

# Design and Applications of a Secure and Decentralized Distributed Hash Table

by

Christopher T. Lesniewski-Laas

S.B., Massachusetts Institute of Technology (2001)  
M.Eng., Massachusetts Institute of Technology (2003)

Submitted to the Department of Electrical Engineering and  
Computer Science  
in partial fulfillment of the requirements for the degree of

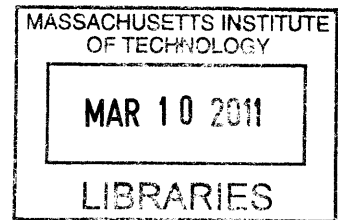
Doctor of Philosophy

at the


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2011

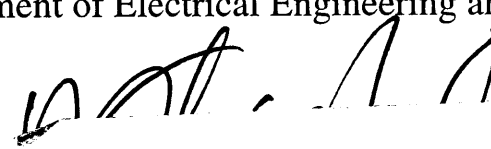
**ARCHIVES**




© Massachusetts Institute of Technology 2011. All rights reserved.

  
Author .....

Department of Electrical Engineering and Computer Science  
September 20, 2010

  
Certified by .....

M. Frans Kaashoek  
Professor of Computer Science and Engineering  
Thesis Supervisor

  
Accepted by .....

Professor Terry P. Orlando  
Chairman, Department Committee on Graduate Students



# Design and Applications of a Secure and Decentralized Distributed Hash Table

by

Christopher T. Lesniewski-Laas

Submitted to the Department of Electrical Engineering and Computer Science  
on September 20, 2010, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

## Abstract

Distributed Hash Tables (DHTs) are a powerful building block for highly scalable decentralized systems. They route requests over a structured overlay network to the node responsible for a given key. DHTs are subject to the well-known Sybil attack, in which an adversary creates many false identities in order to increase its influence and deny service to honest participants. Defending against this attack is challenging because (1) in an open network, creating many fake identities is cheap; (2) an attacker can subvert periodic routing table maintenance to increase its influence over time; and (3) specific keys can be targeted by clustering attacks. As a result, without centralized admission control, previously existing DHTs could not provide strong availability guarantees.

This dissertation describes Whānau, a novel DHT routing protocol which is both efficient and strongly resistant to the Sybil attack. Whānau solves this long-standing problem by using the social connections between users to build routing tables that enable Sybil-resistant one-hop lookups. The number of Sybils in the social network does not affect the protocol's performance, but links between honest users and Sybils do. With a social network of  $n$  well-connected honest nodes, Whānau provably tolerates up to  $\mathcal{O}(n/\log n)$  such "attack edges". This means that an attacker 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 techniques from structured DHTs to build routing tables that contain  $\mathcal{O}(\sqrt{n} \log n)$  entries per node. It introduces the idea of *layered identifiers* to counter clustering attacks, which have proven particularly challenging for previous DHTs to handle. Using the constructed tables, lookups provably take constant time.

Simulation results, using large-scale social network graphs from LiveJournal, Flickr, YouTube, and DBLP, confirm the analytic prediction that Whānau provides high availability in the face of powerful Sybil attacks. Experimental results using PlanetLab demonstrate that an implementation of the Whānau protocol can handle reasonable levels of churn.

Thesis Supervisor: M. Frans Kaashoek

Title: Professor of Computer Science and Engineering



*No man is an Iland, intire of it selfe; every man is a peece of the Continent, a part of the maine; if a Clod bee washed away by the Sea, Europe is the lesse, as well as if a Promontorie were, as well as if a Mannor of thy friends or of thine own were; any mans death diminishes me, because I am involved in Mankinde; And therefore never send to know for whom the bell tolls; It tolls for thee.*

John Donne, *Meditation XVII*



## Acknowledgments

I did not write this dissertation alone; it is the product of many minds. The project's earliest germ, the bootstrap graph model, grew from discussions with George Danezis, Ross Anderson, Emil Sit, Robert Morris, and M. Frans Kaashoek. Mike Kaminsky and Haifeng Yu shared key insights from SybilLimit; Alan Mislove provided invaluable social network datasets. The proofs benefited from careful attention paid by the ever-enthusiastic David Karger, and the model and protocol descriptions were much improved by comments from Barbara Liskov. Raymond Cheng, who developed the WhānauSIP implementation, helped me to transform half-articulated intuition into concrete specification.

I have been enriched by my collaboration with my advisor, M. Frans Kaashoek. His intellect, energy, and vision inspired me to tackle challenging problems, and his wisdom and encouragement made it possible to achieve ambitious goals. Thank you, Frans.

Robert Morris has always been generous with his penetrating insight and keen wit, for which I am grateful. Every idea, artifact, presentation, paper, or person that makes it past Robert is improved by the encounter.

I owe a deep debt to my fellow-travelers in MIT CSAIL. Bryan Ford, Sean Rhea, Anthony Joseph, and Justin Paluska were my co-conspirators on the UIA project, a handy vehicle for whatever peer-to-peer mischief we wanted to perpetrate. To my officemates, Alex Yip, Jinyang Li, Athicha Muthitacharoen, Thomer Gil, Shuo Deng, Jayashree Subramanian, and Ramesh Chandra, thanks for the uncountable and wide-ranging discussions. I've learned an immense amount from my brilliant PDOS colleagues Russ Cox, Emil Sit, Frank Dabek, Mike Walfish, Max Krohn, Jeremy Stribling, Dan Aguayo, John Bicket, Sanjit Biswas, Michael Freedman, Emmett Witchel, David Mazières, Eddie Kohler, John Jannotti, Kevin Fu, Dave Andersen, Petar Maymounkov, Xi Wang, Austin Clements, Neha Narula, Alex Pesterev, Silas Boyd-Wickizer, and Nikolai Zeldovich. Neena Lyall kept the wheels turning smoothly. When I ventured out for coffee, I would sometimes run into Rodrigo Rodrigues, James Cowling, Dan Myers, Yang Zhang, Hariharan Rahul, Kyle Jamieson, Max Van Kleek, Jerry Saltzer, John Guttag, Daniel Jackson, Dina Katabi, Hari Balakrishnan, or Sam Madden. I always left with something new and interesting to think about.

Jacob Strauss entered the MIT undergraduate program at the same time as I did, chose the same major, entered the same MIT graduate program, and joined the same research group in CSAIL. Over the years, we have been classmates, teammates, flatmates, officemates, partners in crime, and co-founders of the UIA project, and I have always relied upon his generosity and good sense. I am very lucky to have Jacob as a friend and colleague.

To all my friends and family, and especially to Jennifer Tu and to my parents Marek and Elizabeth Lesniewski-Laas, thank you for your love, support, and encouragement.

I gratefully acknowledge financial support from MIT, Quanta Computer, Nokia, and the NSF's Project IRIS.





# Contents

<b>1</b>	<b>Introduction</b>	<b>17</b>
1.1	Distributed Hash Tables . . . . .	18
1.2	Denial-of-service attacks on DHTs . . . . .	20
1.2.1	Denying service through brute force . . . . .	20
1.2.2	The Eclipse attack . . . . .	21
1.2.3	Clustering attacks . . . . .	21
1.3	The Whānau Sybil-proof DHT . . . . .	22
1.3.1	Implementations . . . . .	23
1.4	Contributions . . . . .	23
1.5	Limitations . . . . .	24
1.6	Structure of this dissertation . . . . .	25
<b>2</b>	<b>Related work</b>	<b>27</b>
2.1	Distributed Hash Tables . . . . .	27
2.2	Defenses against attacks on DHT overlay routing . . . . .	28
2.3	Social-network-based defenses against the Sybil attack . . . . .	29
2.4	Applying social-network-based Sybil defenses to DHTs . . . . .	30
2.5	Analysis and criticism of social-network Sybil defenses . . . . .	31
<b>3</b>	<b>Assumptions and goals</b>	<b>33</b>
3.1	DHT-to-application interface . . . . .	33
3.1.1	Example application: rendezvous service . . . . .	35
3.1.2	Division of labor: integrity versus availability . . . . .	36
3.2	Social network model . . . . .	37
3.2.1	Social connections: physical rendezvous, social network services . . . . .	38
3.2.2	First assumption: the sparse cut . . . . .	39
3.2.3	Second assumption: a fast-mixing social network . . . . .	40
3.2.4	Why is this a reasonable model? . . . . .	41
3.3	Sampling the social network using random walks . . . . .	42
3.3.1	Winners and losers . . . . .	42
3.3.2	Balancing load using virtual nodes . . . . .	43
3.4	Communications network model and adversary model . . . . .	43
3.5	The main security property . . . . .	44
3.6	Performance . . . . .	45

<b>4</b>	<b>An unstructured DHT protocol</b>	<b>47</b>
4.1	Theseus’s algorithm . . . . .	47
4.2	Introducing caches to reduce lookup cost . . . . .	48
<b>5</b>	<b>The Whānau protocol</b>	<b>51</b>
5.1	Overview of Whānau . . . . .	51
5.1.1	Ring structure . . . . .	51
5.1.2	Layered identifiers . . . . .	52
5.1.3	Handling failures and churn . . . . .	54
5.2	Setup . . . . .	55
5.2.1	Protocol operation . . . . .	56
5.2.2	Pseudocode . . . . .	56
5.2.3	Informal correctness argument . . . . .	57
5.3	Lookup . . . . .	60
5.3.1	Protocol operation . . . . .	60
5.3.2	Pseudocode . . . . .	61
5.3.3	Informal correctness argument . . . . .	61
<b>6</b>	<b>Analysis of Whānau’s performance</b>	<b>63</b>
6.1	Social network model . . . . .	63
6.1.1	The sparse cut and escape probability . . . . .	63
6.1.2	Variation from the uniform distribution . . . . .	65
6.1.3	Quantifying random-walk sample quality . . . . .	66
6.2	Winner key tables are correct . . . . .	67
6.2.1	Preliminary definitions . . . . .	67
6.2.2	SLICE-SAMPLE returns good sample records . . . . .	68
6.2.3	Aggregating $r_k$ samples yields complete key tables . . . . .	69
6.3	Layer-zero IDs are evenly distributed . . . . .	70
6.4	Layers are immune to clustering . . . . .	70
6.4.1	Special case: relatively weak Sybil attacks . . . . .	72
6.5	Main result: lookup is fast . . . . .	72
6.6	Routing tables are small . . . . .	74
<b>7</b>	<b>Implementations</b>	<b>75</b>
7.1	Distributed implementation . . . . .	75
7.2	In-memory simulator . . . . .	76
<b>8</b>	<b>Simulations and experiments</b>	<b>79</b>
8.1	Real-world social networks fit the model’s assumptions . . . . .	80
8.1.1	Large-scale data sets . . . . .	80
8.1.2	Measuring social network mixing properties . . . . .	81
8.1.3	Measuring escape probabilities . . . . .	83
8.2	Performance under clustering attack . . . . .	84
8.3	Impact of layered IDs . . . . .	87
8.3.1	Layers are necessary for attack resistance . . . . .	87

8.3.2	Optimal number of layers increases with table size . . . . .	89
8.4	Performance versus number of nodes . . . . .	90
8.5	Performance on PlanetLab with node churn . . . . .	91
<b>9</b>	<b>Extending Whānau</b>	<b>93</b>
9.1	Taking advantage of network locality . . . . .	93
9.2	Efficient random walks using a systolic mixing process . . . . .	94
9.2.1	Using the systolic process in Whānau . . . . .	95
9.2.2	Using symmetry to protect against denial-of-service . . . . .	95
9.3	Supporting very many or very few keys per node . . . . .	96
9.3.1	Many keys per node . . . . .	96
9.3.2	Fewer keys than nodes . . . . .	96
9.4	Routing over the social network . . . . .	97
9.4.1	Recursive routing . . . . .	97
9.4.2	Tunneling over social links . . . . .	97
9.5	Preserving friend list privacy . . . . .	98
9.6	Handling large, durable data blocks . . . . .	99
9.7	Handling dynamic record updates and insertions . . . . .	99
9.7.1	Protocols for updating values . . . . .	100
9.7.2	Challenges to inserting keys . . . . .	100
9.7.3	Trading bandwidth against key insert latency . . . . .	101
<b>10</b>	<b>Conclusion</b>	<b>103</b>



# List of Figures

3-1	Overview of the Whānau protocol. . . . .	34
3-2	The social network model. A sparse cut separates honest and Sybil nodes. . .	39
4-1	Theseus searching the labyrinth for the Minotaur. . . . .	47
5-1	Layered identifiers. . . . .	54
5-2	SETUP procedure to build structured routing tables. . . . .	58
5-3	LOOKUP procedure to retrieve a record by key. . . . .	62
8-1	Mixing properties of social graphs. . . . .	82
8-2	Escape probability on the Flickr network. . . . .	84
8-3	Number of messages used by LOOKUP. . . . .	85
8-4	Heat map and contours of the number of messages used by LOOKUP. . . .	86
8-5	Optimal layers versus attacker power. . . . .	88
8-6	Optimal layers versus resource budget. . . . .	89
8-7	Number of messages used by LOOKUP, versus system size and table size. .	91
8-8	Lookup latency and retries on PlanetLab under churn. . . . .	92



# List of Tables

3.1	Standard DHT API versus Whānau’s API. . . . .	33
3.2	Summary of social network parameters used in the analysis. . . . .	44
5.1	Routing tables built by SETUP. . . . .	56
5.2	SETUP parameters. . . . .	57
8.1	Properties of the input data sets. . . . .	81





# Chapter 1

## Introduction

Peer-to-peer systems can tie together millions of edge hosts to provide efficient and fault-tolerant infrastructure services, such as search, storage, caching, load-balancing, routing, and rendezvous [37, 38, 45, 53–56, 66, 67, 72, 78, 89, 100, 104, 109, 111, 120]. However, an *open* P2P system, which anyone on the Internet can join, is vulnerable to the well-known *Sybil attack* [48]. In this attack, a malicious adversary creates a large number of pseudonyms (“Sybils”) and joins the system many times. This technique gives the adversary disproportionate influence, which it can use to degrade or deny the legitimate users’ access to the service.

To locate resources such as files, users, or servers, a P2P system sends routing and discovery queries over a network of links between the nodes. These *overlay networks* can be either *unstructured* or *structured*. Unstructured P2P systems find resources by flooding queries over the overlay links at random. On the other hand, structured P2P systems create overlay links between carefully chosen nodes so that queries can be routed directly to the intended recipients. As a consequence, structured overlays use network resources much more efficiently than unstructured overlays and are better at “needle-in-a-haystack” type queries. However, structured overlays are also more vulnerable to Sybil attacks than unstructured overlays are.

