Whānau: A Sybil-proof Distributed Hash Table

Chris Lesniewski-Laas M. Frans Kaashoek
MIT CSAIL, Cambridge, MA, USA
{ctl,kaashoek}@mit.edu

Abstract

Whānau is a novel routing protocol for distributed hash tables (DHTs) that is efficient and strongly resistant to the Sybil attack. Whānau uses the social connections between users to build routing tables that enable Sybil-resistant lookups. The number of Sybils in the social network does not affect the protocol's performance, but links between honest users and Sybils do. When there are n well-connected honest nodes, Whānau can tolerate up to $O(n/\log n)$ such "attack edges". This means that an adversary must convince a large fraction of the honest users to make a social connection with the adversary's Sybils before any lookups will fail.

Whānau uses ideas from structured DHTs to build routing tables that contain $O(\sqrt{n}\log n)$ entries per node. It introduces the idea of *layered identifiers* to counter clustering attacks, a class of Sybil attacks challenging for previous DHTs to handle. Using the constructed tables, lookups provably take constant time. Simulation results, using social network graphs from LiveJournal, Flickr, YouTube, and DBLP, confirm the analytic results. Experimental results on PlanetLab confirm that the protocol can handle modest churn.

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 [9]. This problem is particularly pernicious when the system is responsible for routing messages amongst nodes, as in the Distributed Hash Tables (DHT) [24] which underlie many peer-to-peer systems, because an attacker can prevent honest nodes from communicating altogether [23].

If a central authority certifies identities as genuine, then standard replication techniques can be used to fortify these protocols [4,20]. However, the cost of universal strong identities may be prohibitive or impractical. Instead, recent work [27,26,8,19,17,5] 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 that enables honest nodes to reliably communicate despite a concerted Sybil attack.

To solve this problem, we build on a social network composed of individual trust relations between honest people (nodes). This network might come from personal or business connections, or it might correspond to something more abstract, such as ISP peering relationships. We presume that each participant keeps track of its immediate neighbors, but that there is no central trusted node storing a map of the network.

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). As in previous work [27], 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.

In the context of a DHT, the adversary cannot prevent immediate neighbors from communicating, but can try to disrupt the DHT by creating many Sybil identities which spread misinformation. Suppose an honest node u wants to find a key y and will recognize the corresponding value (e.g., a signed data block). In a typical structured DHT, u queries another node which u believes to be "closer" to y, which forwards to another even-closer node, and so on until y is found. The Sybil nodes can disrupt this process by spreading false information (e.g., that they are close to a particular key), then intercepting honest nodes' routing queries, and responding with "no such key" or delaying queries endlessly. Unstructured protocols that work by flooding or gossip are more robust against these attacks, but pay a performance price, requiring linear time to find a key.

This paper's main contribution is **Whānau** [1], a novel protocol that is the first solution to Sybil-proof routing that has sublinear run time and space usage. Whānau achieves this performance by combining the idea of random walks from recent work [26] with a new way of constructing IDs, which we call *layered identifiers*. To store up to k keys per node, Whānau builds routing tables with $O(\sqrt{kn}\log n)$ entries per node. Using these routing tables, lookups *provably* take $O(1)$ time. Thus, Whānau's security comes at low cost: it scales similarly to one-hop DHTs that provide no security [11, 10]. We have implemented Whānau in simulation and in a simple

[1] *Whānau* , pronounced "far-no", is a Māori word. It is cognate with the Hawai'ian word *'ohana*, and means "extended family" or "kin".

instant-messaging application running on PlanetLab [2]. Experiments with real-world social graphs and these implementations confirm Whānau's theoretical properties.

Whānau provides one-hop lookups, but our implementation is not aware of network locality. Whānau also must rebuild its routing tables periodically to handle churn in the social network and in the set of keys stored in the DHT. However, its routing tables are sufficiently redundant that nodes simply going up and down doesn't impact lookups, as long as enough honest nodes remain online.

The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 informally states our goals. Section 4 states our assumptions about the social network, and provides a precise definition of "Sybil-proof". Section 5 gives an overview of Whānau's routing table structure and introduces layered IDs. Section 6 describes Whānau's setup and lookup procedures in detail. Section 7 states lemmas proving Whānau's good performance. Section 8 describes Whānau's implementation on PlanetLab [2] and in a simulator. Using these implementations Section 9 confirms its theoretical properties by simulations on social network graphs from popular Internet services, and investigates its reaction to churn on PlanetLab. Section 10 discusses engineering details and ideas for future work, and Section 11 summarizes.

2 Related work

Shortly after the introduction of scalable peer-to-peer systems based on DHTs, the Sybil attack was recognized as a challenging security problem [9, 16, 23, 22]. A number of techniques [4, 20, 22] 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 [9, 21, 3].

Several researchers [17, 19, 8, 5] proposed using social network information to fortify peer-to-peer systems against the Sybil attack. The *bootstrap graph* model [8] 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, SybilGuard and SybilLimit [27, 26] have shown how to use a "fast mixing" social network and random walks on these networks (see Section 4.1) to defend against the Sybil attack in general decentralized systems. Using SybilLimit, an honest node can certify other nodes as "probably honest", accepting at most $O(\log n)$ Sybil identities per attack edge. (Each certification uses $O(\sqrt{n})$ 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.

A few papers have adapted the idea of random walks for purposes other than SybilLimit. Nguyen *et al.* used it for Sybil-resilient content rating [25], Yu *et al.* applied it to recommendations [28], and Danezis and Mittal used it for Bayesian inference of Sybils [7]. This paper is the first to use random walks to build a Sybil-proof DHT. [2]

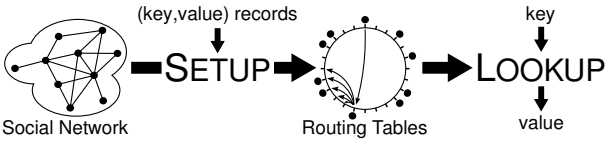


Figure 1: Overview of Whānau. SETUP builds structured routing tables which LOOKUP uses to route queries to keys.

3 Goals

As illustrated in Figure 1, the Whānau protocol is a pair of procedures SETUP and LOOKUP. SETUP$(\cdot)$ is used both to build routing tables and to insert keys. It cooperatively transforms the nodes' local parameters (e.g. key-value records, social neighbors) into a set of routing table structures stored at each node. After all nodes complete the SETUP phase, any node u can call LOOKUP(u, key) to use these routing tables to find the target $value$.

3.1 Scenario

We illustrate Whānau with a simple instant messaging (IM) application which we have implemented on PlanetLab. Whānau provides a rendezvous service for the IM clients. Each user is identified by a public key, and publishes a single self-signed tuple *(public key, current IP address)* into the DHT. [3] To send an IM to a buddy identified by the public key PK, a client looks up PK in the DHT, verifies the returned tuple's signature using PK, and then sends a packet to that IP address.

In our implementation, each user runs a Whānau node which stores that user's record, maintains contact with the user's social neighbors, and contributes to the DHT. (In this example, each node stores a single key-value record, but in general, there may be an arbitrary number k of keys stored per node.) When the user changes location, the client updates the user's record with the new IP address. The user's DHT node need not be continuously available when the user is offline, as long as a substantial fraction of honest nodes are available at any given time.

3.2 Security goal

Whānau handles adversaries who deviate from the protocol in Byzantine ways: the adversaries may make up arbitrary responses to queries from honest nodes and may create any number of pseudonyms (Sybils) which are indistinguishable from honest nodes. When we say that

[2] Our workshop paper [14] noted the opportunity and sketched an early precursor to Whānau.

[3] Of course, a realistic application would require a PKI for human-readable names, protection against replays, privacy controls, and so on.

Whānau is "Sybil-proof", we mean that LOOKUP has a high probability of returning the correct value, despite arbitrary attacks during both the SETUP and LOOKUP phases. (Section 4 makes this definition more precise.)

The adversary can always join the DHT normally and insert arbitrary key-value pairs, including a different value for a key already in the DHT. Thus, Whānau provides availability, but not integrity: LOOKUP should find all values inserted by honest nodes for the specified key, but may also return some values inserted by the adversary. Integrity is an orthogonal concern of the application: for example, the IM application filters out any bad values by verifying the signature on the returned key-value records, and ignoring records with invalid signatures. (As an optimization, DHT nodes may opt to discard bad records proactively, since they are of no use to any client and consume resources to store and transmit.)

3.3 Performance goals

Simply flooding LOOKUP queries over all links of the social network is Sybil-resistant, but not efficient [8]. 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's node through the social network, 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. Unfortunately, a large fraction of the participating nodes are contacted for every lookup, doing $O(n)$ work each time.

On the other hand, known one-hop DHTs are very efficient — requiring $O(1)$ messages for lookups and $O(\sqrt{n})$ table sizes[4]— but not secure against the Sybil attack. Our goal is to combine this optimal efficiency with provable security. 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 (i.e., highly connected nodes should do more work than casual participants). While it is possible to adapt Whānau to different policies, this paper assumes that the goal is a proportional policy.

4 Defining "Sybil-proof"

Like previous work [27, 26], Whānau relies on certain features of social networks. This section describes our assumptions, outlines why they are useful, and defines what it means for a DHT to be "Sybil-proof" under these assumptions.

[4]If $n = 5 \times 10^8$, the approximate number of Internet hosts in 2010, then a table of $\sqrt{n}$ may be acceptable for bandwidth and storage constrained devices, as opposed to a table that scales linearly with the number of hosts.

