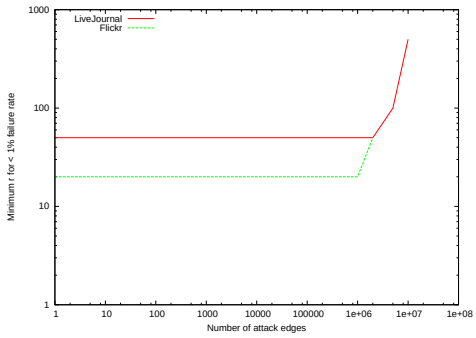


Figure 9: r versus g

Figure 9 shows the relationship between the table size r and the number of attack edges g that Wh$\bar{a}$naungatanga can withstand. It plots, for each number of attack edges g, the minimum r that yields a failure probability less than 1%. When g is below the breaking point, the number of table entries required is insensitive to g and depends on the honest network's size. On the other hand, when g is near the breaking point, adding more table entries can stave off failure, as the upticks on the right side of the graph shows.

However, for all networks, there comes a point at which no table size (which we could simulate) can maintain a low failure rate. For the Flickr and LiveJournal graphs, this point came at $g \approx 5,000,000$ and $g \approx 20,000,000$ respectively.

Again, this sharp cutoff is exactly what we would expect: when g is large compared to m, then the adversary's advantage α grows exponentially with w. Indeed, to test this hypothesis, we ran the same simulations with $w = 5$ (instead of 10). Performance was slightly poorer for small g, but improved greatly for very large g: for example, on the LiveJournal graph, failure rates were below 1% when $r = 5000$, $w = 5$, and $g = 20,000,000$.

9 Dynamic social network

This paper focused on a static view of the social network in order to highlight Wh$\bar{a}$naungatanga's algorithmic content and abstract away implementation details. However, a practical implementation would have to deal with a dynamic social network in which nodes constantly join and leave and social connections are created and destroyed.

The simplest solution would be to re-run SETUP every day to incorporate any changes to the social network. A more sophisticated approach would reuse most of the work from one SETUP run to the next, observing that a single change to the social network only affects a small fraction of the values it computes. For example, an edge creation or deletion affects only those random walks which pass through one of the edge's two nodes; while any values that depended on those walks must be recomputed, all other random walks can be reused. This approach naturally yields a dynamic protocol similar to other DHTs, where nodes update only the relevant parts of their routing tables after a join or leave.

10 Summary

This paper presents the first structured DHT routing layer which uses a social network to provide strong resilience against a Byzantine Sybil attacker with many attack edges. Routing table sizes are typical of one-hop DHTs, at $O(\sqrt{n}\log^{3/2} n)$ entries per node, and lookups take only $O(1)$ messages. This solution has applications to decentralized Internet system design, censorship resistance, and secure network routing.

11 References

[1] N. Borisov. Computational Puzzles as Sybil Defenses. *Peer-to-Peer Computing*, pages 171-176, 2006.

[2] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. *OSDI*, Boston, MA, Dec. 2002.

[3] A. Cheng and E. Friedman. Sybilproof Reputation Mechanisms. *Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 128–132, ACM Press New York, NY, USA, 2005.

[4] F. Chung. *Spectral Graph Theory.* American Mathematical Society, 1997.

[5] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil Nodes Using Social Networks. *NDSS*, San Diego, CA, Feb. 2009.

[6] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. J. Anderson. Sybil-Resistant DHT Routing. *ESORICS*, pages 305-318, 2005.

[7] J. R. Douceur. The Sybil Attack. *IPTPS Workshop*, pages 251-260, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, MA, Mar. 2002.

[8] I. Gupta, K. P. Birman, P. Linga, A. J. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. *IPTPS Workshop*, pages 160-169, M. Frans Kaashoek and Ion Stoica, ed. Springer Lecture Notes in Computer Science 2735, Berkeley, CA, Feb. 2003.

[9] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. *USENIX Symposium on Internet Technologies and Systems*, 2003.

[10] B.N. Levine, C. Shields, and N.B. Margolin. A Survey of Solutions to the Sybil Attack. University of Massachusetts Amherst, Amherst, MA, 2006.

[11] S. Marti, P. Ganesan, and H. Garcia-Molina. DHT Routing Using Social Links. *IPTPS Workshop*, pages 100-111, Geoffrey M. Voelker and Scott Shenker, ed. Springer Lecture Notes in Computer Science 3279, La Jolla, CA, Feb. 2004.

[12] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. *Internet Measurement Comference*, pages 29-42, 2007.

[13] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System. *Security Protocols Workshop*, pages 213-220, 2004.

[14] M. Ripeanu and I. T. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. *IPTPS*, pages 85-93, 2002.

[15] R. Rodrigues and B. Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. MIT CSAIL, Technical Report TR/932, Dec. 2003.

[16] H. Rowaihy, W. Enck, P. McDaniel, and T. L. Porta. Limiting Sybil Attacks in Structured P2P Networks. *INFOCOM*, pages 2596-2600, 2007.

[17] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. *INFOCOM*, 2006.

[18] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. *IPTPS Workshop*, pages 261-269, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, MA, Mar. 2002.

[19] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *ToN*, 11(1):17-32, 2003.

[20] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Rating. *NSDI*, Boston, MA, Apr. 2009.

[21] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. A Near-Optimal Social Network Defense Against Sybil Attacks. *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2008.

[22] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks Via Social Networks. *SIGCOMM*, pages 267-278, Piza, Italy, Sept. 2006.