Many existing P2P systems need to use structured overlay networks for efficiency. Up until now, such systems either ignored Sybil attacks or assumed a central authority to vet participants. Neither approach is completely satisfying: the first gives up any strong availability guarantees, and the second does not reap all the benefits of a fully peer-to-peer approach. What’s more, assuming the existence of a central authority evades the question of *how* the authority should distinguish honest nodes from Sybils. To do this, it must have some out-of-band information about the participants [48]. A certifying authority is not clairvoyant by virtue of being centralized; it must use an online proxy for real identity in order to limit the fraction of Sybil identities. Proposed proxies have included government-issued certificates, IP addresses, computational puzzles, bandwidth, CAPTCHAs, and social network connections.

This dissertation presents a structured and efficient P2P overlay network that is strongly Sybil-resistant and requires no centralized gatekeeper. It achieves this by assuming a well-connected social network between the honest P2P nodes: as long as there are more links amongst honest nodes than between honest and Sybil nodes, the system ensures that routing

queries will succeed quickly. There exist well-known replication and encryption techniques for handling random faults, churn, and security issues such as integrity and confidentiality [41,45,57,65,81,100,106,107,110]. Thus, Sybil-proof availability provides the missing link to a truly secure and reliable structured overlay. This secure overlay network's performance properties are similar to previous (insecure) overlays, and close to the theoretical optimum.

Addressing Sybil attacks removes one of the main remaining barriers to large-scale decentralized peer-to-peer infrastructure and opens up many new possibilities. For example:

- Highly available federated data storage systems, using individuals' computer resources (instead of centralized services) to provide a "cloud" of backend storage for filesystems or databases.
- Internet routing protocols that are "hijack-proof", so that malicious participants cannot redirect traffic away from web sites (as Pakistan accidentally did to YouTube in 2008 [20] and as has happened many times before [34]).
- Vehicular Ad-Hoc Networks (VANETs) which cannot be "jammed" by malicious routing messages.
- Reputation and recommendation engines which do not permit bad reviews to be "whitewashed" [50,51].
- Publication and dissemination systems that are very difficult to censor [116].

This dissertation promotes the vision of a collaborative Internet in which peers cooperate to provide the infrastructure services they use. In some cases, this distributed model may present an alternative to the centralized provider model. In others, it may be complementary, enabling services to be provided by a heterogeneous federation of large and small corporations, government entities, and individuals. This approach can take the best features of each, improving reliability, efficiency, and flexibility by increasing diversity.

## 1.1 Distributed Hash Tables

Distributed Hash Table (DHT) protocols are a basic building block of structured peer-to-peer systems. A DHT maps a key to the node responsible for that key, and provides an overlay routing mechanism to send messages to that node. If each node maintains a local table associating its keys with values, the system can provide an insert/lookup/delete API similar to a hash table data structure. In addition to message routing (key lookup), DHT protocols normally provide interfaces for nodes to join and leave the set of participants, and they maintain redundant state in order to tolerate disruptions caused by failures and churn. Because DHTs provide fast routing and automatic load balancing without any centralized bottlenecks, they have been applied to many distributed systems problems, including caching [53,55,67], filesystems and file sharing [45,78,89,111], databases [66,72], version control [120], publish-subscribe [37,109], multicast streaming [37,38,56], telephony

and IM [104], anonymous communications [54], and reputation and recommendation engines [70].

As a simple example, a DHT can be used as a rendezvous service for a telephony/instant-messaging application. When a client comes online, it inserts into the DHT a key-value pair which maps its username to its current IP address. To send an IM or initiate a voice call, the client can look up the target username in the DHT to retrieve that user’s current address. The DHT service can be provided either by a network of volunteer peers or by the clients themselves.

Most DHTs use consistent hashing [71] (or some closely related scheme) to map ranges of keys to responsible nodes. In consistent hashing, a hash function maps keys onto  $b$ -bit integers, and each node chooses a random  $b$ -bit identifier when it joins the DHT. A node is responsible for all the keys between its ID and its predecessor’s ID. This technique partitions the set of keys evenly across the nodes and minimizes data movement due to node churn.

In the earliest DHTs, each node maintained a list of all the other nodes in the system, and used consistent hashing directly to route messages to keys. However, the modern DHT literature focuses on cases where the number of nodes  $n$  is very large (e.g. millions of nodes), which makes the storage and bandwidth costs of maintaining complete tables impractical. Instead, each node maintains a partial routing table, meaning that not all messages can be sent directly to their destination; sometimes messages must be routed via one or more intermediate hops. Each hop takes the message to a node whose identifier is closer to the target key, eventually reaching the destination node.

If each node has a local table of size  $r$ , and routes are  $h$  hops long, then messages can fan-out to reach at most  $r^h$  nodes. There is a fundamental trade-off between table size and routing latency: to route from any peer to any other peer within  $h$  hops, the tables must have at least  $r = n^{1/h}$  entries per node. DHT protocols can be broadly categorized by where they fall on this curve:

- $r = n$ : complete-graph DHTs [62, 65, 72].
- $r = \mathcal{O}(\sqrt{n})$ : one-hop DHTs [62, 63].<sup>1</sup>
- $r = \mathcal{O}(\log n)$ : log  $n$ -hop DHTs [80, 101, 110].

Most published DHTs are log  $n$ -hop designs which trade higher latency for very small tables. A few DHTs (e.g., Accordion [76]) can adapt to arbitrary table sizes and achieve any point on the trade-off curve. This dissertation describes a one-hop DHT.

---

<sup>1</sup>Note that [62] presents two DHT protocols, which the authors call “one-hop” and “two-hop”. Using this dissertation’s terminology (which is consistent with [63]), they would be called a complete-graph DHT and a one-hop DHT, respectively.

## 1.2 Denial-of-service attacks on DHTs

A secure DHT design must satisfy three properties: <sup>2</sup>

- **Confidentiality:** data stored in the DHT is only readable by authorized parties.
- **Integrity:** data stored in the DHT can only be modified by authorized parties.
- **Availability:** data stored in the DHT can always be retrieved and updated using a reasonable amount of time and resources.

Confidentiality and integrity can be provided by well-understood techniques, such as encryption, signatures, and self-certifying keys [57, 58, 69, 77, 81]. However, because of the Sybil attack and related denial-of-service attacks, guaranteeing availability for an open DHT is a challenging and long-standing problem.

In a Sybil attack, a malicious actor on the Internet creates a large number of false identities and floods the existing DHT nodes with join requests. If each identity has a different IP address, it is very difficult to distinguish such virtual nodes from “real” nodes. Once the adversary has infiltrated the DHT under many pseudonyms and “outvotes” the honest identities, he can perpetrate various sorts of mischief, ranging from blanket denial-of-service attacks to sophisticated, targeted attacks on specific keys. For example, in the telephony rendezvous system outlined in Section 1.1, an attacker could prevent one or more users from receiving incoming calls by joining the DHT, becoming responsible for the targeted users’ IP address records, and returning bogus results to any queries for those records.

On the present-day Internet, it is cheap and easy to create large numbers of apparently-unique identities, given access to large IP subnets, botnets-for-hire, or IPv6 address blocks. <sup>3</sup> Sybil attacks are of particular concern because they inflict much more damage than simple ping-flooding or request-flooding attacks [19] for the same investment of the attacker’s bandwidth and CPU resources. The following sections sketch Sybil attacks of increasing sophistication and lay out the challenges that must be addressed to defend against them.

### 1.2.1 Denying service through brute force

If the adversary can inexpensively create an unlimited number of Sybil identities, then a simple brute-force attack can bring down any open DHT. Suppose the adversary adds  $99n$  Sybil nodes to an  $n$ -node DHT using the normal join protocol. Then, 99% of the entries in honest nodes’ routing tables will be pointers to Sybils, and almost all honest query messages will be routed via a Sybil node. These queries will never reach their target keys: the Sybil nodes can return bogus replies, respond with an error message (*i.e.*, “no such key”), delay the query, or simply drop the messages entirely [105].

This blanket denial-of-service attack leaves the honest nodes with no recourse other than to retry queries over and over, with a tiny probability of reaching any other honest

---

<sup>2</sup>Some applications (*e.g.*, dissemination or caching) may not require confidentiality, but all applications require integrity and availability.

<sup>3</sup>The widespread use of NAT also discourages limiting identities to one-per-IP-address, since that unfairly cuts out a large fraction of the Internet-using population.

node at all. Honest nodes can blacklist nodes giving incorrect responses, but this strategy only helps if new Sybils are added more slowly than they are blacklisted. It is also possible that honest nodes may be incorrectly blacklisted due to transient failures. Moreover, an unsophisticated blacklisting mechanism could be manipulated by the adversary to “frame” honest nodes by feeding them bad information.

Ultimately, a brute-force Sybil attack can only be prevented by limiting the fraction of Sybil identities in the DHT’s routing tables, and any mechanism to accomplish this must use some information from outside the DHT [48].

## 1.2.2 The Eclipse attack

Even if the fraction of incoming Sybil nodes is bounded by some method, it is possible for an attacker to use the dynamics of DHT churn and stabilisation to increase its influence over time. DHT nodes constantly join and leave, and new nodes initialize their routing and key tables by requesting subsets of established nodes’ tables. Also, DHTs periodically execute a stabilization protocol to repair failures by propagating routing table entries.

In the *Eclipse attack* [103], the adversary exploits these join and stabilization protocols to expand a small foothold into a large attack surface. Suppose that there are a small number of Sybil nodes already in the DHT (*i.e.*, present in honest nodes’ routing tables). Sometimes, honest nodes will try to initialize or repair their routing tables by querying Sybil nodes. Instead of replying honestly, the Sybils return plausible routing table entries which point only to other Sybil identities, increasing the overall fraction of Sybil entries in honest routing tables. What’s more, the frequency of queries to Sybils increases with the fraction of Sybil entries, amplifying the attack’s power. Over time, an attack starting from a small base can snowball until most of the honest nodes’ routing table entries point to Sybil nodes. At this point, the attacker essentially controls the DHT and can deny service by corrupting, dropping, or delaying messages, as above.

## 1.2.3 Clustering attacks

As the previous sections showed, to protect against blanket denial-of-service attacks, Sybil identities must be limited to a small fraction of the honest nodes’ routing table entries. However, this does not prevent *clustering attacks*, which deny honest nodes access to a particular key or node. This type of attack requires more than one identity to perform, but since it is targeted at a small part of the DHT, it requires far fewer Sybil identities than a brute-force attack on the entire DHT.

Clustering attacks take advantage of the structure of DHT routing tables. Typically, each node is responsible for storing the keys closest to its node ID, and each routing hop carries a query message closer to its destination key. Sybil nodes can exploit this by choosing all of their IDs to fall closer to the given target key than any honest node’s ID, making the Sybil nodes appear to be responsible for that key. If the Sybil nodes infiltrate sufficiently many honest nodes’ routing tables, then almost all queries for the target key will be forwarded to them.

In another variation, the attacker might insert many bogus keys into the DHT near the

targeted key. If the responsible node’s key table can’t handle this load spike, it may be forced to overflow and discard genuine keys.

Previous work on DHT security addressed ID clustering attacks by assuming that newly joined nodes would be assigned externally generated random IDs, instead of generating their own IDs [36, 97, 103]. In addition to requiring a central ID-issuing authority, this approach can be circumvented by repeatedly joining and leaving the DHT. This guess-and-check strategy enables the attacker to generate arbitrarily clustered IDs for its Sybil nodes.

DHTs add structure to the overlay network in order to perform efficient lookups. However, the more finely-tuned the routing protocol, the more susceptible it tends to be to clustering attacks. For example, consistent hashing alone is subject to key clustering attacks; adding multi-hop routing makes the system more vulnerable to ID clustering attacks. Advanced features such as network locality [44, 62] and jurisdictional locality [64] open up additional avenues of attack. Because many techniques for efficiency can be exploited by a malicious adversary, denial-of-service attacks are a particularly stubborn and challenging problem in structured overlay security. 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.

### 1.3 The Whānau Sybil-proof DHT

This dissertation presents a new and effective approach for guaranteeing DHT availability. **Whānau**, a secure and efficient one-hop DHT protocol, concretely illustrates this approach.<sup>4</sup> Like other DHTs, Whānau can be used as a key-value store or as a routing mechanism to send messages from any node to any other. Its performance is similar to previous (insecure) one-hop DHTs:

- Routing a lookup request uses a constant number of messages, with a latency of only one network round-trip if query messages are sent in parallel.
- Building routing tables consumes  $\mathcal{O}(\sqrt{kn} \log kn)$  bandwidth, CPU, and storage per node (to store up to  $kn$  keys in the whole system).

Whānau relies on existing techniques to provide confidentiality and integrity. It assumes that an honest node querying a key will recognize the corresponding value: for example, by checking a digital signature included with the returned data block. Whānau guarantees availability by applying several new techniques, in order to address the challenges described in Section 1.2:

---

<sup>4</sup>*Whānau*, pronounced “far-no”, is a Māori word meaning “family”, and particularly connoting the wider kinship of an extended family [88]. Etymologically, *whānau*’s root meaning is the verb “to be born”, and the main derived meaning is “a group descended from shared ancestors”; modern usage sometimes includes the metaphorical *kaupapa whānau*, which is a group connected by a common bond other than descent [25, 88, 117]. The Māori word *whānau* is cognate with the Hawai’ian word *’ohana*.

This dissertation’s use of the name Whānau acknowledges the critical role that meaningful social connections play in defending the system against malicious attacks. See the references [11, 16, 25, 43, 117] for more information about the original Māori concept in historical and modern contexts.

- To protect against brute-force Sybil attacks, Whānau uses information from an external social network to filter out most of the bogus identities. While Whānau is oblivious to the number of Sybil identities in the social network, its performance degrades if there are very many connections between honest and Sybil identities.
- To protect against the Eclipse attack, the algorithm to build routing tables is “stateless”: instead of bootstrapping new nodes from existing routing tables, Whānau periodically discards its routing tables and rebuilds them all from scratch.
- To protect against ID clustering attacks, Whānau introduces a technique called *layered identifiers*. Whānau also prevents key clustering attacks through carefully balanced partitioning of the key tables over the honest nodes.

To rule out the possibility of denial-of-service attacks other than those described in Section 1.2, this dissertation includes a proof of Whānau’s correctness and efficiency under reasonable assumptions.