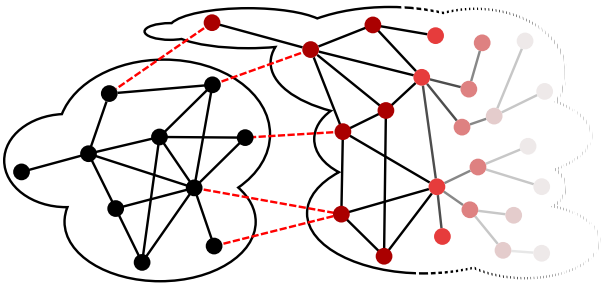


Figure 2: The social network. A sparse cut (the dashed attack edges) separates the honest nodes from the Sybil nodes. The Sybil region's size is not well-defined, since the adversary can create new pseudonyms at will.

4.1 Fast-mixing social networks

The **social network** is an undirected graph whose nodes know their immediate neighbors. Figure 2 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 [27]. An "honest" node whose software's integrity has been compromised by the adversary is considered a Sybil node.

The key assumption is that the number of attack edges, g, is small relative to the number of honest nodes, n. As pointed out by earlier work, one can justify this **sparse cut** assumption by observing that, unlike creating a Sybil identity, creating an attack edge requires the adversary to expend social-engineering effort: the adversary must convince an honest person to create a social link to one of its Sybil identities.

Whānau's correctness will depend on the sparse cut assumption, but its performance will not depend *at all* on the number of Sybils. In fact, the protocol is totally oblivious to the structure of the Sybil region. Therefore, the classic Sybil attack, of creating many fake identities to swamp the honest identities, is ineffective.

Since we rely on the existence of a sparse cut to distinguish the honest region from the Sybil region, we also assume that there is *no* sparse cut dividing the honest region in two. Given this assumption, the honest region forms 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 [6]. Roughly speaking, the ending node of a random walk is a random node in the network, with a probability distribution proportional to the node's degree. The **mixing time**, w, is the number of steps a random walk must take to reach this smooth distribution. For a fast mixing network, $w = O(\log n)$. Section 9.1 shows that graphs extracted from some real social networks are fast mixing.

	Typical magnitude	Description
n	arbitrary $n \geq 1$	number of honest nodes
m	$O(n)$	number of honest edges
w	$O(\log n)$	mixing time of honest region
k	arbitrary $k \geq 1/m$	keys stored per (virtual) node
g	$O(n/w)$	number of attack edges
ϵ	$O(gw/n)$	fraction of loser nodes

Table 1: Social network parameters used in our analysis.

4.2 Sampling by random walk

The random walk is Whānau's main building block, and is the only way the protocol uses the social network. 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, and the social network is fast mixing and has a sparse cut separating the honest nodes and Sybil nodes, then the resulting set will contain a large fraction of random honest nodes and a small number of Sybil nodes [26]. Because the initiating node cannot tell which individual samples are good and which are bad, Whānau treats all sampled nodes equally, relying only on the fact that a large fraction will be good nodes.

Some honest nodes may be near a concentration of attack edges. Such **loser nodes** have been lax about ensuring that their social connections are real people, and their view of the social graph does not contain much information. Random walks starting from loser nodes are more likely to escape into the Sybil region. As a consequence, loser nodes must 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, because a higher concentration of attack edges in one part of the network means a lower concentration elsewhere. Most honest nodes will be **winner nodes**.

In the stationary distribution, proportionally more random walks will land on high-degree nodes than low-degree nodes. To handle high-degree nodes well, each Whānau participant creates one virtual node [24] per social network edge. Thus, good random samples are distributed uniformly over the virtual nodes. All virtual nodes contribute equal resources to the DHT and obtain equal levels of service (i.e., keys stored/queried). This use of virtual nodes fulfils the policy goal (Section 3.3) of allocating both workload and trust according to each person's level of participation in the social network.

4.3 Main security definition

Table 1 summarizes the social network parameters introduced thus far. We can now succinctly define our main security property:

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

Given a $(g, \epsilon, 1/2)$-Sybil-proof protocol, it is always possible to amplify the probability of success p exponentially close to 1 by, for example, running multiple independent instances of the protocol in parallel.For example, running $3 \log_2 n$ instances would reduce the failure probability to less than $1/n^3$, essentially guaranteeing that all lookups will succeed with high probability (since there are only n^2 possible source-target node pairs). [5]

The parameter 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 may be losers.

We use the parameters in Table 1 to analyze our protocol, but do not assume that all of them are known by the honest participants. Whānau needs order-of-magnitude estimates of m, w, and k to choose appropriate table sizes and walk lengths. It does not need to know g or ϵ.

Proving that a protocol is Sybil-proof doesn't imply that it cannot be broken. For example, Whānau is Sybil-proof but can be broken by social engineering attacks that invalidate the assumption that there is a sparse cut between the honest and Sybil regions. Similarly, a protocol may be broken by using cryptographic attacks or attacks on the underlying network infrastructure. These are serious concerns, but these are not the Sybil attack as described by Douceur [9]. Whānau's novel contribution is that it is the first DHT protocol totally insensitive to the number of Sybil identities.

5 Overview of Whānau

This section outlines Whānau's main characteristics.

5.1 Challenge

The Sybil attack poses three main challenges for a structured DHT. First, structured DHTs forward queries using small routing tables at each node. Simply by creating many cheap pseudonyms, an attacker will occupy many of these table entries and can disrupt queries [23].

Second, a new DHT node builds and maintains its routing tables by querying its neighbors' tables. An attacker can reply to these queries with only its own nodes. Over time, this increases the fraction of table entries the attacker occupies [22].

Third, DHTs assign random IDs to nodes and apply hash functions to keys in order to spread load evenly. By applying repeated guess-and-check, a Sybil attacker can choose its own IDs and bypass these mechanisms. This enables **clustering attacks** targeted at a specific key. For example, if the adversary inserts many keys near the targeted key, then it might overflow the tables of honest nodes responsible for storing that part of the key space.

[5] For Whānau, it turns out to be more efficient to increase the routing table size instead of running multiple parallel instances.

Alternatively, the adversary might choose all its IDs to fall near the targeted key. Then, honest nodes might have to send many useless query messages to Sybil nodes before eventually querying an honest node.

5.2 Strawman protocol

To illustrate how random walks apply to the problem of Sybil-proof DHT routing, consider the following strawman protocol. In the setup phase, each node initiates $r = O(\sqrt{km})$ independent length-w random walks on the social network. It collects a random key-value record from the final node of each walk, and stores these nodes and records in a local table.

To perform a lookup, a node u consults its local record table. If the key is not in this table (which is likely), u broadcasts the key to the $O(\sqrt{km})$ nodes $v_1, \ldots, v_r$ in its table. If those nodes' tables are sufficiently large, with high probability, at least one node v_i will have the needed key-value record in its local table.

The strawman protocol shows how random walks address the first and second challenges above. If the number of attack edges is small, most random walks stay within the honest region. Thus, the local tables contain mostly honest nodes and records. Furthermore, nodes use only random walks to build their tables: they never look at each other's tables during the setup process. As a result, the adversary's influence does not increase over time.

The strawman sidesteps the third challenge by eschewing node IDs entirely, but this limits its efficiency. Lookups are "one-hop" in the sense that the ideal lookup latency is a single network round-trip. However, since each lookup sends a large number of messages, performance will become limited by network bandwidth and CPU load as the network size scales up. By adding structure, we can improve performance. The main challenge is to craft the structure in such a way that it cannot be exploited by a clustering attack.

5.3 Whānau's global structure

Whānau's structure resembles other DHTs such as Chord [24], SkipNet [12], and Kelips [11]. Like SkipNet and Chord, Whānau assumes a given, global, circular ordering $\prec$ on keys (e.g., lexical ordering). 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).

No metric space. Like SkipNet, but unlike Chord and many other DHTs, Whānau does *not* embed the keys into a metric space using a hash function. If Whānau were to use a hash function to map keys into a metric space, an adversary could use guess-and-check to construct many keys that fall between any two neighboring honest keys. This would warp the distribution of keys in the system and defeat the purpose of the hash function. Therefore, Whānau has no *a priori* notion of "distance" between two keys; it can determine only if one key falls between two other keys. This simple ordering provides some structure (e.g., a node can have a successor table), but still requires defenses to clustering attacks.

Fingers and successors. Most structured DHTs have routing tables with both "far pointers", sometimes called *fingers*, and "near pointers", called *leaves* or *successors*. Whānau follows this pattern. All nodes have **layered IDs** (described below) which are of the same data type as the keys. Each node's **finger table** contains $O(\sqrt{km})$ pointers to other nodes with IDs spaced evenly over the key space. Likewise, each node's **successor table** contains the $O(\sqrt{km})$ honest key-value records immediately following its ID. Finger tables are constructed simply by sending out $O(\sqrt{km})$ random walks, collecting a random sample of (honest and Sybil) nodes along with their layered IDs. Successor tables are built using a more complex sampling procedure (described in Section 6.1).

Together, an honest node's finger nodes' successor tables cover the entire set of honest keys, with high probability. This structure enables fast one-hop lookups: simply send a query message to a finger node preceding the target key. The chosen finger is likely to have the needed record in its successor table. (If not, a few retries with different fingers should suffice.) In contrast with the strawman protocol above, this approach uses a constant number of messages on average, and $O(\log n)$ messages (which may be sent in parallel) in the worst case.

Layered IDs. Whānau defends against clustering attacks using **layers**, illustrated in Figure 3. Each node uses a random walk to choose a random key as its layer-0 ID. This ensures that honest nodes' layer-0 IDs are distributed evenly over the keys stored by the system.

To pick a layer-1 ID, each node picks a random entry from its own layer-0 finger table and uses that node's ID. To pick a layer-2 ID, each node takes a random layer-1 finger's ID, and so on for each of the $\ell = O(\log km)$ lay-

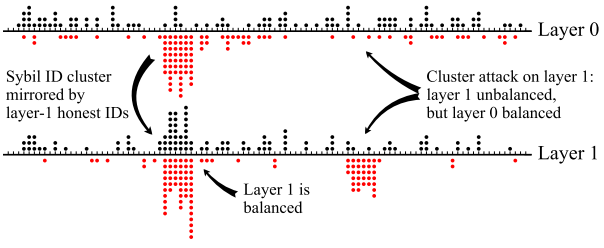


Figure 3: Honest IDs (black dots) in layer 0 are uniformly distributed over the set of keys (X axis), while Sybil IDs (red dots) may cluster arbitrarily. Honest nodes choose their layer $i + 1$ IDs from the set of all layer i IDs (honest and Sybil). Thus, most layers are balanced. Even if there is a clustering attack on a key, it will always be easy to find an honest finger near the key using a random sampling procedure.

ers. In the end, each node is present at, and must collect successor tables for, ℓ positions in the key space.

Layers defend against key clustering and ID clustering attacks. If the attacker inserts many keys near a target key, this will simply cause more honest nodes to choose layer-0 IDs in that range. The number of keys the attacker can insert is limited by the number of attack edges. Thus, a key clustering attack only shifts around the honest nodes' IDs without creating any hot or cold spots.

Nodes choose their own IDs; thus, if the attacker chooses all its layer-0 IDs to fall immediately before a target key, it might later be difficult to find an honest finger near the key. However, if the adversary manages to supply an honest node u with many clustered layer-0 IDs, then this increases the probability that u will pick one of these clustered IDs as its own layer-1 ID. As a result, the distribution of honest layer-1 IDs tends to mimic any clusters in the Sybil layer-0 IDs. This increases the honest nodes' presence in the adversary-chosen range, and increases the likelihood that layer 1 finger tables are balanced between honest and Sybil nodes.

The same pattern of balance holds for layers 2 through ℓ–1. As long as most layers have a good ratio of honest to Sybil IDs in every range, random sampling (as described in Section 6.2) can find honest fingers near any target key.

5.4 Churn

There are three sources of churn that Whānau must handle. First, computers may become temporarily unavailable due to network failures, mobility, overload, crashes, or being turned off daily. We call this **node churn**. Whānau builds substantial redundancy into its routing tables to handle Sybil attacks, and this same redundancy is sufficient to handle temporary node failures. Section 9.5 shows that increasing node churn results in a modest additional overhead.

The second source of churn is changes to the social relationships between participants. This **social churn** results in adding or deleting social connections. A single deleted link doesn't impact Whānau's performance, as long as the graph remains fast mixing and neither endpoint became malicious. (If one did become malicious, it would be treated as an attack edge.) However, Whānau doesn't immediately react to social churn, and can only incorporate added links by rebuilding its routing tables. Nodes which leave the DHT entirely are not immediately replaced. Therefore, until SETUP is invoked, the routing tables, load distribution, and so on will slowly become less reflective of the current social network, and performance will slowly degrade.

Social network churn occurs on a longer time scale than node churn. For example, data from Mislove *et al.* indicates that the Flickr social network's half-life is approximately 6 weeks [18]. Running SETUP every day, or every few minutes, would keep Whānau closely in sync with the current social graph.

The final and most challenging source of churn is changes to the set of keys stored in the DHT. This **key churn** causes the distribution of keys to drift out of sync with the distribution of finger IDs. Reacting immediately to key additions and deletions can create "hot spots" in successor tables; this can only be repaired by re-running SETUP. Thus, in the worst case, newly stored keys will not become available until the tables are rebuilt. For some applications — like the IM example, in which each node only ever stores one key — this is not a problem as long as tables are refreshed daily. Other applications may have application-specific solutions.

Unlike key churn, turnover of values does not present a challenge for Whānau: updates to the value associated with a key may always be immediately visible. For example, in the IM application, a public key's current IP address can be changed at any time by the record's owner. Value updates are not a problem because Whānau does not use the value fields when building its routing tables.

Key churn presents a trade-off between the bandwidth consumed by rebuilding tables periodically and the delay from a key being inserted to the key becoming visible. This bandwidth usage is similar to stabilization in other DHTs; however, insecure DHTs can make inserted keys visible immediately, since they do not worry about clustering attacks. We hope to improve Whānau's responsiveness to key churn; we outline one idea in Section 10.

6 The Whānau protocol

This section defines SETUP and LOOKUP in detail.

6.1 Setup

The SETUP procedure takes each node's social connections and the local key-value records to store as inputs, and constructs four routing tables:

- $ids(u, i)$: u's layer-i ID, a random key x.
- $fingers(u, i)$: u's layer-i fingers as $(id, address)$ pairs.
- $succ(u, i)$: u's layer-i successor $(key, value)$ records.
- $db(u)$: a sample of records used to construct $succ$.

The global parameters r_f, r_s, r_d, and ℓ determine the sizes of these tables; SETUP also takes an estimate of the mixing time w as a parameter. Typically, nodes will have a fixed bandwidth and storage budget to allocate amongst the tables. Section 7 and Section 9 will show how varying these parameters impacts Whānau's performance.

The SETUP pseudocode (Figure 4) constructs the routing tables in $\ell+1$ phases. The first phase sends out r_d random walks to collect a sample of the records in the social network and stores them in the db table. These samples are used to build the other tables. The db table has the good property that each honest node's stored records are frequently represented in other honest nodes' db tables.

```
SETUP (stored-records(·), neighbors(·); w, r_d, r_f, r_s, ℓ)
1  for each node u
2      do db(u) ← SAMPLE-RECORDS(u, r_d)
3  for i ← 0 to ℓ − 1
4      do for each node u
5             do ids(u, i)      ← CHOOSE-ID(u, i)
6                fingers(u, i)  ← FINGERS(u, i, r_f)
7                succ(u, i)     ← SUCCESSORS(u, i, r_s)
8  return fingers, succ

SAMPLE-RECORDS(u, r_d)
1  for j ← 1 to r_d
2      do v_j ← RANDOM-WALK(u)
3         (key_j, value_j) ← SAMPLE-RECORD(v_j)
4  return {(key_1, value_1), ..., (key_{r_d}, value_{r_d})}

SAMPLE-RECORD(u)
1  (key, value) ← RANDOM-CHOICE(stored-records(u))
2  return (key, value)

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

CHOOSE-ID(u, i)
1  if i = 0
2     then (key, value) ← RANDOM-CHOICE(db(u))
3          return key
4     else (x, f) ← RANDOM-CHOICE(fingers(u, i − 1))
5          return x

FINGERS(u, i, r_f)
1  for j ← 1 to r_f
2      do v_j ← RANDOM-WALK(u)
3         x_j ← ids(v_j, i)
4  return {(x_1, v_1), ..., (x_{r_f}, v_{r_f})}

SUCCESSORS(u, i, r_s)
1  for j ← 1 to r_s
2      do v_j ← RANDOM-WALK(u)
3         R_j ← SUCCESSORS-SAMPLE(v_j, ids(u, i))
4  return R_1 ∪ ··· ∪ R_{r_s}

SUCCESSORS-SAMPLE(u, x_0)
1  {(key_1, value_1), ..., (key_{r_d}, value_{r_d})} ← db(u)
          (sorted so that x_0 ⪯ key_1 ⪯ ··· ⪯ key_{r_d} ≺ x_0)
2  return {(key_1, value_1), ..., (key_t, value_t)}   (for small t)
```

Figure 4: SETUP procedure to build structured routing tables. Each function's first parameter is the node it executes on.

The remaining phases are used to construct the ℓ layers. For each layer i, SETUP chooses each node's IDs and constructs its successor and finger tables. The layer-0 ID is chosen by picking a random key from db. Higher-layer IDs and finger tables are defined mutually recursively. FINGERS(u, i, r_f) sends out r_f random walks and collects the resulting nodes and i^{th} layered IDs into u's i^{th} layer finger table. For $i > 0$, CHOOSE-ID(u, i) chooses u's i^{th} layered ID by picking a random finger ID stored in u's $(i-1)^{\text{th}}$ finger table. As explained in Section 5.3, this causes honest IDs to cluster wherever Sybil IDs have clustered, ensuring a rough balance between good fingers and bad fingers in any given range of keys.

Once a node has its ID for a layer, it must collect the successor list for that ID. 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, as pointed out in Section 5.1, this recursive approach would enable the adversary to fill up the successor tables with bogus records over time. To avoid this, Whānau fills each node's *succ* table without using any other node's *succ* table; instead, it uses only the *db* tables.

The information about any layered ID's successors is spread around the *db* tables of many other nodes, so the SUCCESSORS subroutine must contact many nodes and collect little bits of the successor list together. The straightforward way to do this is to ask each node v for the closest record in $db(v)$ following the ID.

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 $db(v)$ by key, and then returns the closest few records to the requestor's ID. The precise number t of records sampled does not matter for correctness, so long as t is small compared to r_d. Section 7's analysis simply lets $t = 1$.