### 1.3.1 Implementations

The Whānau protocol has three existing implementations:

- The reference implementation is 1,104 lines of Python code, and has been tested with up to 4,000 virtual nodes running on 400 physical PlanetLab nodes [31].
- WhānauSIP [40] is 4,280 lines of Java and integrates the Session Initiation Protocol [98]. It demonstrates how Whānau can be used to provide secure rendezvous for Voice-over-IP telephones.
- Because the PlanetLab testbed does not support large numbers of nodes, Whānau is also implemented in a large-scale, in-memory simulation, supporting up to 5 million virtual nodes. For efficiency, this simulation abstracts most of the details of real-world computer networks; it measures Whānau’s performance under attack by counting simulated messages and table entries. The simulator is implemented as 3,234 lines of C++ code using the STL and Boost [3, 108].

## 1.4 Contributions

This dissertation aims to demonstrate that peer-to-peer structured overlay networks do not require centralized admission control to resist denial-of-service attacks. In support of this thesis, it presents:

- **Whānau**, a structured Sybil-proof DHT protocol employing a combination of new and existing techniques to guarantee availability.
- **Layered identifiers**, a new technique to combat ID clustering attacks.
- A **proof** of Whānau’s correctness under reasonable assumptions.

- A **reference implementation** of Whānau.
- A high-performance **simulator** using large-scale graph data extracted from real-world social networks.
- Detailed simulated **experiments** confirming Whānau’s theoretically-predicted performance under the most challenging known attacks.

Along the way, it also demonstrates how to use some of these techniques in unstructured overlay networks.

## 1.5 Limitations

The core Whānau protocol (Chapter 5) has several limitations as compared with previous DHTs. By adding additional complexity, some of these limitations can be mitigated; nevertheless, Whānau may be inappropriate for some applications (*e.g.*, full-text search).

**Assumptions.** The honest nodes must be connected by a social network with high conductivity, and there must be a sparse cut separating the honest identities from the Sybil identities. If the social network does not have these properties, then Whānau will not tolerate Sybil attacks.

Furthermore, the application must provide Whānau with a way to distinguish whether the records returned by DHT lookups are authentic or not. This is necessary to prevent attackers from inserting bogus values under the application’s keys.

**Overhead.** A Whānau implementation pays for its resistance to denial-of-service attacks by consuming more resources than an insecure DHT when not under attack. Nodes must encrypt and authenticate their routing messages to prevent impersonation attacks, and node addresses stored in routing tables must be accompanied by cryptographic keys for this purpose. Layered identifiers, which protect Whānau against ID clustering attacks, impose an additional  $\mathcal{O}(\log n)$  overhead factor.

Whānau obtains its resiliency through aggressive replication. If each node inserts  $\mathcal{O}(1)$  key-value records, then Whānau replicates every record to  $\mathcal{O}(\sqrt{n} \log n)$  different nodes. This could be expensive for large records; however, the overhead can be mostly eliminated by adding a layer of indirection (Section 9.6).

In order to prevent resource exhaustion, each node can insert a fixed number of records into the DHT. If some nodes do not use their entire quota, the unused resources are wasted. An application could layer a quota sharing or trading mechanism on top of Whānau to reduce this inefficiency, but Whānau does not provide such a protocol.

**Static routing tables.** The Whānau API does not have dynamic JOIN and INSERT operations: once constructed, Whānau’s routing tables are a static snapshot of the social network and the set of records stored in the DHT. To incorporate changes, nodes must periodically discard and rebuild their routing tables. This means that keys can only be inserted at discrete intervals, with a frequency defined by the application’s needs.



In general, this trade-off between freshness and efficiency is not unique to Whānau: for example, DNS tolerates stale records up to several hours old in order to improve caching performance. However, other DHTs do not impose this compromise on (non-caching) applications. An application using Whānau must be able to adapt to the delay between inserting a key into the DHT, and that key becoming visible to other nodes. Although key insertion is always delayed, Whānau can be extended to support efficient and dynamic updates to the *values* associated with keys already stored in the DHT. Thus, mutable records are supported (see Section 9.7).

Periodically rebuilding routing tables of size  $\mathcal{O}(\sqrt{kn} \log kn)$  per node also adds to Whānau’s overhead. Frequent rebuilding is costly; however, if the routing tables are permitted to drift out of sync with the state of the social network, then Whānau’s load balance will become progressively more skewed and the available resources less effectively utilized. Fortunately, measurements on social networks have shown that they turn over on a timescale of months [82, 83]; in most cases, applications will choose to rebuild tables more often than required by social network changes.

## 1.6 Structure of this dissertation

The rest of this document is structured as follows:

Chapter 2 reviews the existing body of research on peer-to-peer structured overlay networks and the Sybil attack, and traces the development of Sybil defenses based on social networks.

Chapter 3 sets out the goals of this work in both formal and informal language. It states assumptions about the social network and about the adversary model, leading to a precise definition of a “Sybil-proof distributed hash table”.

Chapter 4 briefly illustrates how a social network can be used in a simple, unstructured DHT protocol which is Sybil-proof, but not efficient.

Chapter 5 describes the Whānau structured DHT protocol. It outlines the design of Whānau’s routing tables, and introduces the new concept of layered identifiers. It then gives detailed distributed protocols to build structured routing tables and to route queries to keys using these tables.

Chapter 6 formally proves the correctness and good performance of the Whānau routing protocol.

Chapter 7 describes the reference implementation on PlanetLab and the in-memory protocol simulator.

Using these implementations, Chapter 8 confirms the theoretically-predicted properties through simulations on social network graphs from popular Internet services, and experimentally investigates the protocol’s reaction to churn on PlanetLab.

Chapter 9 discusses various extensions to the protocol to adapt Whānau to a variety of real-world situations.

Chapter 10 concludes the dissertation.



# Chapter 2

## Related work

This chapter reviews the history of Distributed Hash Tables (DHTs) and the Sybil attack. It covers a sample of the most relevant DHT security research; additional references can be found in Levine, Shields, and Margolin's 2006 survey [75].

Research on structured overlay networks grew out of the popularity of peer-to-peer file sharing systems in the early 2000s, and grew to encompass many distributed applications. Shortly after DHTs were invented, the Sybil attack was recognized as an important challenge, spawning several lines of research exploring possible solutions. In 2005, the popularity of online social network services prompted researchers to examine social networks as a potential Sybil defense, and the following years brought successive refinements to this approach. Recent work has built on social networks for recommendations, spam filtering, data backup, and DHT routing.

### 2.1 Distributed Hash Tables

Unstructured overlay networks which propagate messages using flooding, such as Usenet and IRC, were among the earliest Internet protocols. In the early 2000s, the sudden popularity of peer-to-peer file sharing services such as Napster and Gnutella [96] created renewed research interest in decentralized overlays. This rapidly led to the invention of the first structured distributed hash tables: Chord, Pastry, Tapestry, and CAN [94, 100, 110, 124]. These DHT designs adopted key partitioning techniques similar to consistent hashing [71] in order to tolerate node churn, and all offered multi-hop routing using compact tables. Efficient one-hop DHTs [62, 63] followed the next year.

**DHT applications.** Researchers immediately applied distributed hash tables to a variety of distributed applications such as file systems (CFS [45], PAST [100], Ivy [89]), Web caches (Squirrel [67]), databases (PIER [66]), multicast (SCRIBE [37], SplitStream [38]), mixnets (Tarzan [54]), and reputation engines (EigenTrust [70]). These early applications were refined by later systems such as WheelFS [111], Coral [53, 55], Pastwatch [120], and Oasis [56].

In recent years, DHTs have increasingly been adapted and deployed for production use in industry. For example, Amazon developed the Dynamo DHT [65], and Facebook

developed Cassandra [72], which is also used by Twitter, Digg, Reddit, Rackspace, Cisco WebEx, and others. Microsoft Windows Vista implemented a DHT-based “Peer Name Resolution Protocol” [17, 18, 24] for rendezvous, and Windows 7 includes a newer “Distributed Routing Table” protocol [23].

BitTorrent’s new “trackerless” distributed file sharing mechanism [78] is a prominent application of DHTs in a decentralized context. This system is based on the Kademia DHT protocol [80] and implemented by the reference client and by the popular Vuze/Azureus client [14].

**DHT security.** The earliest DHT publications recognized the need to protect the integrity and confidentiality of stored records using both standard cryptographic primitives and specialized techniques such as SFS’s self-certifying identifiers [57, 81]. Contemporary systems such as Plutus [69] and SUNDR [77] demonstrated how to store data on untrusted nodes while guaranteeing confidentiality, integrity, and freshness.

In early 2002, Sit and Morris [105] first catalogued attacks specifically targeted against DHT routing protocols. At the same time, Douceur’s *The Sybil Attack* [48] pointed out that an adversary controlling a large number of identities could interfere with a DHT’s correct operation, and first proposed addressing this problem using a central authority for admission control. Since then, many publications have identified additional attacks and proposed a wide variety of defenses.

The Vanish system for “disintegrating data” [61] depends on an underlying DHT, either Vuze [14] or OpenDHT [95]. This system was successfully attacked using a variant of the Sybil attack [119]; the Vanish authors responded with a set of recommendations for security applications on DHTs [60].

## 2.2 Defenses against attacks on DHT overlay routing

Douceur’s and Sit and Morris’s papers on DHT routing security were published in March 2002. Later the same year, Castro *et al.* [36] proposed the first secure DHT routing protocol, based on two principles: lookup queries would be forwarded using multiple redundant paths, and a node was only permitted to forward a query to a constrained set of finger IDs. This protocol was shown to route correctly as long as at least 75% of the nodes were honest; an attacker controlling more than 25% of the nodes could break the system. Castro *et al.* also recognized the possibility of clustering attacks if malicious nodes could choose their own IDs; for this reason, their protocol (like many later protocols) assumed that the central certificate authority assigned random identities to nodes.

The large body of subsequent research on DHT security has sought to improve this result in three directions:

1. By limiting the number of malicious nodes participating in the system.
2. By increasing the fraction of malicious nodes required to break the system’s security.
3. By changing the model so that some resource other than identities is required to break the system’s security.

Most work has focused on the first and second categories; this dissertation falls into the third category.

The Rosebud [97] DHT first applied the techniques of Byzantine fault tolerance (BFT) to overlay routing. In Rosebud, a subset of nodes forms a BFT cluster to provide a logically-centralized *configuration service* responsible for controlling admission and assigning IDs to nodes.

Singh *et al.* [103] observed that Castro *et al.*'s approach limits the DHT's ability to take advantage of network locality. They proposed an alternative technique which constrains each node's in-degree and out-degree, preventing a small number of nodes from launching an Eclipse attack.

Bhattacharjee *et al.* [32] proposed using threshold signatures to prevent malicious nodes from sending spurious negative replies. This ensured that lookup queries would always be able to make progress.

The above secure DHTs could generally defend against a constant fraction of malicious participating nodes. However, an attacker with sufficiently many identities, or an attacker who can choose non-random node IDs, could effectively disrupt routing. Therefore, most work on secure DHTs assumed node IDs would be certified by a central authority. Ryu *et al.* [35] proposed using Identity-Based Cryptography to avoid the overhead and complexity associated with storing and checking identity certificates.

In a *join-leave attack*, malicious nodes repeatedly join and leave the system, requesting new random IDs from the central authority and discarding those IDs which are not clustered. Once they have collected enough clustered IDs, the malicious nodes can perform a denial-of-service attack. Scheideler and Awerbuch [27, 102] developed efficient algorithms to assign new node IDs and re-shuffle the existing IDs so that the malicious nodes can never form a cluster.

Other work has examined how admission control can limit the rate at which malicious nodes enter the overlay. Several systems have proposed using computational puzzles to defend against attackers with limited CPU resources [30, 33, 99]. Baden *et al.* [28] proposed using the pigeonhole principle to limit identities based on a combination of observable attributes, such as autonomous system number or operating system.

## 2.3 Social-network-based defenses against the Sybil attack

In 2005, several systems (Turtle [92], SPROUT [79], and Danezis *et al.* [47]) independently proposed using social network information to fortify peer-to-peer systems against the Sybil attack. Turtle [92], an unstructured overlay, simply flooded requests over the social network. SPROUT [79] proposed augmenting a structured DHT's finger tables with social links; however, since social links have no structure in the DHT's ID space, these added links could only be used for the first few routing hops. Danezis *et al.*'s DHT [47] proposed a "zig-zag routing" technique which alternated routing hops between structured links and links chosen using the social network. These three publications introduced a new model for DHT security based on properties of the underlying social network rather than on the number of Sybil identities; however, the proposed systems did not provide efficient Sybil-proof routing.

**Sybil classification systems.** In 2006, the SybilGuard system [122] demonstrated how to use random walks on a social network to classify nodes as either “probably honest” or “probably Sybils”. For example, this could be used to check that at least one of a set of storage replicas was likely to be honest. The classification error rate grew with the number of edges between honest nodes and Sybil nodes, up to  $\mathcal{O}(\sqrt{n})$  such edges; the system could not provide any guarantees beyond that point. To make random walks useful, SybilGuard introduced a new *fast mixing* assumption about the social network (see Section 3.2.3). Although SybilGuard was not DHT-specific, it was applicable in peer-to-peer contexts, since it did not require a centralized server to store a map of the social network.

The subsequent SybilLimit system [121] improved SybilGuard’s design by using many short random walks instead of a few long random walks. Unlike SybilGuard, SybilLimit can tolerate up to  $\mathcal{O}(n/\log n)$  edges between honest and Sybil nodes, close to the theoretical limit, and misclassifies at most  $\mathcal{O}(\log n)$  Sybil nodes per such edge. Recently, the Gatekeeper system [113] improved on SybilLimit’s performance by admitting only  $\mathcal{O}(1)$  Sybil nodes when the number of attack edges is  $\mathcal{O}(1)$ . SybilLimit and Gatekeeper admit fewer Sybils and withstand much stronger attacks than SybilGuard could tolerate.

SybilLimit inspired the SybilInfer algorithm [46], which uses Bayesian inference to estimate the likelihood that any node is a Sybil. Unlike SybilLimit, SybilInfer stores the entire social graph on a single machine, which performs a heavyweight Monte Carlo simulation of many short random walks in order to discover the graph’s sparse cut and assign a Sybil probability to each node. Consequently, SybilInfer is inappropriate for large-scale peer-to-peer systems, but could be used for centralized recommendation engines or small-scale peer-to-peer networks.