This successor sampling technique ensures that for appropriate values of r_d and r_s, the union of the repeated queries will contain all the desired successor records. Section 7.1 will state this quantitatively, but the intuition is as follows. Each SUCCESSORS-SAMPLE query is an independent and random sample of the set of keys in the system which are near the ID. There may be substantial overlap in the result sets, but for sufficiently large r_s, we will eventually receive all immediate successors. Some of the records returned will be far away from the ID and thus not really successors, but they will show up only a few times. Likewise, bogus results returned by Sybil nodes consume some storage space, but do not affect correctness, since they do not prevent the true successors from being found.

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

TRY(u, key)
1  {(x_1, f_1), ..., (x_{r_f}, f_{r_f})} ← fingers(u, 0)
                  (sorted so key ⪯ x_1 ⪯ ··· ⪯ x_{r_f} ≺ key)
2  j ← r_f
3  repeat (f, i) ← CHOOSE-FINGER(u, x_j, key)
4         value ← QUERY(f, i, key)
5         j ← j − 1
6    until QUERY found valid value, or hit retry limit
7  return value

CHOOSE-FINGER(u, x_0, key)
1  for i ← 0 to ℓ − 1
2       do F_i ← { (x, f) ∈ fingers(u, i) | x_0 ⪯ x ⪯ key }
3  i ← RANDOM-CHOICE({i ∈ {0, ..., ℓ−1} | F_i non-empty})
4  (x, f) ← RANDOM-CHOICE(F_i)
5  return (f, i)