**Recommendation and filtering engines.** Recommendation engines are closely related to Sybil classifiers: in both cases, the goal is to ignore inputs from dishonest participants. Reputation and recommendation engines have a long history of using an underlying trust network to improve rankings, from PageRank [90] to EigenTrust [70]. Cheng and Friedman [39] were the first to explicitly analyze the Sybil effect on reputation engines.

Nguyen *et al.* [114] developed SumUp, a recommendation engine which aggregates users’ ratings using the social graph; the Sybils’ influence is proportional to the size of the sparse cut between the honest and Sybil users. Yu *et al.* [123] gave an improved algorithm, DSybil, which yields provably optimal results under a set of plausible assumptions.

The Ostra email system [84] uses a social network to limit the number of messages that any user can send to another. Thus, a sparse cut bounds the amount of junk mail that any spammer can successfully send.

## 2.4 Applying social-network-based Sybil defenses to DHTs

Although SybilGuard and SybilLimit were intended to be generally applicable to all peer-to-peer systems, they are unfortunately not a good match for DHTs. In SybilLimit, each certification uses  $\mathcal{O}(\sqrt{n})$  bandwidth. Furthermore, in the worst case, a node must attempt to certify at least  $\mathcal{O}(n)$  candidates before being guaranteed to find any honest nodes at all. Thus, in order to build correct routing tables by straightforwardly applying SybilLimit,

each node must spend  $\mathcal{O}(n\sqrt{n})$  bandwidth — far more than the  $\mathcal{O}(\log n)$  bandwidth for a  $\log n$ -hop DHT or the  $\mathcal{O}(\sqrt{n})$  bandwidth for a one-hop DHT, and even more than the  $\mathcal{O}(n)$  bandwidth needed to fully replicate the entire DHT’s contents to every node (assuming each node stores  $\mathcal{O}(1)$  keys). DHT-specific Sybil defenses are required for an efficient design.

This dissertation’s Whānau protocol was inspired by SybilLimit and aims to fill this gap. Whānau uses the same fast-mixing social network model as SybilLimit, and borrows its technique of using short random walks to sample the social network. However, except for the random walk subroutine, the Whānau protocol has no other similarities with the SybilLimit protocol: Whānau is optimized for providing fast DHT lookup routing, and does not even attempt to classify nodes as honest or Sybils.

**Contributions in context.** A 2008 workshop paper [73] introduced the idea of constructing Sybil-proof DHT routing tables using random walks on a fast-mixing social network. It presented two DHTs: an unstructured DHT (described in Chapter 4), and a structured DHT (a precursor to Whānau) with finger and key tables constructed using novel sampling procedures. It recognized the problem of clustering attacks and offered a distributed balancing algorithm based on cuckoo hashing [91], similar to Scheideler and Awerbuch’s centralized solution [27]. However, this approach turned out to be difficult to prove correct.

The Whānau protocol, published in 2010 [74], introduced layered identifiers to combat clustering attacks. With this new technique, Whānau’s LOOKUP algorithm provably uses  $\mathcal{O}(1)$  messages to find the value associated with a key. This dissertation expands on the earlier publications, and explains how to apply Whānau in many scenarios of practical interest (Chapters 3 and 9).

## 2.5 Analysis and criticism of social-network Sybil defenses

As systems using random walks on social networks have continued to proliferate, a few researchers have recently started to reexamine the underlying model and its assumptions.

Viswanath *et al.* [115] have analyzed a variety of social-network-based Sybil defenses, and argue that all of the protocols essentially measure whether a node is the same community as a “root node”. They conclude that previous sophisticated techniques for detecting community structure in social networks should be incorporated into Sybil defenses. They also raise concerns that an adversary could target his attack by choosing to form links within the root node’s immediate community.

Whānau uses random walks not because they are the most accurate way to detect community structure, but because they are easily implemented as a distributed protocol. Moreover, Whānau’s goal is not to distinguish honest nodes from Sybil nodes, but to efficiently route DHT lookups; in this context, randomly sampled nodes are actually more useful than a sophisticated community detector would be. Clustered attack edges are certainly a concern for Whānau, but the problem is mitigated since every node is its own “root” in the social network. Consequently, only a small fraction of nodes can be affected by clustered attack edges (see Section 6.1.1).

Mohaisen, Yun, and Kim [87] have studied the mixing time of various social network datasets, and conclude that the time to full mixing is longer than previously thought. The

underlying reason is that a small number of nodes are very poorly connected to the main community, and are unable to reach most other nodes. Since these obscure nodes would themselves be unlikely to be reached by random walks from core nodes, they are largely invisible to Whānau; this would only pose a problem if they constituted a large fraction of the social network. Mohaisen *et al.*'s analysis does suggest that maximum mixing time is not a very good measure of a social graph's mixing properties for the purpose of this class of protocols. For this reason, Section 8.1 presents fine-grained data on the mixing properties of the social network datasets used in the simulations.

Mohaisen, Hopper, and Kim [86] advocated that, since trust is not binary, the model should be refined to account for varying trust weights on social connections. They offer several modulated random walks which protocols (including Whānau) could use to incorporate these weights.



# Chapter 3

## Assumptions and goals

This chapter describes assumptions we will make about the client applications using the DHT; about the communications network used by the DHT; about the social network underpinning the DHT’s security; and about the adversary model. In short, the Whānau DHT’s design assumes that:

- The application supports integrity for key-value records.
- The social network is highly connected.
- The adversary cannot prevent honest nodes from communicating directly.

We then define what it means for a DHT to be “Sybil-proof” under these assumptions.

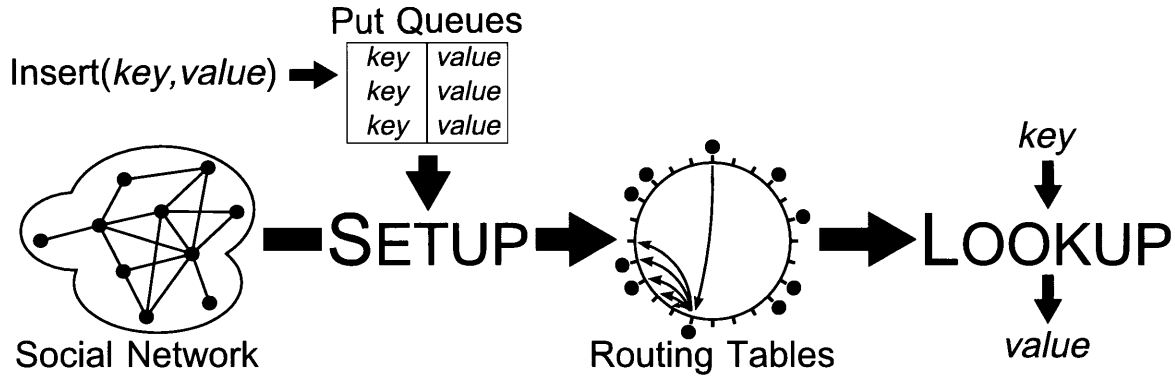
We will also set out ambitious performance and efficiency goals, and show that it is straightforward to provide either performance without security or security without performance. It is the combination of performance requirements and security requirements that makes the problem complex.

### 3.1 DHT-to-application interface

The prototypical application programming interface for a distributed hash table includes methods to join and leave the overlay, to insert, update, and delete key-value records, and

<u>Standard DHT API</u>	<u>Core Whānau API</u>
JOIN( <i>bootstrap-address</i> )	
LEAVE( $\cdot$ )	SETUP( $\{neighbor_1, \dots, neighbor_d\},$
INSERT( <i>key</i> , <i>value</i> )	$\{(key_1, value_1), \dots, (key_k, value_k)\}$ )
UPDATE( <i>key</i> , <i>value</i> )	
DELETE( <i>key</i> )	
LOOKUP( <i>key</i> ) $\rightsquigarrow$ <i>value</i>	LOOKUP( <i>key</i> ) $\rightsquigarrow$ <i>value</i>

**Table 3.1:** Standard DHT application programming interface compared with Whānau’s application programming interface.



**Figure 3-1:** Overview of Whānau. SETUP builds structured routing tables which LOOKUP uses to route queries to keys.

to look up a key. (Table 3.1.) In contrast, the core Whānau DHT reduces this interface to two procedures, illustrated in Figure 3-1:

- **SETUP**( $\cdot$ ) is used both to build routing tables and to insert keys. Each node must periodically call **SETUP** with its local parameters, *i.e.*, the contact information of the node's  $d$  social neighbors, and the  $k$  key-value records to store into the DHT. **SETUP** initiates a distributed protocol which cooperatively transforms these local parameters into a set of routing table structures stored at each node.
- **LOOKUP**( $key$ ) uses the routing tables built by **SETUP** to find and return the target *value*.

When an application running on a node  $u$  needs to insert a record into the DHT, it appends the records to a local **put queue**:

$u.ININSERT(key, value)$   
 1  $u.putqueue \leftarrow u.putqueue \cup \{(key, value)\}$

Similarly, when a social connection is established with another node, it is appended to the neighbors list:

$u.JOIN(address, public-key)$   
 1  $u.neighbors \leftarrow u.neighbors \cup \{(address, public-key)\}$

These local modifications to  $u.putqueue$  and  $u.neighbors$  are incorporated into the distributed routing tables when the next **SETUP** round runs. One drawback of this interface is that new keys are not immediately visible to other DHT clients; the reason for this delay is that the nodes must re-partition the key space to account for any newly introduced key clusters. Section 9.7.2 discusses techniques to mitigate this problem.

Although the effect of **INSERT** is delayed, Section 9.7.1 explains how to extend Whānau with an **UPDATE** protocol which dynamically mutates the value associated with an existing key. Many DHT applications, such as rendezvous, routing, and filesystems, update values more frequently than inserting new keys, and can be adapted to use Whānau.

Whānau has no methods corresponding directly to DELETE or LEAVE: nodes may delete keys simply by not including them in the next round of SETUP, and they may leave the DHT by opting not to participate at all.

In order to start each SETUP round at approximately the same time, all nodes must have loosely synchronized clocks. The SETUP protocol proceeds in lock-step. Since all messages from the previous step must be processed before starting the next step, the maximum tolerable clock skew is the fixed step time (which could be, *e.g.*, 1 second to 1 hour, depending on application requirements) minus one network time-of-flight. Thus, the total SETUP execution time increases with clock skew.

The specifics of SETUP's parameters depend on the application's needs. The neighbor contact information must be sufficient to send and receive authenticated messages over the underlying network. Typically, this might be an IP address and a public key or shared secret for each neighbor. Alternatively, Whānau could send messages over a dedicated, physically secured network, over a telephone connection, over a social website such as Facebook [4], or over a combination of these mediums.

The core Whānau DHT protocol treats keys and values as opaque data blobs, and replicates them over many nodes. To limit the load on individual nodes, all key-value pairs should be small and approximately the same size. Section 9.6 describes how to extend Whānau with an additional layer of indirection to support large, variable-sized data blocks.

### 3.1.1 Example application: rendezvous service

As a simple but illustrative scenario, consider a telephony (VOIP) or instant messaging (IM) application which uses a DHT as a rendezvous service. Suppose that each user is identified by a public key, and publishes a single, self-signed key-value record (*public key, current IP address*) into the DHT. When a user moves to a new location, the application updates this record with its new IP address. To initiate a phone call or to send an IM to a user, a client looks up the user's public key in the DHT, verifies the returned tuple's signature, and then opens a connection to the returned IP address.

Storing IP addresses into the DHT uses it as an indirection layer [109]. An alternative approach, discussed in Section 9.4, would use Whānau as a routing layer, tunneling messages over the persistent channels between social neighbors. Whether to use the DHT for indirection or routing depends on application requirements: indirection has lower overhead, but routing can tunnel messages through NATs and firewalls, and even works in scenarios where there is no underlying IP network, but only direct social connections. Both techniques can be employed side-by-side in a single Whānau DHT.

In the peer-to-peer infrastructure model, every user runs a node to contribute to the DHT. This node need not be continuously available when the user is offline, as long as a large number of other honest nodes are available to serve DHT requests at any given time.

This rendezvous application, although trivial, has a number of features which make it a useful and interesting example for Whānau:

- It is **simple**, with little additional machinery.
- It demonstrates the common pattern of using a DHT for **routing** messages to nodes instead of storing data blocks.

- It is **broadly applicable** in many contexts where users or machines must locate each other, such as telephony, IM, videoconferencing, games, and file sharing.
- It is **relevant**: DHTs are currently used in systems such as P2PSIP, WebEx, and BitTorrent [72, 78, 104], and Sybil attacks are recognized as a problem.
- It is **open** to anyone on the Internet.
- It is **not inherently centralized**, and adding a central authority might introduce undesirable performance and trust bottlenecks.
- It requires **large scale** to support millions of users.
- It demands **fast lookups** to minimize connection setup latency.
- It **updates records** frequently (Section 9.7.1), but only inserts new keys when users join the system for the first time.
- It stores **one key per node**, which turns out to be the most challenging case for Whānau to handle efficiently (Section 9.3).

Most realistic applications will layer substantial additional functionality on top of the DHT infrastructure. For instance, a practical telephony/IM application would need a PKI to map human-readable names to public keys, timestamps to ensure the freshness of returned address records, privacy controls for location information and buddy lists, and so on. While these features are generally outside the scope of this dissertation, they must be carefully implemented to avoid introducing performance or availability bottlenecks. Whānau is designed to support this aim.

### 3.1.2 Division of labor: integrity versus availability

Malicious nodes can always join the DHT normally and attempt to insert arbitrary key-value records, including a different value for a key already in the DHT. Thus, a secure DHT must support some mechanism to protect the integrity of records inserted by honest clients. There are a variety of standard techniques [110]:

- **Digital signatures**: before inserting the pair  $(key, value)$  into the DHT, a client signs the pair and appends the signature to the value. This requires that when clients perform lookups, they know (via, *e.g.*, a PKI) which public key to check the signature against.
- **Self-certifying keys**: before inserting the pair  $(PK, value)$  into the DHT, a client signs the pair using the public key  $PK$  and appends the signature to the value. Such a self-signed record is straightforward to check, but the DHT routing key must be a cryptographic public key (or a cryptographic hash of a public key).
- **Content-hash keys**: a client stores a data block  $B$  by inserting the pair  $(\mathcal{H}(B), B)$  into the DHT, where  $\mathcal{H}$  is a cryptographic hash function (*e.g.*, SHA-256 [22]). Any node can easily hash the value  $B$  and check its computed  $\mathcal{H}(B)$  against the key. However, records using this technique are not mutable.