QUERY(u, i, key)
1  if (key, value) ∈ succ(u, i) for some value
2    then return value
3    else error "not found"
```

Figure 5: LOOKUP procedure to retrieve a record by key.

In order to process requests quickly, each node should sort its finger tables by ID and its successor tables by key.

6.2 Lookup

The basic goal of the LOOKUP procedure is to find a finger node which is honest and which has the target record in its successor table. The SETUP procedure ensures that any honest finger f which is "close enough" to the target key y will have $y \in succ(f)$. Since every finger table contains many random honest nodes, each node is likely to have an honest finger which is "close enough" (if r_f is big enough). However, if the adversary clusters 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) chooses fingers carefully to foil this category of attack.

To prevent the adversary from focusing its attack on a single node's finger table, LOOKUP tries first 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_0 to the target key key. It then chooses a random layer i to try, and a random finger f whose ID in that layer lies between x_0 and the target key. TRY then queries f for the target key.

If there is no clustering attack, then the layer-zero ID x_0 is likely to be an honest ID; if there is a clustering attack, that can only make x_0 become closer to the target key. Therefore, in *either* case, any honest finger found between x_0 and key will be close enough to have the target record in its successor table.

Only one question remains: how likely is CHOOSE-FINGER to pick an honest finger versus a Sybil finger? Recall from Section 5.3 that, during SETUP, if the adversary clustered its IDs in the range $[x_0, key]$ in layer i, then the honest nodes tended to cluster in the same range in layer $i+1$. 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 from that layer is likely to be honest.

In conclusion, for most honest nodes' finger tables, CHOOSE-FINGER has a good probability of returning an honest 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 of Whānau's performance

For the same reason as a flooding protocol, Whānau'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 Whānau's added complexity is to improve lookup performance beyond a flooding algorithm. This section sketches the reasoning why LOOKUP uses $O(1)$ messages to find any target key, leaving out most proofs; more detailed proofs can be found in an accompanying technical report [15].

To the definitions in Section 4, we will add a few more in order to set up our analysis.

Definition (good sample probability)**.** Let $\boldsymbol{p}$ be the probability that a random walk starting from a winner node returns a good sample (a random honest node). p decreases with the number of attack edges g. Specifically, we have previously shown that $p \geq \frac{1}{2}\left(1 - \frac{gw}{\epsilon n}\right)$ for any ϵ [15]. We are interested in the case where $g < \frac{\epsilon n}{2w} = O\left(\frac{n}{w}\right)$. In this case, we have that $p > 1/4$, so a substantial fraction of random walks return good samples.

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

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

Intuitively, we expect honest nodes' records to be heavily represented in $\mathcal{D}$. $\mathcal{D}$ has exactly $r_d m$ elements; we expect at least $(1-\epsilon)pr_d m$ of those to be from honest nodes.

Definition (distance metric d_{xy})**.** Recall from Section 5.3 that Whānau has no *a priori* notion of distance between two keys. However, with the definition of $\mathcal{D}$, we can construct an *a posteriori* distance metric.
Let $\mathcal{D}_{xy} \stackrel{\text{def}}{=} \{z \in \mathcal{D} \mid x \preceq z \prec y\}$ be all the records (honest and Sybil) in $\mathcal{D}$ on the arc $[x, y)$. Then define

$$d_{xy} \stackrel{\text{def}}{=} \frac{|\mathcal{D}_{xy}|}{|\mathcal{D}|} = \frac{|\mathcal{D}_{xy}|}{r_d m} \in [0, 1)$$

Note that d_{xy} is not used (or indeed, observable) by the protocol; we use it only in the analysis.

7.1 Winner successor tables are correct

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 $y \in \mathcal{D}$. If the r_d and r_s are sufficiently large, and d_{xy} is sufficiently small, then y will almost certainly be returned by some call to SUCCESSORS-SAMPLE. Thus, any winner node u's table $succ(u, i)$ will ultimately contain all records y close enough to the ID $x = ids(u, i)$.

Lemma. *Call* SUCCESSORS-SAMPLE(x) r_s *times. We then have (for* $r_d, d_{xy}^{-1} \gg 1$ *and* $r_s \ll n$*) a* Prob[*fail*] *of:*

$$\text{Prob}[y \notin succ(u, i)] \lesssim \left[1 - \frac{(1-\epsilon)p}{1 + \frac{km}{pr_d}} e^{-r_d d_{xy}}\right]^{r_s}$$

Under the simplifying assumption $r_d < d_{xy}^{-1} \ll km$*:*

$$\text{Prob}[y \notin succ(u, i)] \lesssim e^{-e(1-\epsilon)p^2 \frac{r_s r_d}{km}} \quad (1)$$

We can intuitively interpret this result as follows: to get a complete successor table with high probability, we need $r_s r_d = \Omega(km \log km)$. This is related to the Coupon Collector's Problem: the SUCCESSORS subroutine examines $r_s r_d$ random elements from $\mathcal{D}$, and it must examine the entire set of km honest records.

7.2 Layer zero IDs are evenly distributed

Consider an arbitrary winner u's layer-zero finger table $\mathcal{F}_0 = fingers(u, 0)$: approximately pr_f of the nodes in $\mathcal{F}_0$ will be random honest nodes. Picking a random honest node $f \in \mathcal{F}_0$ and then picking a random key from $db(f)$ is the same as picking a random key from $\mathcal{D}$. Thus, pr_f of the IDs in $\mathcal{F}_0$ are random keys from $\mathcal{D}$. For any keys $x, y \in \mathcal{D}$, the probability that a random honest finger's layer-zero ID falls in the range $[x, y)$ is simply d_{xy}.

Lemma. *With* r_f *fingers, we have a* Prob[*fail*] *of:*

$$\text{Prob}[\textit{ no layer-0 finger in } [x, y)\ ] \lesssim (1 - d_{xy})^{pr_f} \quad (2)$$

We expect to find approximately $pr_f d_{xy}$ *of these honest fingers with IDs in the range* $[x, y)$.

We can intuitively interpret this result as follows: to see $\Omega(1)$ fingers in $[x, y)$ with high probability, we need $r_f = \Omega(\log m / d_{xy})$. In other words, large finger tables enable nodes to find a layer-0 finger in any small range of keys. Thus layer-0 finger tables tend to cover $\mathcal{D}$ evenly.

7.3 Layers are immune to clustering

The adversary may attack the finger tables by clustering its IDs. CHOOSE-ID line 4 causes honest nodes to respond by clustering their IDs on the same keys.

Pick any keys $x, y \in \mathcal{D}$ sufficiently far apart that we expect at least one layer-zero finger ID in $[x, y)$ with high probability (as explained above). Let β_i ("bad fingers") be the average (over winners nodes' finger tables) of the number of Sybil fingers with layer-i IDs in $[x, y)$. Likewise, let γ_i ("good fingers") be the average number of winner fingers in $[x, y)$. Define $\boldsymbol{\mu} \stackrel{\text{def}}{=} (1 - \epsilon)p$.

Lemma. *The number of good fingers in* $[x, y)$ *is proportional to the total number of fingers in the previous layer:*

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

Corollary. *Let the density* $\boldsymbol{\rho_i}$ *of winner fingers in layer* i *be* $\rho_i \stackrel{def}{=} \gamma_i/(\gamma_i + \beta_i)$*. Then* $\prod_{i=0}^{\ell-1} \rho_i \gtrsim \mu^{\ell-1}/(1-\mu)r_f$.

Because the density of winner fingers ρ_i is bounded below, this result means that the adversary's scope to affect ρ_i is limited. The adversary may strategically choose any values of β_i between zero and $(1 - \mu)r_f$. However, the adversary's strategy is limited by the fact that if it halves the density of good nodes in one layer, the density of good nodes in another layer will necessarily double.

Theorem. *The average layer's density of winner fingers is at least* $\bar{\rho} \stackrel{def}{=} \frac{1}{\ell}\sum_{i=0}^{\ell-1} \rho_i \gtrsim \frac{\mu}{e}\left[(1-\mu)\mu r_f\right]^{-\frac{1}{\ell}}$.

Observe that as $\ell \to 1$, the average layer's density of good fingers shrinks exponentially to $O(1/r_f)$, and that as $\ell \to \infty$, the density of good fingers asymptotically approaches the limit μ/e. We can get $\bar{\rho}$ within a factor of e of this ideal bound by setting the number of layers ℓ to

$$\ell = \log\left[(1-\mu)\mu r_f\right] \quad (3)$$

For most values of $\mu \in [0, 1]$, $\ell \approx \log r_f$. However, when μ approaches 1 (no attack) or 0 (strong attack), $\ell \to 1$.

7.4 Main result: lookup is fast

The preceding sections' tools enable us to prove that Whānau uses a constant number of messages per lookup.

Theorem (Main theorem)**.** *Define* $\boldsymbol{\kappa} = kme/(1-\epsilon)p^3$*. Suppose that we pick* r_s*,* r_f*,* r_d*, and* ℓ *so that* (3) *and* (4) *are satisfied, and run* SETUP *to build routing tables.*

$$r_s r_f > \frac{r_s r_d}{p} > \kappa \quad (4)$$

Now run LOOKUP *on any valid key* y*. Then, a single iteration of* TRY *succeeds with probability better than* $\text{Prob}[\textit{ success }] > \frac{1}{20}(1-\epsilon)p = \Omega(1)$.

The value κ is the aggregate storage capacity km of the DHT times an overhead factor $e/(1-\epsilon)p^3$ which represents the extra work required to protect against Sybil attacks. When $g < \frac{\epsilon n}{2w}$, this overhead factor is $O(1)$.

The formula (4) may be interpreted to mean that both $r_s r_d$ and $r_s r_f$ must be $\Omega(\kappa)$: 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.

Proof sketch. Let $x \in \mathcal{D}$ be a key whose distance to the target key y is $d_{xy} = 1/pr_f$, the average distance between honest fingers.

First, substitute the chosen d_{xy} into (2). By the lemma, the probability that there is an honest finger $x_h \in [x, y)$ is at least $1 - 1/e$. TRY line 1 finds x_{r_f}, the closest layer-zero finger to the target key, and TRY passes it to CHOOSE-FINGER as x_0. x_0 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 x_h. Thus, $x_0 \in [x, y)$ with probability at least $1 - 1/e$.