Because different techniques may be appropriate for different applications, the Whānau protocol is designed to be compatible with a range of integrity mechanisms. Thus, **Whānau is responsible for providing availability, and the application is responsible for checking integrity.** Specifically, Whānau guarantees that looking up a key will find all values inserted by honest nodes for the specified key, but may also return some bogus values inserted by malicious nodes. The application should be able to filter the returned set to retain only authentic values.

As an optimization, if the application provides the DHT with an efficient black-box function to check the authenticity of a record, Whānau nodes can avoid storing and transmitting any bogus values in the first place. In this case, lookup requests will only return authentic values.

This division of labor enables applications to use integrity techniques outside the standard set, or to extend the standard techniques. For instance:

- An application may wish to guarantee freshness by adding serial numbers or timestamps to signed records.
- The DHT might be used as a cache of results that are difficult to compute but easy to check.
- In the example rendezvous application, the IP addresses need not even be signed — the client could simply attempt to initiate a secure (*e.g.*, TLS [21]) session with every returned IP address, and reject any counterparty which cannot interactively authenticate itself.

While most DHT applications support some form of integrity, there are a few exceptions, such as file sharing and search. A file stored under the key “Nosferatu” may appear to be a well-formed video stream, but a downloading client cannot automatically check whether it is the authentic 1922 film or an advertisement uploaded by a spammer. In this situation, Whānau can guarantee that the authentic film will be amongst the results (if it has been uploaded by an honest node), but the user may be forced to manually filter out any chaff.<sup>1</sup>

## 3.2 Social network model

Whānau depends on having access to an external **social network**, an undirected graph of connections between Whānau’s users. Whānau’s design is independent of the source of these connections: they might derive from personal or business relationships, or they might correspond to something more abstract, such as ISP peering agreements. Each user operates a Whānau node, and must provide this software with initial contact addresses for her social neighbors’ nodes; the nodes then establish and maintain persistent, authenticated communication channels with their neighbors. We do *not* assume a central trusted node storing a map of the entire social network — each node only needs to know its neighbors.

---

<sup>1</sup>Various reputation systems (*e.g.*, [26]) can be used to amortize this work over many users, but it still must be done manually if no automatic spam filter is available.

Like previous work [46, 84, 113, 114, 121–123], the Whānau DHT relies on an underlying network with certain useful properties: it should be *fast mixing*, and should have a *sparse cut* between the honest region and the Sybil region. Whether real social networks have these properties is an active subject of research [83, 87, 115, 121], but there are good reasons (outlined in Section 3.2.4) to be optimistic that they will.

### 3.2.1 Social connections: physical rendezvous, social network services

As noted above, Whānau nodes can use social connections from any source (or combination of sources), as long as nodes can communicate with their social neighbors, and as long as the resulting social network has the properties described in the following sections. For example, face-to-face meetings and online social network services are both potential sources of social connections for Whānau.

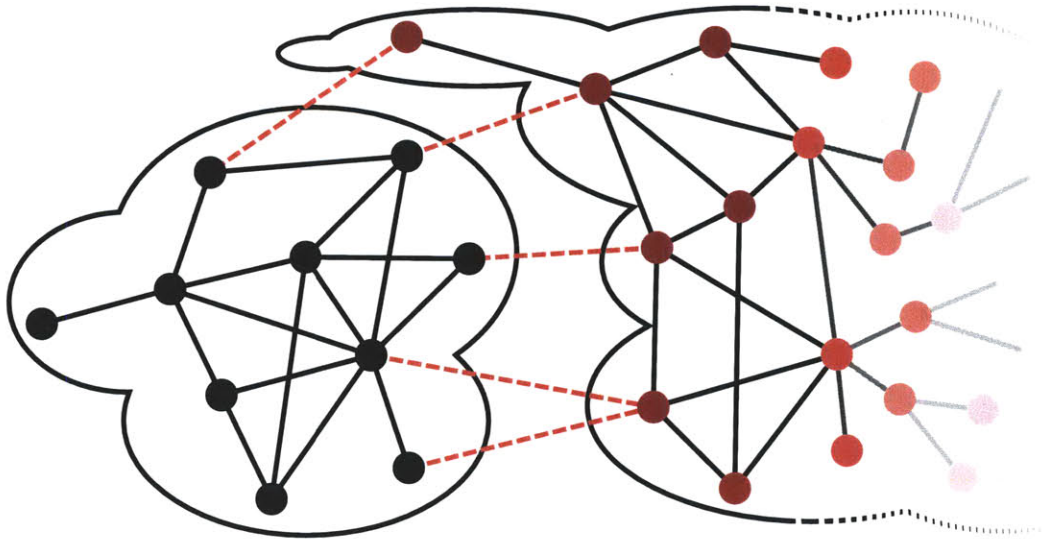
**Face-to-face meetings.** Physical rendezvous between DHT nodes creates social connections based on direct human-to-human interactions. Suppose that two friends meet in a coffeeshop and agree to stay in touch. They can pull out their mobile phones (which are Whānau nodes) and instruct them to rendezvous and form a permanent connection with each other. The phones exchange public keys and contact information over the local connection, and each creates an entry for the other in its local address book (which is Whānau’s social neighbor list).

After the friends part, their phones continue to maintain contact with each other over the Internet. Each time they run the Whānau SETUP procedure, the nodes use the social neighbor lists they have collected in this way.

In order to maintain a long-term persistent connection, the nodes must update each other whenever their current IP addresses change. This technique works well when only one node’s address changes at a time, but if both nodes go offline and then return with new IP addresses, they may not be able to contact each other. However, using a technique similar to UIA [52], the nodes can use the DHT itself to rendezvous (Section 3.1.1): as long as *some* neighbors are reachable when they come online, the nodes can re-establish a direct connection by looking each other’s IP addresses up in the DHT.

**Online social network sites.** Of course, physical rendezvous may be inconvenient in some cases. Existing social network services such as Facebook, MySpace, LinkedIn, Orkut, Flickr, and Twitter [4–6, 8, 9, 13], and implicit social networks such as email and mobile phone address books, are therefore a natural source for bootstrapping Whānau nodes’ social connections. The main requirement is that the service permits a user’s Whānau instance to send authenticated messages to her friends’ Whānau instances. This capability can be used either to broadcast the node’s current contact information, or to send Whānau protocol messages directly over the social network site.

For example, a Whānau implementation could use Facebook in a primitive way by periodically posting the node’s current IP address and public key hash to the user’s Wall, and using the Facebook API to probe the user’s friends for similar posts. The node could



**Figure 3-2:** 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.

then use this contact information to establish secure TCP connections with the friends’ Whānau nodes, and send Whānau routing traffic over these direct connections.

A more sophisticated approach would use the Facebook LiveMessage API [1] to send messages between Whānau instances using the Facebook service as a transport. This could be used, as before, to bootstrap secure TCP connections over the public Internet; alternatively, all Whānau messages could be tunneled over Facebook in this way, at some cost in efficiency.

It is generally desirable either to cut external social network services out of the loop as quickly as possible, or to bootstrap social connections using multiple sources. Otherwise, the social network service provider itself becomes a trust bottleneck.

### 3.2.2 First assumption: the sparse cut

An adversary can infiltrate the social network by creating many **Sybil nodes** (phoney identities which behave in a Byzantine manner) and gaining the trust of honest people. We’ll assume that the Whānau protocol cannot directly distinguish Sybil identities from genuine ones — if it could, it would be trivial to eject Sybils from the system!

In the analysis, we conceptually divide the social network into two parts: an **honest region** containing all honest nodes, and a **Sybil region** containing all Sybil identities. (Figure 3-2.) An **attack edge** is a connection between a Sybil node and an honest node. An **honest edge** is a connection between two honest nodes [122]. 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, this **sparse cut** assumption can be justified 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.

Returning to the physical-rendezvous scenario outlined in Section 3.2.1, in order to create a million attack edges, an attacker would need to send agents to meet a million honest users in person and persuade each of them to form a persistent social connection with the agent. This social attack requires many more resources than stealing IP addresses or solving computational puzzles requires.

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

### 3.2.3 Second assumption: a fast-mixing social network

Since we’re relying on the existence of a sparse cut to isolate the Sybil region from the honest region, we must also assume that there is *no* sparse cut dividing the honest region in two. This is equivalent to the assumption that the honest region is 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 [42]. Roughly speaking, the ending node of a random walk will be a random node in the network, with a probability distribution proportional to the node’s degree.

The **mixing time**  $w$  of the graph is the number of steps a random walk must take to reach this smooth distribution.

**Definition** (mixing time). Let  $P_u^i$  be the probability distribution of an  $i$ -step random walk starting at  $u$ , and let  $\pi$  be its stationary distribution. We use the standard textbook definitions for  $P_u^i$  and  $\pi$ :

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

The **mixing time** of the network is the smallest  $w$  such that for all node pairs  $(u, v)$ :

$$\frac{1}{2}\pi(v) \leq P_u^w(v) \leq \frac{3}{2}\pi(v)$$

In other words, the distribution of a random walk with at least  $w$  steps is approximately the stationary distribution.

**Definition** (fast mixing). The social network is said to be **fast mixing** if  $w = \mathcal{O}(\log n)$ .

Section 8.1.2 shows that graphs extracted from some real social networks closely approximate this fast-mixing model. Note that it is only the honest region that we expect to be fast mixing; the Sybil region’s structure is chosen by the adversary!



### 3.2.4 Why is this a reasonable model?

Mixing time is a measurement of a graph’s connectivity. Despite a wide variety of data sources with different characteristics [2, 49, 83], and some disagreement over appropriate definitions [87], it is generally agreed that natural social graphs are highly connected. Chapter 8 confirms that several real social graph datasets have good mixing properties and that Whānau can perform well using them.

On the other hand, the reasonableness of assuming a sparse cut separating honest nodes from Sybil nodes cannot be experimentally established at this time: after all, malicious parties do not currently have a strong incentive to create attack edges. Preliminary measurements of social network imposters and spam [12, 59, 118] suggest that current percentages of fake and compromised accounts, while a cause for concern, are insufficient to mount a Sybil attack against Whānau. However, widely-deployed, valuable infrastructure services based on social network Sybil defenses would change the present dynamic, and the outcome will depend primarily on the users’ behavior. If honest users are sent friend requests by spammers, will they accept them unconditionally, or will they exercise discretion?

The software’s user interface for adding social links will likely have an effect on user behavior. For example, simply reminding users that creating links to malicious users can affect their software’s performance may prompt some to evaluate friend requests more carefully. More subtly, separating social connections into trust categories (*e.g.*, “I met this person face-to-face” vs “I met this person online”) would permit users to reciprocate friend links while enabling the software to treat some links as more trusted than others [86]. Since Whānau only uses reciprocated links (which constitute the majority of links [82]), unidirectional links created by spammers have no effect.

Growth in social network spam would also spur greater technical efforts to suppress it, and may also lead to stronger social defense mechanisms. For example, in some circles today, if a user reciprocates a friend connection to an obvious spammer, the user’s real friends will notice and notify her of her mistake. This kind of mutual reality check may become more widespread and sophisticated if social spamming becomes more common.

Currently, Facebook spammers prefer compromising real accounts over creating fake accounts and convincing users to reciprocate friend links [59]. While this would only deny service to non-compromised Whānau users if many other users were compromised, this attack vector will always remain a concern. Few large-scale distributed systems can tolerate the compromise of a majority of their nodes.

Experience with phishing scams suggests that, despite countermeasures, some people are likely to reciprocate large numbers of bogus friend requests. However, such an indiscriminating user can only increase the number of attack edges  $g$  by the number of connections he has to other honest users, because his node itself can be cut out of the honest region. Thus, a few indiscriminate individuals cannot greatly influence the outcome: in order for Whānau’s availability to be compromised, a large fraction of users must create social links to Sybils.

To summarize, the Whānau model relies upon a fast mixing social network with a sparse cut between the honest region and the Sybil region. Measurements of existing social networks show that they have good mixing properties. Current social networks contain few attack edges; however, the continued existence of a sparse cut depends on future user be-

havior. With a combination of social and technical defense mechanisms, we can be optimistic that most users will be able to make good decisions and the social network’s quality will remain high.

### 3.3 Sampling the social network using random walks

The fast-mixing and sparse-cut assumptions introduced in the previous section enable random walks to be used as a powerful building block for distributed protocols [84, 113, 121, 122]. 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 [121].

This random walk sampling technique is, in fact, the only way in which the Whānau protocol uses the social network. 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.

Because Whānau has such a narrow interface to the social network, it is possible to imagine using another source of trust instead. For example, sampling randomly from a master list published by a trusted authority (such as the phone book) could also plausibly satisfy the property that most of the returned samples will be honest. One of Whānau’s strengths is that it can use any such sampling technique, or a combination — combining the samples returned by multiple different techniques might incur some additional overhead, but would be secure as long as one of the techniques is secure. This means that this dissertation’s social-network-based techniques are compatible with and complementary to other approaches using IP addresses, puzzles, or central authorities.

#### 3.3.1 Winners and losers

One weakness of the random-walk sampling technique is that some honest nodes may be situated near a concentration of attack edges in the social network. These nodes, which we will call **loser nodes**, have been lax about ensuring that their social connections are to real people, and their view of the social graph does not contain as much useful trust 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 querying bad nodes before they find a good node to query.

Luckily, only a small fraction  $\epsilon$  of honest nodes can be losers, because a higher concentration of attack edges in one part of the network means a lower concentration elsewhere (for a given total number of attack edges). Most honest nodes will be **winner nodes**. Section 6.1.3 will treat losers and winners in more detail.

### 3.3.2 Balancing load using virtual nodes

In the stationary distribution, proportionally more random walks will land on high-degree nodes than on low-degree nodes. To handle high-degree nodes well, each Whānau participant creates one virtual node [110] per adjacent 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).

As a matter of policy and fairness, this dissertation argues that both workload and trust should be allocated according to each person's level of participation in the social network. In other words, highly connected nodes should do more work than casual participants. Whānau's use of virtual nodes fulfils this policy goal by allocating table size and bandwidth consumption proportionally to each social node's degree. While it is possible to adapt Whānau to different policies, this dissertation assumes that the goal is a proportional policy.