Second, recall that CHOOSE-FINGER first chooses a random layer, and then a random finger f from that layer with ID $x_f \in [x_0, y]$. The probability of choosing any given layer i is ℓ^{-1}, and the probability of getting an honest finger from the range is ρ_i from Section 7.3. Thus, the total probability that CHOOSE-FINGER returns an honest finger is simply the average layer's density of good nodes $\frac{1}{\ell}\sum \rho_i = \bar{\rho}$. Since we assumed (3) was satisfied, Section 7.3 showed that the probability of success is at least $\bar{\rho} \gtrsim (1-\epsilon)p/e^2$.

Finally, if the chosen finger f is honest, the only question remaining is whether the target key is in f's successor table. Substituting $d_{x_f y} < d_{xy}$ and (4) into (1) yields $\text{Prob}[\, y \in succ(f)\,] \gtrsim 1 - 1/e$. Therefore, when QUERY$(f, y)$ checks f's successor table, it succeeds with probability at least $1 - 1/e$.

A TRY iteration will succeed if three conditions hold: (1) $x_f \in [x, y)$; (2) CHOOSE-FINGER returns a winning finger f; (3) $y \in succ(f)$. Combining the probabilities calculated above for each of these events yields the total success probability $\left(1-\frac{1}{e}\right)\frac{(1-\epsilon)p}{e^2}\left(1-\frac{1}{e}\right) > \frac{1}{20}(1-\epsilon)p$. □

Corollary. *The expected number of queries sent by* LOOKUP *is bounded by* $\frac{20}{(1-\epsilon)p} = O(1)$. *With high probability, the maximum number of queries is* $O(\log n)$.

7.5 Routing tables are small

Each (virtual) node has $S = r_d + \ell(r_f + r_s)$ table entries in total. To minimize S subject to (4), set $r_s = r_f = \sqrt{\kappa}$ and $r_d = p\sqrt{\kappa}$. Therefore, the optimal total table size is $S \approx \sqrt{\kappa}\log\kappa$, so $S = O(\sqrt{km}\log km)$, as expected.

As the number of attack edges g increases, the required table size grows as $(1-\epsilon)^{-1/2}p^{-3/2}$. A good approximation for this security overhead factor is $1 + 2\sqrt{\frac{gw}{n}} + 6\frac{gw}{n}$ when $g < \frac{n}{6w}$. Thus, overhead grows linearly with g.

As one might expect for a one-hop DHT, the optimum finger tables and the successor tables are the same size. The logarithmic factor in the total table size comes from the need to maintain $O(\log km)$ layers to protect against clustering attacks. If the number of attack edges is small, (3) indicates that multiple layers are unnecessary. This is consistent with the experimental data in Section 9.3.

8 Implementation

We have implemented Whānau in a simulator and on PlanetLab. To simulate very large networks — some of the social graphs we use have millions of nodes — we wrote our own simulator. Existing peer-to-peer simulators don't scale to such a large number of nodes, and our simulator uses many Whānau-specific optimizations to reduce memory consumption and running time. The simulator directly implements the protocol as described in Figures 4 and 5, takes a static social network as input, and provides knobs to experiment with Whānau's different parameters. The simulator does not simulate real-world network latencies and bandwidths, but only counts the number of messages that Whānau sends. The primary purpose of the simulator is to validate the correctness and scaling properties of Whānau with large social networks.

We also implemented Whānau and the IM application in Python on PlanetLab. This implementation runs a message-passing protocol to compute SETUP and uses RPC to implement LOOKUP. When a user starts a node, the user provides the keys and current IP addresses that identify their social neighbor nodes. The IM client stores its current IP address into the DHT. When a user wants to send an IM to another user, the IM client looks up the target user's contact information in the DHT and authenticates the returned record using the key. If the record is authentic, the IM application sends the IM to the IP address in the record. Whānau periodically rebuilds its tables to incorporate nodes which join and leave.

The average latency for a lookup is usually one round-trip on PlanetLab. Using locality-aware routing, Whānau could achieve lower than one network round-trip on average, but we haven't implemented this feature yet.

Our PlanetLab experiments were limited by the number of PlanetLab nodes available and their resources: we were able to run up to 4000 Whānau nodes simultaneously. Unfortunately, at scales smaller than this, Whānau nearly reduces to simple broadcast. Given this practical limitation, it was difficult to produce insightful scaling results on PlanetLab. Furthermore, although results were broadly consistent at small scales, we could not cross-validate the simulator at larger scales. The PlanetLab experiments primarily demonstrated that Whānau works on a real network with churn, varying delays, and so on.

	n=#nodes	m=#edges	avg. degree
Flickr	1,624,992	15,476,835	9.52
LiveJournal	5,189,809	48,688,097	9.38
YouTube	1,134,890	2,987,624	2.63
DBLP	511,163	1,871,070	3.66

Table 2: Properties of the input data sets.

9 Results

This section experimentally verifies several hypotheses: (1) real-world social networks exhibit the properties that Whānau relies upon; (2) Whānau can handle clustering attacks (tested by measuring its performance versus table size and the number of attack edges); (3) layered IDs are essential for handling clustering attacks; (4) Whānau achieves the same scalability as insecure one-hop DHTs; and (5) Whānau can handle node churn in Planetlab.

Our Sybil attack model permits the adversary to create an unlimited number of pseudonyms. Since previous DHTs cannot tolerate this attack at all, this section does not compare Whānau's Sybil-resistance against previous DHTs. However, in the non-adversarial case, the experiments do show that Whānau scales like any other insecure one-hop DHT, so (ignoring constant factors such as cryptographic overhead) adding security is "free". Also, similarly to other (non-locality-aware) one-hop DHTs, the lookup latency is one network round-trip.

9.1 Real-world social nets fit assumptions

Nodes in the Whānau protocol bootstrap from a social network to build their routing tables. It is important for Whānau that the social network is fast mixing: that is, a short random walk starting from any node should quickly approach the stationary distribution, so that there is roughly an equal probability of ending up at any edge (virtual node). We test if this fast-mixing property holds for social network graphs, extracted from Flickr, LiveJournal, YouTube, and DBLP, which have also been used in other studies [18, 26]. These networks correspond to real-world users and their social connections. The LiveJournal graph was estimated to cover 95.4% of the users in Dec 2006, and the Flickr graph 26.9% in Jan 2007.

We preprocessed the input graphs by discarding unconnected nodes and transforming directed edges into undirected edges. (The majority of links were already symmetric.) The resulting graphs' basic properties are shown in Table 2. The node degrees follow power law distributions, with coefficients between 1.6 and 2 [18].

To test the fast-mixing property, we sample the distribution of random walks as follows. We pick a random starting edge i, and for each ending edge j, compute the probability p_{ij} that a walk of length w ends at j. Computing p_{ij} for all m possible starting edges i is too time-intensive, so we sampled 100 random starting edges i and computed p_{ij} for all m end edges j. For a fast-mixing network, we expect the probability of ending up at a particular edge to approach $1/m$ as w increases to $O(\log n)$.

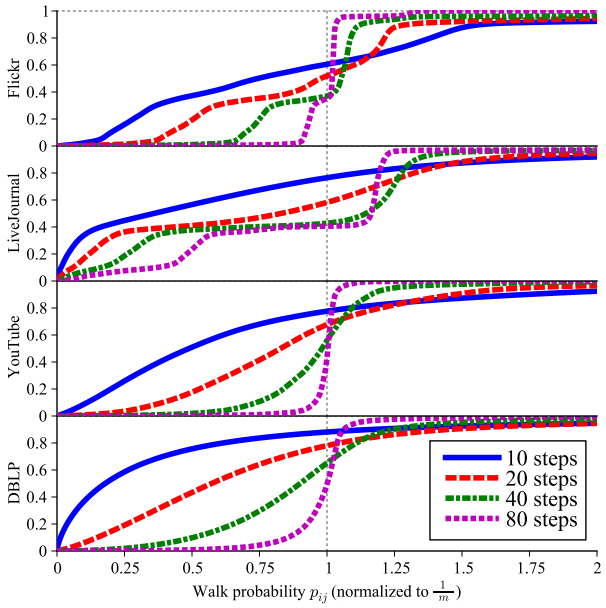


Figure 6: Mixing properties of social graphs. Each line shows a CDF of the probability that a w-step random walk ends on a particular edge. The X axis is normalized so that the mean is 1.

Figure 6 plots the CDF of p_{ij} for increasing values of w. To compare the different social graphs we normalize the CDFs so that they have the same mean. Thus, for all graphs, $p_{ij} = 1/m$ corresponds to the ideal line at 1. As expected, as the number of steps increases to 80, each CDF approaches the ideal uniform distribution.

The CDFs at $w = 10$ are far from the ideal distribution, but there are two reasons to prefer smaller values of w. First, the amount of bandwidth consumed scales as w. Second, larger values of w increase the chance that a random walk will return a Sybil node. Section 9.2 will show that Whānau works well even when the distribution of random walks is not perfect.

Recall from Section 4.2 that when a fast-mixing social network has a sparse cut between the honest nodes and Sybil nodes, random walks are a powerful tool to protect against Sybil attacks. To confirm that this approach works with real-world social networks, we measured the probability that a random walk escapes the honest region of the Flickr network with different numbers of attack edges. To generate an instance with g attack edges, we marked random nodes as Sybils until there were at least g edges between marked nodes and non-marked nodes, and then removed any honest nodes which were connected only to Sybil nodes. For example, for the Flickr graph, in the instance with $g = 1,940,689$, there are $n = 1,442,120$ honest nodes (with $m = 13,385,439$ honest edges) and 182,872 Sybil nodes. Since increasing

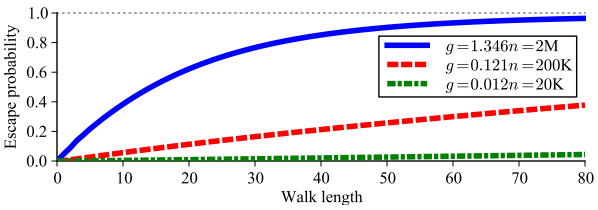


Figure 7: Escape probability on the Flickr network.

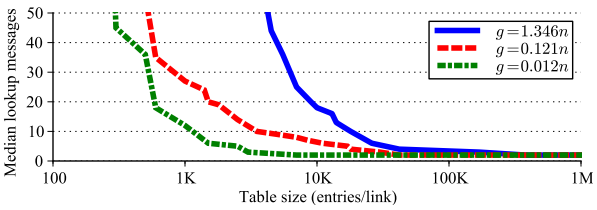


Figure 8: Number of messages used by LOOKUP decreases as table size increases (Flickr social network).

the number of attack edges this way actually consumes honest nodes, it is not possible to test the protocol against g/n ratios substantially greater than 1.

Figure 7 plots the probability that a random walk starting from a random honest node will cross an attack edge. As expected, this escape probability increases with the number of steps and with the number of attack edges. When the number of attack edges is greater than the number of honest nodes, the adversary has convinced essentially all of the system's users to form links to its Sybil identities. In this case, long walks almost surely escape from the honest region; however, short walks still have substantial probability of reaching an honest node. For example, if the adversary controls 2 million attack edges on the Flickr network, then each user has an average of 1.35 links to the adversary, and random walks of length 40 are 90% Sybils. On the other hand, random walks of length 10 will return 60% honest nodes, although those honest nodes will be less uniformly distributed than a longer random walk.