## 3.4 Communications network model and adversary model

Like other DHTs, Whānau is an overlay network which constructs its peer-to-peer links over an underlying communications network. The Internet is the most obvious context, but cellular or ad-hoc networks (especially mobile ad-hoc networks and vehicular ad-hoc networks) are also natural settings. In some cases, it may even make sense to tunnel routing messages over a social network Web site like Facebook — for example, when a friend's IP address is unavailable.

To work well with the Whānau DHT, the underlying network should have the following properties:

- The message delivery delay between social neighbors is less than the SETUP step time.
- Messages between social neighbors are authenticated and cannot be forged.
- A malicious attacker cannot prevent social neighbors from communicating directly.
- Random, non-malicious connection failures and dropped messages may occur, as long as a substantial fraction of honest connections are up at any time.
- Random, non-malicious node failures and churn may occur, as long as a substantial fraction of honest nodes are up at any time.

The Internet generally satisfies these properties, as long as social neighbors know each other's cryptographic public keys. Care must be taken to avoid connection-sniping denial-of-service attacks against the transport layer [68,93]. Packet flooding [19], which works by consuming a peer's entire bandwidth allocation, remains a concern in Whānau. However, Whānau's high level of redundancy mitigates this problem.

In this dissertation, we will assume that all malicious nodes are controlled by a single adversary. (While separate denial-of-service attacks by multiple adversaries are possible, this is weaker than an attack in which the adversaries cooperate with each other.) This adversary can observe messages between immediate social neighbors, but cannot block or

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

**Table 3.2:** Summary of social network parameters used in the analysis.

delay them. However, the adversary can try to disrupt the DHT’s operation by creating many Sybil identities which spread misinformation. These Sybil nodes may deviate from the DHT protocol in Byzantine ways: they may send arbitrary routing messages at any time and may make up arbitrary responses to queries from honest nodes.

### 3.5 The main security property

Informally, when we say that 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. Using the social network parameters introduced thus far (summarized in Table 3.2), we can make this main security property more precise:

**Definition** (Security goal). A DHT protocol is  $(g, \epsilon, p)$ -**Sybil-proof** if, against a Byzantine adversary controlling up to  $g$  attack edges, the protocol’s LOOKUP procedure succeeds with probability at least  $p$  on any honest key, for at least  $(1 - \epsilon)n$  of the honest nodes.

Given a  $(g, \epsilon, \frac{1}{2})$ -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.<sup>2</sup> Using this approach, 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).

The parameter  $\epsilon$  represents the fraction of loser nodes, which is a function of the distribution of attack edges in the network. If the  $g$  attack edges are distributed uniformly, then  $\epsilon$  may be zero; if the attack edges are clustered, then a small fraction of nodes may be losers.

Chapter 6 will use the parameters in Table 3.2 to analyze the Whānau protocol, but we will not assume that all of these parameters are known by the honest participants. Whānau nodes need order-of-magnitude estimates of  $m$ ,  $w$ , and  $k$  to choose appropriate table sizes and walk lengths. They do not need to know  $g$  or  $\epsilon$ .

Proving that a protocol is Sybil-proof doesn’t imply that it cannot be broken. For example, the Whānau DHT is Sybil-proof, but could be broken by social engineering attacks that invalidate the assumption that there is a sparse cut between the honest and Sybil regions. Alternatively, a DHT protocol might be broken by using cryptographic attacks or attacks on

<sup>2</sup>For the Whānau DHT, it turns out to be more efficient to increase the routing table size instead of running multiple parallel instances.

the underlying network infrastructure. These attacks are all concerns to be taken seriously, but they are not Sybil attacks, since they do not involve creating Sybil identities [48]. This dissertation focuses on techniques to protect against Sybil attacks; these new techniques should be combined with existing techniques to protect against other classes of attacks.

## 3.6 Performance

Simply flooding LOOKUP queries over all links of the social network is arbitrarily Sybil-proof, but not efficient [47, 92]. Malicious 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 node storing the target key, then the adversary cannot prevent each of these nodes from forwarding the query to the next. Thus, the query will always reach the target node, which will reply with the correct value. Unfortunately, in this trivial protocol, most of the honest nodes must be contacted for every lookup, doing  $\mathcal{O}(n)$  work each time.

On the other hand, existing one-hop DHTs are very efficient — requiring only  $\mathcal{O}(1)$  messages for lookups and  $\mathcal{O}(\sqrt{n})$  work to set up routing tables<sup>3</sup>— but not secure against Sybil attacks. The following chapters will develop a new structured DHT which combines this optimal efficiency with provable security.

---

<sup>3</sup>If  $n = 5 \times 10^8$ , the approximate number of Internet hosts in 2010, then a per-node table of  $\sqrt{n}$  entries may be much more feasible for bandwidth- and storage-constrained devices than a table that scales linearly with  $n$ .



# Chapter 4

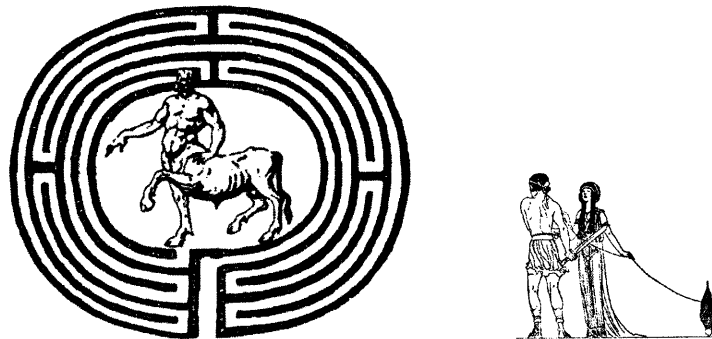
## An unstructured DHT protocol

In the previous chapter, we learned that taking a random walk on a fast-mixing social network is a powerful technique, enabling us to sample random nodes that are probably honest. This chapter illustrates how this building block can be used to build a simple, unstructured DHT protocol which is Sybil-proof and which performs better than flooding, but not as well as a structured protocol. The following chapters, which do not depend on this chapter, will develop the highly efficient, but more complex, Whānau protocol.

### 4.1 Theseus's algorithm

Our goal is to find a specific key stored in the social network and return the associated value, akin to the mythical Theseus's goal of finding the Minotaur imprisoned in King Minos's labyrinth and returning with its head. (Figure 4-1.) Lacking a map or signposts (*i.e.*, routing tables), Theseus wanders the labyrinth randomly until he runs out of Ariadne's thread, which he has used to mark his path. He then follows the thread back to the entrance and tries again until he finds the Minotaur.

Theseus's random-walk algorithm has similar characteristics to the bounded-flooding query protocols used by unstructured peer-to-peer overlays like Gnutella and Turtle [92,96]. We can express Theseus's algorithm in pseudocode as follows:



**Figure 4-1:** Theseus searching the labyrinth for the Minotaur.

```

u.LOOKUP(key; w)
1  repeat  $v \leftarrow u.\text{RANDOM-WALK}(w)$            ▷ Send a RANDOM-WALK RPC to u
2      $value \leftarrow v.\text{QUERY}(key)$            ▷ Send a QUERY RPC to v
3  until found authentic value

```

```

u.RANDOM-WALK(w)
1  if  $w > 0$ 
2     then  $v \leftarrow \text{RANDOM-CHOICE}(u.\text{neighbors})$  ▷ Pick a random social connection
3         return  $v.\text{RANDOM-WALK}(w - 1)$        ▷ Send recursive RPC
4     else return u                             ▷ Return address of final node

```

```

v.QUERY(key)
1  if exists  $(key, value) \in v.\text{putqueue}$        ▷ Table lookup of locally-inserted keys
2     then return value
3     else error “not found”

```

Theseus’s algorithm illustrates how random walks address the brute-force Sybil attack described in Chapter 1.<sup>1</sup> Given that a sparse cut separates the honest region from the Sybil region (Section 3.3), most random walks will stay within the honest region, and thus will sample random honest nodes. Hence, within  $\mathcal{O}(m \log m)$  trials, the algorithm will have visited every single honest node (*cf.* the Coupon Collector’s Problem [85]).<sup>2</sup> Regardless of the adversary’s actions, the target key will be found if it is stored in some honest node’s put-queue. The expected time before finding the target key is  $\mathcal{O}(m)$ .

## 4.2 Introducing caches to reduce lookup cost

Theseus’s lookup algorithm above repeatedly visits many of the same nodes every time it executes. If we introduce mutable state at each node, Theseus can cache visited keys from one run to another, avoiding some of that repeated work. Then, each node would do at most  $\mathcal{O}(m)$  random walks in total, instead of  $\mathcal{O}(m)$  random walks per lookup.

Moreover, when querying a random node’s put-queue, it costs nothing extra to also check whether the target key is in the remote node’s *cache* table:

```

v.QUERY(key)
1  if     exists  $(key, value) \in v.\text{putqueue}$ 
2     then return value
3  elseif exists  $(key, value) \in v.\text{cache}$ 
4     then return value
5  else error “not found”

```

<sup>1</sup>By maintaining no routing table state at all, Theseus’s algorithm sidesteps the Eclipse and clustering attacks entirely.

<sup>2</sup>The time to visit all nodes is expressed in terms of  $m$  (the number of honest edges) instead of  $n$  (the number of honest nodes) because low-degree nodes are visited less frequently than high-degree nodes.



Used in this way, caches multiply the effective work done by each query message; *e.g.*, if the average cache is the same size as the average put-queue, lookups will require half as many queries to find a key. Given this, it makes sense to pre-fill the caches during the SETUP phase, like this:

$u.$ SETUP( $\{neighbor_1, \dots, neighbor_d\}, \{(key_1, value_1), \dots, (key_k, value_k)\}; w, r$ )

```

1  $u.neighbors \leftarrow \{neighbor_1, \dots, neighbor_d\}$ 
2  $u.putqueue \leftarrow \{(key_1, value_1), \dots, (key_k, value_k)\}$ 
3  $u.cache \leftarrow u.SAMPLE-RECORDS(w, r)$ 
4 return  $u.neighbors, u.putqueue, u.cache$ 

```

$u.$ SAMPLE-RECORDS( $w, r$ )

```

1 for  $i \leftarrow 1$  to  $r$ 
2     do  $v_i \leftarrow u.RANDOM-WALK(w)$ 
3          $(key_i, value_i) \leftarrow v_i.SAMPLE-RECORD()$ 
4 return  $\{(key_1, value_1), \dots, (key_r, value_r)\}$ 

```

$v.$ SAMPLE-RECORD()

```

1  $(key, value) \leftarrow RANDOM-CHOICE(v.putqueue)$ 
2 return  $(key, value)$ 

```

The above pseudocode fills each node's cache by sending out  $r$  independent length- $w$  random walks to sample  $r$  nodes, and querying each node for a random key-value record. Some of the samples may be Sybil nodes which return bogus values, but a large number (at least  $\Omega(r)$ ) will be good.

**Theorem.** *If SETUP gets, on average,  $r'$  good samples per virtual node, then caching LOOKUP is expected to query approximately  $\frac{km}{r'}$  good nodes before completing.*

**Corollary.** *If the honest region and Sybil region are separated by a sparse cut, then caching LOOKUP completes in  $\mathcal{O}\left(\frac{km}{r}\right)$  time.*

*Proof.* If  $r' \leq k$ , then the cache is not relevant: the expected number of queries before finding the key in a node's put-queue is less than  $m \leq \frac{km}{r'}$ .

If, on the other hand,  $r' \gg k$ , then  $\frac{km}{r'} \ll m$ . Therefore, the first  $\frac{km}{r'}$  good queries will rarely hit a previously-queried virtual node. Each such query is equivalent to sampling  $r'$  random keys with replacement. We need to sample an expected  $km$  keys before finding the target key. Thus, the expected number of good queries needed is  $\frac{km}{r'}$ .  $\square$

The cache size for this unstructured protocol is a trade-off between storage and lookup latency/bandwidth. For example, if  $r = \mathcal{O}\left(\sqrt{km}\right)$ , then lookups will require  $\mathcal{O}\left(\sqrt{km}\right)$  messages on average. The lookup latency can be reduced substantially by broadcasting these messages in parallel. Unfortunately, this does not improve the bandwidth and CPU consumption, which will ultimately become the DHT's limiting factor.

The unstructured caching protocol is simple to understand and is evidently Sybil-proof. By adding additional state to nodes, it improves lookup performance over Theseus's algorithm or previous unstructured overlay protocols that work by flooding query messages over the network [92, 96]. However, its efficiency is not as good as one might hope: existing, insecure one-hop structured DHTs with similar table sizes would find a target key using only  $\mathcal{O}(1)$  messages. The following chapters will show how to bridge this gap between efficient-but-insecure and secure-but-inefficient. By adding structure, we can improve performance, resulting in a DHT protocol which is both efficient and secure.

# Chapter 5

## The Whānau protocol

The previous chapter demonstrated a simple, secure protocol using random walks on the social network. This unstructured DHT protocol was Sybil-proof, but not efficient. In this and the following chapters, we will explore Whānau, a more complex *structured* DHT design. Whānau’s SETUP has similar resource usage to the unstructured protocol, but Whānau’s LOOKUP protocol uses only  $\mathcal{O}(1)$  messages to find a key. The main design challenge is to craft Whānau’s structure in such a way that it cannot be exploited by a clustering attack.

### 5.1 Overview of Whānau

This section outlines Whānau’s main characteristics and explains how the protocol works in the non-adversarial case. Sections 5.2 and 5.3 will explain the SETUP and LOOKUP protocols in detail, and Chapter 6 will analyze Whānau’s performance in an adversarial Sybil-attack scenario.

#### 5.1.1 Ring structure

Whānau’s structure resembles other DHTs such as Chord [110], SkipNet [64], and Keliips [63]. Like SkipNet and Chord, Whānau assumes a given, global, circular ordering  $\prec$  on keys (*e.g.*, lexical ordering on strings). The notation  $x_1 \prec x_2 \prec \dots \prec x_n$  means that for any indices  $i < j < k$ , the key  $x_j$  is on the arc  $(x_i, x_k)$ .

On a ring, one key cannot be said to be “before” or “after” another key. However, with reference to a *given set* of more than two keys, **successor** and **predecessor** keys *within that set* are well-defined. In a set  $x_1 \prec x_2 \prec \dots \prec x_n \prec x_1$ , the successor of  $x_1$  is  $x_2$ , the successor of  $x_{n-1}$  is  $x_n$ , and the successor of  $x_n$  is  $x_1$ .

**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. The usual reason for such a hash function is to distribute keys uniformly over the metric space so that keys will be spread evenly over nodes. However, an adversary can use guess-and-check to construct many keys that fall between any two neighboring honest keys, warping the distribution

of keys in the system and defeating 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.*, the set of keys can be partitioned into contiguous slices), but still requires defenses against clustering attacks, since the adversary can insert keys anywhere in the ordering.

**Finger tables and key tables.** Most structured DHTs have routing tables with both “far pointers”, sometimes called *fingers*, and “near pointers”, called *leaves* or *successors*. Whānau follows this pattern. Each node has a set of **layered IDs** (described in Section 5.1.2) which are of the same data type as the keys. Each node has a **finger table** containing  $\mathcal{O}(\sqrt{km})$  pointers<sup>1</sup> to other nodes with IDs distributed evenly over the key space. Likewise, each node has a **key table**<sup>2</sup> storing the  $\mathcal{O}(\sqrt{km})$  honest key-value records immediately following each of its IDs (*i.e.*, the IDs’ successors).

Finger tables are constructed simply by sending out  $\mathcal{O}(\sqrt{km})$  random walks, collecting a random sample of (honest and Sybil) nodes along with their layered IDs. Key tables are built using a more complex sampling procedure, described in Section 5.2.

**Lookup.** Together, the key tables of a node’s fingers cover the entire set of  $km$  keys stored into the DHT by honest nodes (with high probability). Moreover, each individual finger’s key table stores its IDs’ successor keys from this set. This structure enables any node to perform fast one-hop lookups: simply find the ID preceding the target key within the finger table, and send a query message to the associated finger node.

Because the chosen ID is likely to be close to the target key (Section 6.2 will define what is meant by “close”), the chosen finger is likely to have the needed successor record in its key table. (If it does not, a few retries with different predecessor fingers should suffice to find one that does.) In contrast with the unstructured protocol from Section 4.2, this lookup strategy uses a constant number of messages on average, and  $\mathcal{O}(\log n)$  messages (which may be sent in parallel) in the worst case.

## 5.1.2 Layered identifiers

Recall from Section 1.2.3 that an attacker can try to prevent access to a particular target key  $x_t$  by clustering his own keys or IDs near  $x_t$ . In a *key clustering* attack against Whānau, the Sybils would insert many immediately preceding keys (*e.g.*,  $x_t - 1$  if keys are integers) into the DHT, filling the key tables of nodes which would otherwise have stored  $x_t$ . In an *ID clustering* attack against Whānau, the Sybil nodes would set their own IDs to these immediately preceding keys, causing the lookup procedure described above to send most queries to Sybil nodes.

---

<sup>1</sup>As a reminder from Section 3.1, there are  $2m + g = \mathcal{O}(m)$  honest virtual nodes, and each virtual node can store up to  $k$  key-value pairs into the DHT.

<sup>2</sup>We’ll use the term “key table” instead of “leaves” or “successors” to avoid confusion with other DHTs (such as Chord and Pastry) whose leaf or successor tables contain pointers to other nodes, instead of key-value records. Of course, in Whānau, the value associated with a key may be the address of another node, which would have a similar effect.

Whānau defends against these clustering attacks by replicating the entire DHT routing structure into  $\ell = \mathcal{O}(\log km)$  **layers**. For each layer, every node has an independent ID, finger table, and key table; the finger tables and key tables are constructed as described above, but nodes' **layered IDs** are carefully constructed to defeat clustering attacks. Lookups can use any layer.

**Layer zero.** The  $\ell$  layers must be constructed strictly sequentially during SETUP. Layer zero, which is constructed first, is responsible for preventing key clustering attacks. Each node chooses a random key (from the set of keys inserted into the DHT) as its layer-0 ID, then constructs layer-0 finger and key tables using that ID.

This layer-0 ID distribution ensures that honest nodes evenly partition the keys stored by the system, *including* any keys inserted by the adversary. If Sybil nodes insert 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 Sybils 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.

**Layer one.** In contrast with honest IDs, the Sybil IDs stored in honest nodes' finger tables may be arbitrarily clustered. 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.

Layers one through  $\ell - 1$  are responsible for mitigating ID clustering attacks. Once layer zero's finger tables are completely constructed, SETUP can begin to construct layer one.<sup>3</sup> To pick a layer-1 ID, each node chooses a random entry from its own layer-0 finger table and uses that node's ID.

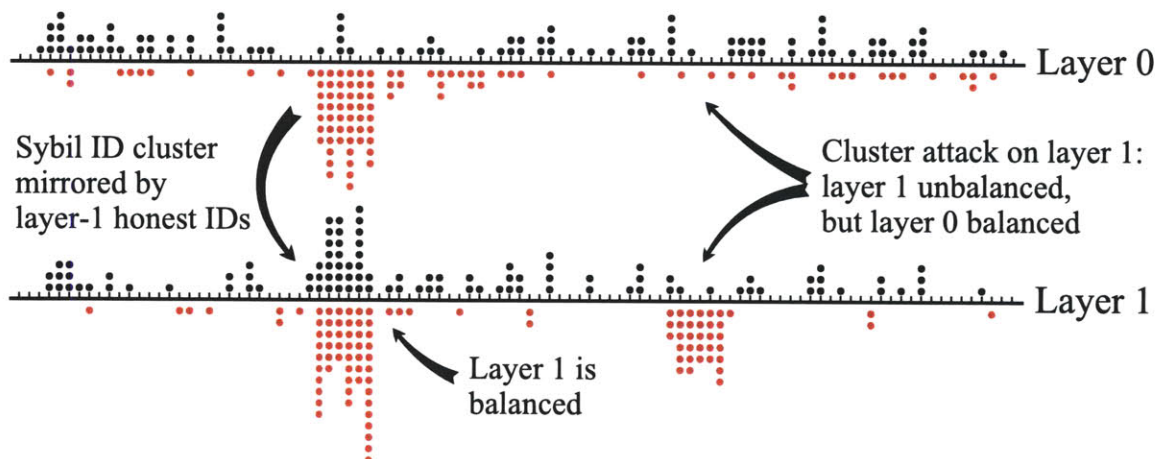
If the adversary manages to supply an honest node  $u$  with many clustered layer-0 IDs, then this increases the probability that  $u$  will pick one of these clustered IDs as its own layer-1 ID. As illustrated in Figure 5-1, this has the aggregate effect that the distribution of honest layer-1 IDs tends to mimic any clusters of Sybil layer-0 IDs. As a consequence, layer-1 finger tables are more likely to be balanced between honest and Sybil IDs.

The adversary may choose to cluster his layer-1 IDs but not his layer-0 IDs; in this case, the unclustered honest layer-1 IDs will be swamped by the Sybil IDs. However, this adversary strategy means that layer zero will be balanced between honest and Sybil IDs, so lookups using layer zero will still succeed.

**Layers two through  $\ell - 1$ .** The optimal clustering strategy against two-layer Whānau is to cluster a little bit in layer zero and to cluster heavily in layer one. More specifically, if the adversary has 100 Sybil IDs at his disposal, he should cluster  $10 = \sqrt{100}$  of them in layer zero, and all 100 of them in layer one. This yields a similar ratio of Sybil to honest IDs in each layer. The Sybil ratio is smaller than the ratio without layers, but still grows with the size of routing tables.

---

<sup>3</sup>To prevent ID clustering attacks, it's critical that nodes do not add more entries to layer-zero finger tables after layer-one IDs are chosen. If it were otherwise, the adversary could simply delay his attack until it was too late for honest nodes to react to it. However, it's perfectly safe to interleave the construction of different layers' key tables.



**Figure 5-1:** Honest IDs (black dots) in a typical layer-0 finger table are uniformly distributed over the set of keys (X axis), while Sybil IDs (red dots) may cluster arbitrarily. Honest nodes choose their layer-1 IDs from the set of all layer-0 IDs (honest and Sybil). Thus, a clustering attack on layer 0 will create a counterbalancing cluster of honest IDs in layer 1. The same property holds for layers 2 through  $\ell - 1$  (not shown). In any range of keys, at least half of the layers will always have a rough balance between Sybil and honest IDs, making it possible to reliably find an honest finger.

Increasing the number of layers from two to  $\ell = \mathcal{O}(\log km)$  solves this problem, reducing the maximum ratio of Sybil to honest IDs to the  $\mathcal{O}(1)$  required for Whānau to provide constant-time lookups. SETUP builds the layers in strict sequence, from layer zero to layer  $\ell - 1$ . As with layer one, each subsequent layer's IDs are picked randomly from the previous layer's finger tables, and then the layer's finger and key tables are constructed. Once all layers are complete, SETUP exits. Since most layers will have a low ratio of Sybil to honest IDs in every range, LOOKUP can use a random sampling procedure (described in Section 5.3) to find honest fingers preceding any target key.

### 5.1.3 Handling failures and churn

Under normal, non-attack conditions, there are three sources of churn that Whānau must handle. We call these phenomena *node churn*, *social churn*, and *key churn*.

**Node churn.** Computers may become temporarily unavailable due to network failures, mobility, overload, crashes, or a nightly-shutoff schedule. Whānau builds substantial redundancy into its routing tables to handle Sybil attacks, and this same redundancy is sufficient to handle temporary node failures. Section 8.5 shows that increasing node churn results in a modest additional overhead.

**Social churn.** The second source of churn is changes to participants' social relationships which result 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, the analysis would categorize it 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 Misllove *et al.* indicates that the Flickr social network's doubling time was between 5 and 10 months [82, 83]. This suggests that running `SETUP` every week, or every hour, would keep Whānau closely in sync with the current social graph.

**Key churn.** The final and most challenging source of churn is changes to the set of keys stored in the DHT. This causes the distribution of keys in put-queues to drift out of sync with the distribution of finger IDs.

If Whānau were to react immediately to key insertions (Section 9.7.2), then an adversary with many attack edges could flood the DHT with clustered keys to create “hot spots” in honest nodes' key tables. This would prevent newly-inserted honest keys from becoming available until `SETUP` repartitioned the stored keys by rebuilding the routing tables.

Key churn creates 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 always make inserted keys visible immediately, since they do not worry about clustering attacks. Section 9.7.2 outlines techniques to improve Whānau's responsiveness to key churn.

Unlike key churn, turnover of values does not present a challenge for Whānau: updates to the value associated with a key may always be made immediately visible. For example, in the IM application (Section 3.1.1), 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; they are simply “carried along” with the keys in the key tables. Section 9.7.1 shows how to implement an `UPDATE` protocol for Whānau.

For some applications — like the IM example, in which each node only ever stores one key — the delay from a key being inserted to the key becoming visible is not a problem, as long as tables are refreshed hourly or daily. Other applications may have application-specific solutions. For example, a filesystem using Whānau for durable long-term storage might maintain a log of pointers to recently-inserted blocks in the mutable root record. Because this log would be cleared after every `SETUP` run, it would never grow to be very large.

## 5.2 Setup

The `SETUP` procedure takes each node's social connections and the local key-value records to store as inputs, and constructs the routing tables stored on each node (Table 5.1).

Table	Pseudocode	Type	Description
Layered IDs	$u.ids[i]$	random <i>key</i>	$u$ 's layer- $i$ ID
Finger table	$u.fingers[i]$	( <i>id</i> , <i>address</i> )	$u$ 's layer- $i$ fingers, sorted by <i>id</i>
Key table	$u.keys[i]$	( <i>key</i> , <i>value</i> )	successor records of $u.ids[i]$
Intermediate table	$u.intermediate$	( <i>key</i> , <i>value</i> )	random sample used to build <i>keys</i>

**Table 5.1:** Routing tables used by SETUP and stored on each node  $u$ . After SETUP completes, the *fingers* and *keys* tables are stored and used by LOOKUP, and the *ids* and *intermediate* tables may be discarded.

## 5.2.1 Protocol operation

The SETUP protocol proceeds in synchronized lock-step. In each step, every node samples  $\mathcal{O}(\sqrt{km})$  other nodes from the social network by taking  $\mathcal{O}(\sqrt{km})$  parallel random walks. It sends a (step-specific) request to each of the sampled nodes in parallel, and uses the replies to build a portion of its routing tables. Nodes must complete a given step's tables before responding to any requests from the next step. Thus, Whānau tolerates up to one step-time of clock skew between honest nodes.

In the first step, a node samples  $\mathcal{O}(\sqrt{km})$  random nodes (using random walks) and requests a random record from each node's put-queue. It collects all of these records into an *intermediate* table, which it sorts by key and stores for later use by the following steps.

In the second step, a node constructs its layer zero routing tables. It first assigns its own layer-0 ID,  $x$ , by choosing a random key from its *intermediate* table. Next, it samples  $\mathcal{O}(\sqrt{km})$  random nodes and requests each node's layer-0 ID, collecting the resulting (*id*, *address*) pairs into its layer-0 finger table. Finally, it samples  $\mathcal{O}(\sqrt{km})$  additional random nodes and requests, from each one, the *first* record following  $x$  in the sampled node's *intermediate* table. It collects the replies into its layer-0 key table.

In the third and all subsequent steps, a node sequentially constructs its routing tables for layers one through  $\ell - 1$ . To take layer one as an example, each node assigns its own layer-1 ID by choosing a random key from its layer-0 finger table. It then constructs its layer-1 finger and key tables exactly as above, using layer-1 IDs instead of layer-0 IDs.

Subsequent steps build the remaining layers in the same way as layer one was built; each layer's IDs depend on the previous layer. Once all layers have been constructed, SETUP sorts all of a node's finger tables and key tables, and stores them for later use by LOOKUP. The *intermediate* table may be discarded.

## 5.2.2 Pseudocode

Figure 5-2 restates the above algorithm for SETUP using pseudocode. A call to METHOD on Whānau node  $u$  is represented by the notation  $u.METHOD(arguments)$ ; if the callee is a different node from the caller, this is a remote procedure call. Since an RPC to a Sybil node may result in Byzantine behavior, RPCs should validate return values and have timeouts.

The RANDOM-CHOICE subroutine returns a random element of the given set. The RANDOM-WALK subroutine takes a random walk on the social network and returns the



	Range	Description
<i>neighbors</i>	list of addresses	<i>u</i> 's social connections
<i>putqueue</i>	list of ( <i>key</i> , <i>value</i> )	records to insert into the DHT
<i>w</i>	$\mathcal{O}(\log n)$	estimate of social graph's mixing time
<i>ℓ</i>	$\mathcal{O}(\log r_f)$	number of layers
<i>r<sub>f</sub></i>	$\mathcal{O}(\sqrt{km})$	number of fingers (per layer)
<i>r<sub>k</sub></i>	$\mathcal{O}(\sqrt{km})$	number of records stored in key table (per layer)
<i>r<sub>i</sub></i>	$\mathcal{O}(\sqrt{km})$	number of records in intermediate table (total)
<i>T<sub>0</sub></i>	clock time	start time of next SETUP round
$\Delta T$	milliseconds to minutes	protocol step time (at least RTT + clock skew)

**Table 5.2:** Parameters passed to *u*.SETUP by the application.

final node. The same SAMPLE-RECORDS subroutine as in Section 4.2 is used to construct Whānau's *intermediate* tables.