9.2 Performance under clustering attack

To evaluate Whānau's resistance against the Sybil attack, we ran instances of the protocol using a range of table sizes, number of layers, and adversary strengths. For each instance, we chose random honest starting nodes and measured the number of messages used by LOOKUP to find randomly chosen target keys. Our analysis predicted that the number of messages would be $O(1)$ as long as $g \ll n/w$. Since we used a fixed $w = 10$, the number of messages should be small when the number of attack edges is less than 10% of the number of honest nodes. We also expected that increasing the table size would reduce the number of messages.

Our simulated adversary employs a clustering attack on the honest nodes' finger tables, choosing all of its IDs to immediately precede the target key. In a real-world deployment of Whānau, it is only possible for an adversary to target a small fraction of honest keys in this way: to increase the number of Sybil IDs near a particular key, the adversary must move some Sybil IDs away from other keys. However, in our simulator, we allowed the adversary to change its IDs between every LOOKUP operation: that is, it can start over from scratch and adapt its attack to the chosen target key. Our results therefore show Whānau's worst case performance, and not the average case performance for random target keys.

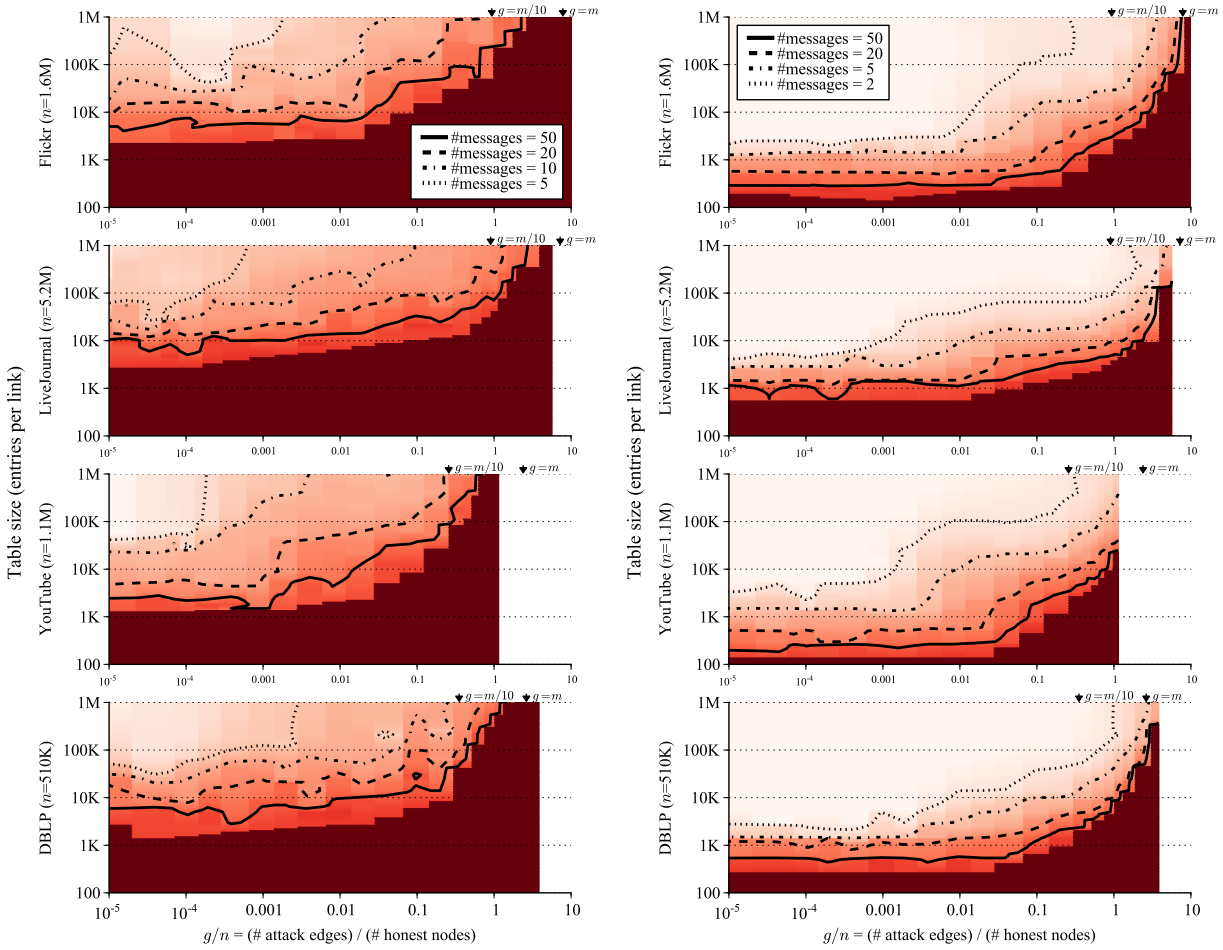
Figure 8 plots the number of messages required by LOOKUP versus table size. Since our policy is that resources scale with node degree (Section 3.3), we measure table size in number of entries per social link. Each table entry contains a key and a node's address (finger tables) or a key-value pair (successor and db tables).

As expected, the number of messages decreases with table size and increases with the adversary's power. For example, on the Flickr network and with a table size of 10,000 entries per link, the median LOOKUP required 2 messages when the number of attack edges is 20,000, but required 20 messages when there are 2,000,000 attack edges. The minimum resource budget for fast lookups is $1,000 \approx \sqrt{n}$: below this table size, LOOKUP messages increased rapidly even without any attack. Under a massive attack ($g > n$) LOOKUP could still route quickly, but it required a larger resource budget of $\gtrsim 10,000$.

Figure 9 shows the full data set of which Figure 8 is a slice. Figure 9(a) shows the number of messages required for 100% of our test lookups to succeed. Of course, most lookups succeeded with far fewer messages than this upper bound. Figure 9(b) shows the number of messages required for 50% of lookups to succeed. The contour lines for maximum messages are necessarily noisier than for median messages, because the lines can easily be shifted by the random outcome of a single trial. The median is a better guideline to Whānau's expected performance: for a table size of 5,000 on the Flickr graph, most lookups will succeed within 1 or 2 messages, but a few outliers may require 50 to 100 messages.

We normalized the X-axis of each plot by the number of honest nodes in each network so that the results from different datasets could be compared directly. Our theoretical analysis predicted that Whānau's performance would drop sharply (LOOKUP messages would grow exponentially) when $g > n/10$. However, we observed that, for all datasets, this transition occurs in the higher range $m/10 < g < m$. In other words, the analytic prediction was a bit too pessimistic: Whānau functions well until a substantial fraction of all edges are attack edges.

When the number of attack edges g was below $n/10$, we observed that performance was more a function of ta-



(a) Maximum messages required for every lookup to succeed.

(b) Median messages required for lookups to succeed.

Figure 9: Heat map and contours of the number of messages used by LOOKUP, versus attacker strength and table size. In the light regions at upper left, where there are few attack edges and a large resource budget, LOOKUP succeeded using only one message. In the dark regions at lower right, where there are many attack edges and a small resource budget, LOOKUP needed more than the retry limit of 120 messages. Wedges indicate where $g = m/w$ and $g = m$; when $g \gg m/w$, LOOKUP performance degrades rapidly. The plots' right edges do not line up because it was not always possible to create an adversary instance with $g = 10n$.

ble size, which must always be at least $\Omega(\sqrt{m})$ for Whānau to function, than of g. Thus, Whānau's performance is insensitive to relatively small numbers of attack edges.

9.3 Layers vs. clustering attacks

Section 9.2 showed that Whānau handles clustering attacks. For the plots in Figure 9, we simulated several different numbers of layers and chose the best-performing value for a given table size. This section evaluates whether layers are important for Whānau's attack resistance, and investigates how the number of layers should be chosen.

Are layers important? We ran the same experiment as in Section 9.2, but we held the total table size at a constant 100,000 entries per link. We varied whether the protocol spent those resources on more layers, or on bigger per-layer routing tables, and measured the median number of messages required by LOOKUP.

We would expect that for small-scale attacks, one layer is best, because layers come at the cost of smaller per-layer tables. For more large-scale attacks, more layers is better, because layers protect against clustering attacks. Even for large-scale attacks, adding more layers yields quickly diminishing returns, and so we only simulated numbers of layers between 1 and 10.

The solid lines in Figure 10 shows the results for the clustering attack described in Section 9.2. When the number of attack edges is small, the best performance would be achieved by spending all resources on bigger routing tables, mostly avoiding layers. For Flickr, layers become important when the number of attack edges exceeds 5,000 (0.3% of n); for $g > 20,000$, a constant

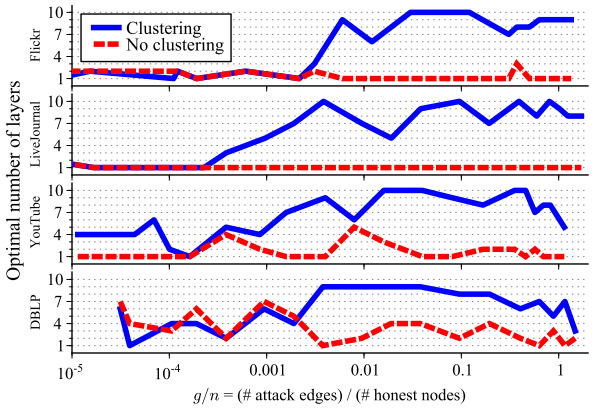


Figure 10: Optimal layers versus attacker power. The resource budget was fixed at 100K table entries per link.

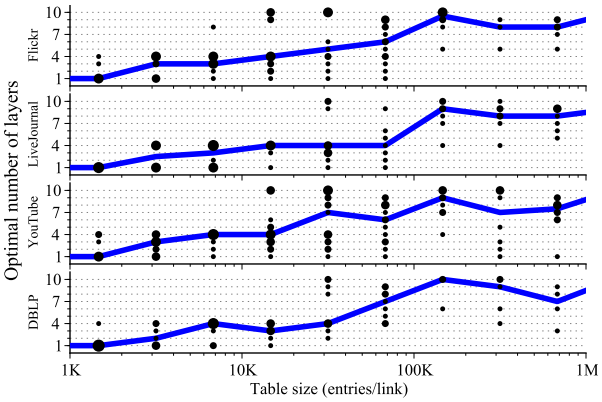


Figure 11: Optimal layers versus resource budget. Each point is a table size / attacker power instance. Larger points correspond to multiple instances. The trend line passes through the median point for each table size.

number of layers (around 8) would yield the best performance. At high attack ratios (around $g/n \gtrsim 1$), the data becomes noisy because performance degrades regardless of the choice of layers.

The dashed lines in Figure 10 show the same simulation, but pitted against a naïve attack: the adversary swallows all random walks and returns bogus replies to all requests, but does not cluster its IDs. This control data clearly shows that multiple layers are only helpful against a clustering attack. The trends are clearer for the larger graphs (Flickr and LiveJournal) than for the smaller graphs (YouTube and DBLP). 100,000 table entries is very large in comparison to the smaller graphs' sizes, and therefore the differences in performance between small numbers of layers are not as substantial.

How many layers should nodes use? The above data showed that layers improve Whānau's resistance against powerful attacks but are not helpful when the DHT is not under attack. However, we cannot presume that nodes know the number of attack edges g, so the number of layers must be chosen in some other way. Since layers cost resources, we would expect the optimal number of layers to depend on the node's resource budget. If the number of table entries is large compared to $\sqrt{m}$, then increasing the number of layers is the best way to protect against powerful adversaries. On the other hand, if the number of table entries is relatively small, then no number of layers will protect against a powerful attack; thus, nodes should use a smaller number of layers to reduce overhead.

We tested this hypothesis by re-analyzing the data collected for Section 9.2. For a given table size, we computed the number of layers that yielded optimal performance over a range of attack strengths. The results are shown in Figure 11. The overall trend is clear: at small table sizes, fewer layers is preferred, and at large table sizes, more layers is better.

The optimal number of layers is thus a function of the social network size and the resource budget, and we presume that honest nodes know both of these values at least approximately. Since Whānau's performance is not very sensitive to small changes in the number of layers, a rough estimate is sufficient to get good performance over a wide range of situations.

9.4 Whānau's scalability

Whānau is designed as a one-hop DHT. We collected simulated data to confirm that Whānau's performance scales asymptotically the same as an insecure one-hop DHT such as Kelips [11]. Since we don't have access to a wide range of social network datasets of different sizes, we generated synthetic social networks with varying numbers of nodes using the standard technique of preferential attachment [1], yielding power-law degree distributions with exponents close to 2. For each network, we simulated Whānau's performance for various table sizes and layers, as in the preceding sections. Since our goal was to demonstrate that Whānau reduces to a standard one-hop DHT in the non-adversarial case, we did not simulate any adversary.

Figure 12 plots the median number of LOOKUP messages versus table size and social network size. For a one-hop DHT, we expect that, holding the number of messages to a constant $O(1)$, the required table size scales as $O(\sqrt{m})$: the blue line shows this predicted trend. The heat map and its contours (black lines) show simulated results for our synthetic networks. For example, for $m = 10,000,000$, the majority of lookups succeeded using 1 or 2 messages for a table size of $\approx 2,000$ entries per link. The square and triangle markers plot our four real-world datasets alongside the synthetic networks for comparison. While each real network has idiosyncratic

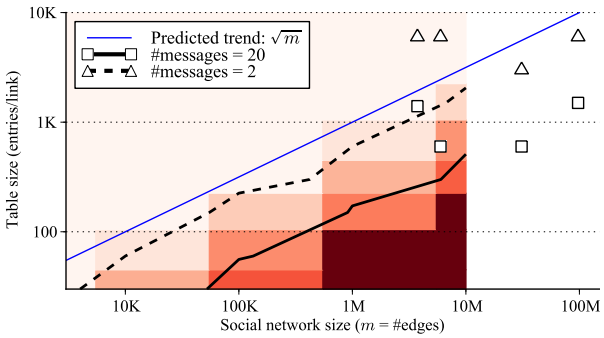


Figure 12: Number of messages used by LOOKUP, versus system size and table size. The heat map and contour lines show data from synthetic networks, while the markers show that real-world social networks fall roughly onto the same contours. Whānau scales like a one-hop DHT.

features of its own, it is clear that the table sizes follow the $O(\sqrt{m})$ scaling trend we expect of a one-hop DHT.

9.5 PlanetLab and node churn

Whānau's example IM application runs on PlanetLab. We performed an experiment in which we started 4000 virtual nodes, running on 400 PlanetLab nodes. This number of virtual nodes is large enough that, with a routing table size of 200 entries per social link, most requests cannot be served from local tables. Each node continuously performed lookups on randomly-chosen keys.

We simulated node churn by inducing node failure and recovery events according to a Poisson process. These events occurred at an average rate of two per second, but we varied the average node downtime. At any given time, approximately 10% or 20% of the virtual nodes were offline. (In addition to simulating 10% and 20% failures, we simulated an instance without churn as a control.) We expected lookup latency to increase over time as some finger nodes became unavailable and some lookups required multiple retries. We also expected latency to go down whenever SETUP was re-run, building new routing tables to reflect the current state of the network.

Figure 13 plots the lookup latency and retries for these experiments, and shows that Whānau is largely insensitive to modest node churn. The median latency is approximately a single network roundtrip within PlanetLab, and increases gradually as churn increases. As expected, the fraction of requests needing to be retried increased with time when node churn was present, but running SETUP restored it to the baseline.

While this experiment's scale is too small to test Whānau's asymptotic behavior, it demonstrates two points: (1) Whānau functions on PlanetLab, and (2) Whānau's simple approach for maintaining routing tables is sufficient to handle reasonable levels of churn.

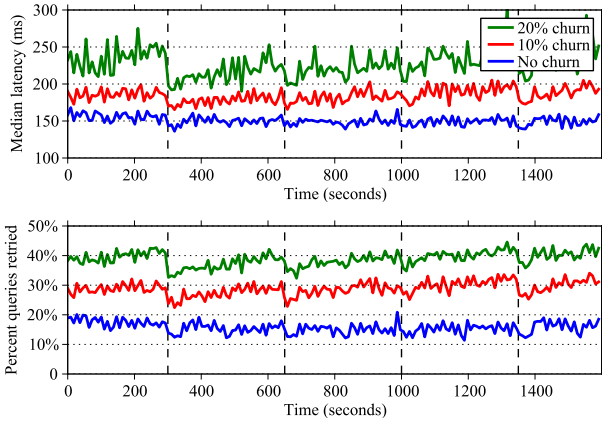


Figure 13: Lookup latency and fraction of lookups which required retries on PlanetLab under various levels of node churn. Vertical lines indicate when SETUP installed new routing tables. Under churn, the retry frequency slowly increases until SETUP runs again, at which point it reverts to the baseline.

10 Discussion

This section discusses some engineering details and suggests some improvements that we plan to explore in future work.

Systolic mixing process. Most of Whānau's bandwidth is used to explore random walks. Therefore, it makes sense to optimize this part of the protocol. Using a recursive or iterative RPC to compute a random walk, as suggested by Figure 4, is not very efficient: it uses w messages per random node returned.

A better approach, implemented in our PlanetLab experiment, is to batch-compute r walks at once. Suppose that every node maintains a pool of r addresses of other nodes; the pools start out containing r copies of the node's own address. At each time step, each node randomly shuffles its pool and divides it equally amongst its social neighbors. For the next time step, the node combines the messages it received from each of its neighbors to create a new pool, and repeats. After w such mixing steps, each node's pool is a randomly shuffled assortment of addresses. If r is sufficiently large, this process approximates sending out r random walks from each node.

Very many or very few keys per node. The protocol described in this paper handles $1 \lesssim k \lesssim m$ well, where k is the number of keys per honest node. The extreme cases outside this range will require tweaks to Whānau to handle them. Consider the case $k > m$. Any DHT requires at least $k = \Omega(m)$ resources per node just to transmit and store the keys. This makes the task easier, since we could use $O(m)$ bandwidth to collect a nearly-complete list of all other honest nodes on each honest node. With such a list, the task of distributing successor records is a simple variation of consistent hashing [13].

The analysis in Section 7.1 breaks down for $k > m$: more than m calls to SUCCESSORS-SAMPLE will tend to have many repeats, and thus can't be treated as independent trials. To recover this property, we can treat each node as k/m virtual nodes, as we did with node degrees.

Now consider the other extreme: $k < 1$, i.e. only some nodes are storing key-value records in the system. The extreme limiting case is only a single honest node storing a key-value record into the system, i.e. $k = 1/m$. Whānau can be modified to handle the case $k < 1$ by adopting the systolic mixing process described above and omitting empty random walks. This reduces to flooding in the extreme case, and smoothly adapts to larger k.

Handling key churn. It is clear that more bandwidth usage can be traded off against responsiveness to churn: for example, running SETUP twice as often will result in half the latency from key insertion to key visibility. Using the observation that the DHT capacity scales with the table size squared, we can improve this bandwidth-latency tradeoff. Consider running SETUP every T seconds with R resources, yielding a capacity of $K = O(R^2)$ keys. Compare with this alternative: run SETUP every $T/2$ seconds using $R/2$ resources, and save the last four instances of the routing tables. Each instance will have capacity $K/4$, but since we saved four instances, the total capacity remains the same. The total resource usage per unit time also remains the same, but the responsiveness to churn doubles, since SETUP runs twice as often.

This scaling trick might seem to be getting "something for nothing". Indeed, there is a price: the number of lookup messages required will increase with the number of saved instances. However, we believe it may be possible to extend Whānau so that multiple instances can be combined into a single larger routing table, saving both storage space and lookup time.

11 Summary

This paper presents the first efficient DHT routing protocol which is secure against powerful denial-of-service attacks from an adversary able to create unlimited pseudonyms. Whānau combines previous ideas — random walks on fast-mixing social networks — with the idea of layered identifiers. We have proved that lookups complete in constant time, and that the size of routing tables is only $O(\sqrt{km}\log km)$ entries per node for an aggregate system capacity of km keys. Simulations of an aggressive clustering attack, using social networks from Flickr, LiveJournal, YouTube, and DBLP, show that when the number of attack edges is less than 10% of the number of honest nodes and the routing table size is $\sqrt{m}$, most lookups succeed in only a few messages. Thus, the Whānau protocol performs similarly to insecure one-hop DHTs, but is strongly resistant to Sybil attacks.

Acknowledgements. Many thanks to Alan Mislove for providing invaluable social network datasets. We are grateful to the anonymous reviewers and to our shepherd, Timothy Roscoe, for their many helpful comments. This research was supported by the T-Party Project (a joint research program between MIT and Quanta Computer Inc., Taiwan) and by the NSF FIND program.

References

[1] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science,* 286(5439), 1999.
[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. *NSDI,* San Francisco, CA, Mar. 2004.
[3] N. Borisov. Computational Puzzles as Sybil Defenses. *Peer-to-Peer Computing,* Cambridge, UK, Sept. 2006.
[4] 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.
[5] A. Cheng and E. Friedman. Sybilproof Reputation Mechanisms. *Workshop on the Economics of Peer-to-Peer Systems,* Philadelphia, PA, Aug. 2005.
[6] F. Chung. *Spectral Graph Theory.* American Mathematical Society, 1997.
[7] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil Nodes Using Social Networks. *NDSS,* San Diego, CA, Feb. 2009.
[8] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. J. Anderson. Sybil-Resistant DHT Routing. *ESORICS,* 2005.
[9] J. R. Douceur. The Sybil Attack. *IPTPS,* Cambridge, MA, Mar. 2002.
[10] A. Gupta, B. Liskov, and R. Rodrigues. Efficient Routing for Peer-to-Peer Overlays. *NSDI,* San Francisco, CA, Mar. 2004.
[11] 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,* Berkeley, CA, Feb. 2003.
[12] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. *USITS,* Seattle, WA, Mar. 2003.
[13] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. *STOC,* El Paso, TX, May 1997.
[14] C. Lesniewski-Laas. A Sybil-Proof One-Hop DHT. *Workshop on Social Network Systems,* Glasgow, Scotland, April 2008.
[15] C. Lesniewski-Laas and M. F. Kaashoek. Whanaungatanga: Sybil-Proof Routing with Social Networks. MIT, Technical Report MIT-CSAIL-TR-2009-045, Sept. 2009.
[16] B.N. Levine, C. Shields, and N.B. Margolin. A Survey of Solutions to the Sybil Attack. University of Massachusetts Amherst, Amherst, MA, 2006.
[17] S. Marti, P. Ganesan, and H. Garcia-Molina. DHT Routing Using Social Links. *IPTPS,* La Jolla, CA, Feb. 2004.
[18] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. *IMC,* San Diego, CA, Oct. 2007.
[19] 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,* Cambridge, UK, Apr. 2004.
[20] R. Rodrigues and B. Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. MIT CSAIL, Technical Report TR/932, Dec. 2003.
[21] H. Rowaihy, W. Enck, P. McDaniel, and T. L. Porta. Limiting Sybil Attacks in Structured P2P Networks. *INFOCOM,* Anchorage, AK, May 2007.
[22] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach. Eclipse Attacks on Overlay Networks: Threats and Defenses. *INFOCOM,* Barcelona, Spain, Apr. 2006.
[23] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. *IPTPS,* Cambridge, MA, Mar. 2002.
[24] 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), 2003.
[25] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Rating. *NSDI,* Boston, MA, Apr. 2009.
[26] 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.
[27] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: Defending Against Sybil Attacks Via Social Networks. *SIGCOMM,* Pisa, Italy, Sept. 2006.
[28] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao. DSybil: Optimal Sybil-Resistance for Recommendation Systems. *IEEE Symposium on Security and Privacy,* Oakland, CA, May 2009.