The CHOOSE-ID, FINGERS, and SLICE subroutines build each layer's routing tables. FINGERS builds a finger table using random walks and the GET-ID RPC, and SLICE builds a key table using random walks and the SLICE-SAMPLE RPC.

SETUP's parameters are described in Table 5.2. The values  $\ell$ ,  $r_f$ ,  $r_k$ , and  $r_i$  determine the sizes of the routing tables; typically, nodes will have a fixed bandwidth and storage budget to allocate amongst the tables. Chapters 6 and 8 show how varying these parameters impacts Whānau's performance.

The  $\ell + 1$  synchronized protocol steps are scheduled using the parameters  $(T_0, \Delta T)$ . Increasing  $\Delta T$  makes SETUP tolerate more clock skew but take longer to complete.

### 5.2.3 Informal correctness argument

Each intermediate table is a random sample of the set of key-value records stored into the DHT. The intermediate tables are designed to replicate honest records widely: they have the good property that each record in an honest node's put-queue becomes frequently represented in other honest nodes' intermediate tables.

As explained above in Section 5.1.2, layer-0 IDs and fingers are chosen randomly in order to evenly partition the honest records, and higher-layer IDs and fingers are chosen to defeat ID clustering attacks. The recursive construction of layered IDs and finger tables ensures that, in the majority of layers, finger tables will be balanced between honest and Sybil nodes.

Once a node has its ID for a layer, it must collect the key table for that ID — that is, a **slice** of successor keys (and their values) immediately following the ID. It might seem that we could solve this the same way Chord does, by using LOOKUP to find the ID's first predecessor node, then asking that node for its own successor table. However, as pointed out in Section 1.2.2, this bootstrapping approach would enable the adversary to use an Eclipse attack to fill up the routing tables with bogus records over time. To avoid this, Whānau builds each node's key table without using any other node's key table; instead, it uses only the intermediate tables.

```

u.SETUP( $\{nbr_1, \dots, nbr_d\}, \{(key_1, value_1), \dots, (key_k, value_k)\}; w, \ell, r_f, r_k, r_i, T_0, \Delta T$ )
1   $u.neighbors \leftarrow \{nbr_1, \dots, nbr_d\}$ 
2   $u.putqueue \leftarrow \{(key_1, value_1), \dots, (key_k, value_k)\}$ 
3  sleep until  $T_0$ 
4   $u.intermediate \leftarrow u.SAMPLE-RECORDS(w, r_i)$ 
5  for  $i \leftarrow 0$  to  $\ell - 1$ 
6      do sleep until  $T_0 + (i + 1)\Delta T$ 
7           $u.ids[i] \leftarrow u.CHOOSE-ID(i)$ 
8           $u.fingers[i] \leftarrow u.FINGERS(i; w, r_f)$ 
9           $u.keys[i] \leftarrow u.SLICE(i; w, r_k)$ 
10 return  $u.neighbors, u.putqueue, u.fingers, u.keys$ 

u.SAMPLE-RECORDS( $w, r_i$ )
1  for  $j \leftarrow 1$  to  $r_i$ 
2      do  $v_j \leftarrow u.RANDOM-WALK(w)$ 
3           $(key_j, value_j) \leftarrow v_j.SAMPLE-RECORD()$ 
4  return  $\{(key_1, value_1), \dots, (key_{r_i}, value_{r_i})\}$ 

u.RANDOM-WALK( $w$ )
1  if  $w > 0$ 
2      then  $v \leftarrow RANDOM-CHOICE(u.neighbors)$ 
3          return  $v.RANDOM-WALK(w - 1)$ 
4  else return  $u$ 

v.SAMPLE-RECORD()
1   $(key, value) \leftarrow RANDOM-CHOICE(v.putqueue)$ 
2  return  $(key, value)$ 

```

**Figure 5-2:** SETUP procedure to build structured routing tables. The SAMPLE-RECORDS subroutine constructs the *intermediate* tables using the RANDOM-WALK and SAMPLE-RECORD remote procedure calls.

```

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

u.FINGERS(i; w, rf)
1  for j ← 1 to rf
2    do vj ← u.RANDOM-WALK(w)
3    xj ← vj.GET-ID(i)
4  return {(x1, v1), ..., (xrf, vrf)}

v.GET-ID(i)
1  return v.ids[i]

u.SLICE(i; w, rk)
1  for j ← 1 to rk
2    do vj ← u.RANDOM-WALK(w)
3    Rj ← vj.SLICE-SAMPLE(u.ids[i])
4  return R1 ∪ ... ∪ Rrk

v.SLICE-SAMPLE(x0)
1  {(key1, value1), ..., (keyri, valueri)} ← v.intermediate
   (sort records so that  $x_0 \preceq key_1 \preceq \dots \preceq key_{r_i} \prec x_0$ )
2  return {(key1, value1), ..., (keyt, valuet)} (for some small t, may be 1)

```

**Figure 5-2:** SETUP subroutines, continued. The local CHOOSE-ID, FINGERS, and SLICE subroutines construct the layer-*i* *ids*, *fingers*, and *keys* tables using the RANDOM-WALK, GET-ID, and SLICE-SAMPLE remote procedure calls.

Because any layered ID's successor keys are spread randomly around the intermediate tables of many other nodes, the SLICE subroutine must contact many random nodes and collect little bits of the key table together. The straightforward way to do this is to ask each node  $v$  for the closest few records in  $v.intermediate$  following the ID. Each such request accumulates a few more potential-successors; the precise number  $t$  of records sampled does not matter for correctness, as long as  $t$  is small compared to  $r_i$ . Section 6.2's analysis simply lets  $t = 1$ .

This successor sampling technique ensures that, for appropriate values of  $r_i$  and  $r_k$ , the union of the repeated SLICE-SAMPLES will contain all the needed key-value records. Section 6.2 will state this quantitatively, but the intuition is as follows. Each SLICE-SAMPLE request independently and randomly samples the set of keys in the system which are near the given ID. There may be substantial overlap in the result sets, but after making sufficiently many requests, the node will eventually receive all the keys in the ID's slice. Some of the keys returned will be far away from the ID and thus not actually useful successors, but they will show up only a few times. Likewise, bogus results returned by Sybil nodes may consume some storage space, but do not affect correctness, since they do not prevent honest nodes from finding the true successor records.

## 5.3 Lookup

The LOOKUP procedure takes a key and a set of routing tables as inputs, and finds an honest finger node with the target key in its local key table. It can then query and return the value associated with the key.

### 5.3.1 Protocol operation

LOOKUP uses the routing tables constructed by the last run of SETUP. To look up a key, a node first checks whether it is in the local put-queue, and returns immediately if it is. Otherwise, the node searches its local layer-0 finger table for the target key's first predecessor finger, with ID  $x_0$ .

Next, the node chooses one of its  $\ell$  layers at random, and constructs the set of that layer's finger table entries with IDs between  $x_0$  and the target key. (If the set is empty, it retries with a new random layer until it gets a non-empty set.) LOOKUP then chooses a random finger node from this set, and sends it a query message containing the target key.

If the target key is in the queried node's key table, it replies with the associated value. LOOKUP checks whether the record is authentic; if so, it returns the value to the application.

If the query times out, fails, or returns a bogus record, the original node retries the lookup. To do this, it uses a random walk to choose a delegate node, and requests the delegate to repeat the process from the beginning, using the delegate's finger tables.

Chapter 6 predicts that lookups will successfully complete using a constant number of retries. To reduce latency, a node can perform multiple independent lookups in parallel.

### 5.3.2 Pseudocode

Figure 5-3 expresses the above LOOKUP protocol in pseudocode. The outer LOOKUP procedure is responsible for retrying using random delegates; the TRY subroutine is the main algorithm. As an optimization, TRY permits multiple retries using different entries of a single delegate’s finger table. With each such retry, TRY expands the range of fingers under consideration.

TRY uses CHOOSE-FINGER to pick an appropriate finger table entry, and then sends a simple QUERY RPC to the chosen finger. CHOOSE-FINGER implements the finger selection algorithm given above. Note that there is always at least one non-empty set  $F_i$ , since  $x_0 \in F_0$  by definition.

### 5.3.3 Informal correctness argument

The basic goal of the LOOKUP procedure is to use CHOOSE-ID to find a finger node which is both honest and contains the target record. The SETUP procedure ensured that any honest finger “close enough” to the target key will have it in its key table. Since every finger table contains many random honest nodes, each node is likely to have an honest finger which is “close enough” (if the finger tables are big enough). However, if the adversary clusters its IDs near the target key, then a node might have to waste many queries to Sybil fingers before finding this honest finger. LOOKUP carefully chooses fingers to query in order to foil this category of attack.

The TRY subroutine aims to choose an “anchor”  $x_0$  which is close to the target key. If there is no ID clustering attack, then  $x_0$  is likely to be an honest ID. With  $\mathcal{O}(\sqrt{km})$  evenly-distributed honest layer-0 fingers, the closest finger  $x_0$  is probably close enough to have the target record in its key table of size  $\mathcal{O}(\sqrt{km})$ . On the other hand, if there *is* a clustering attack, the adversary can only force  $x_0$  to be *closer* to the target key than it otherwise would have been. Therefore, in *either* case, any honest finger found between  $x_0$  and the target key will be close enough to contain the target record.

One question remains: given that it is limited to picking fingers from the range  $[x_0, key]$ , how likely is CHOOSE-FINGER to pick an honest finger versus a Sybil finger? Recall from Section 5.1.2 that, during SETUP, if the adversary had clustered its layer- $i$  IDs in the range  $[x_0, key]$ , then the honest nodes would also cluster their layer- $(i+1)$  IDs in the same range. As a consequence, in the majority of layers, Sybil fingers do not dominate the range. Moreover, in a *randomly chosen* layer, Sybil fingers *probably* do not dominate the range! Thus, the random finger returned by CHOOSE-FINGER 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 to the target key. The previous run of SETUP will have ensured that the target key-value record is in that finger node’s key table. The adversary may focus its clustering attack to cause a few honest nodes to have bad finger tables, but retrying lookups using random delegate nodes negates this attack. Therefore, LOOKUP should almost always succeed after only a few calls to TRY.

```

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

v.TRY(key)
1  if exists (key, value) ∈ v.putqueue
2  then return value
3   $\{(x_1, f_1), \dots, (x_{r_f}, f_{r_f})\} \leftarrow v.fingers[0]$ 
   (sort fingers so that  $key \preceq x_1 \preceq \dots \preceq x_{r_f} \prec key$ )
4  j ← rf
5  repeat (f, i) ← v.CHOOSE-FINGER(xj, key)
6      value ← f.QUERY(i, key)
7      j ← j − 1
8  until QUERY found valid value, or hit retry limit
9  return value

v.CHOOSE-FINGER(x0, key)
1  for i ← 0 to l − 1
2      do  $F_i \leftarrow \{ (x, f) \in v.fingers[i] \mid x_0 \preceq x \preceq key \}$ 
3  i ← RANDOM-CHOICE( $\{ i \in \{0, \dots, l - 1\} \mid F_i \text{ non-empty} \}$ )
4  (x, f) ← RANDOM-CHOICE(Fi)
5  return (f, i)

f.QUERY(i, key)
1  if exists (key, value) ∈ f.keys[i]
2  then return value
3  else error “not found”

```

**Figure 5-3:** LOOKUP procedure to retrieve a record by key.

# Chapter 6

## Analysis of Whānau’s performance

For the same reason as Theseus’s protocol (Section 4.1), Whānau’s LOOKUP will always eventually succeed if the honest region is a connected graph: after many retries, some random walk (Figure 5-3, LOOKUP, line 3) will happen to land on the node with the target key in its put-queue. However, the point of Whānau’s added complexity is to improve lookup performance beyond an unstructured DHT protocol. This chapter will prove that, under the reasonable assumptions stated in Chapter 3, Whānau’s LOOKUP algorithm uses an expected  $\mathcal{O}(1)$  messages to find any target key.

### 6.1 Social network model

Before delving into the Whānau protocol specifically, we’ll first formalize and expand on the social network model outlined in Chapter 3.

The honest region of the social network is a graph with  $n$  nodes and  $m$  undirected edges. Let  $d_u$  and  $g_u$  be the number of honest and attack edges (respectively) adjacent to the node  $u$ , so that  $m = \frac{1}{2} \sum_u d_u$  and  $g = \sum_u g_u$ . The number of honest virtual nodes is  $2m + g$ .

#### 6.1.1 The sparse cut and escape probability

**Definition** (escape probability). Let  $p_u^w$  be the **escape probability** of the honest node  $u$ : i.e., the probability that a  $w$ -step random walk starting at  $u$  will cross over an attack edge into the Sybil region. It is defined by the recurrence

$$p_u^1 \stackrel{\text{def}}{=} \frac{g_u}{d_u + g_u} \quad p_u^w \stackrel{\text{def}}{=} p_u^1 + \sum_{u' \sim u} \frac{p_{u'}^{w-1}}{d_u + g_u} = \frac{1}{d_u + g_u} \left( g_u + \sum_{u' \sim u} p_{u'}^{w-1} \right)$$

A virtual node  $v$ ’s escape probability  $p_v$  is equal to its social node’s escape probability  $p_u$ .

**Theorem.** Order the honest virtual nodes  $\{v_1, \dots, v_{2m+g}\}$  so that  $p_{v_1} \geq p_{v_2} \geq \dots \geq p_{v_{2m+g}}$  (i.e., a random walk starting at  $v_1$  is the most likely to escape). For all  $i$ ,  $p_{v_i} \leq \frac{g^w}{i}$ .

*Proof.* Consider a hypothetical experiment. Draw a random social node  $u$  according to the honest region’s stationary distribution  $\pi(u) = \frac{d_u}{2m}$ . Now, take one random step on the social

network. For random  $u$ , the probability of crossing an attack edge after one step at most:

$$\sum_u p_u^1 \pi(u) = \sum_u \frac{g_u}{d_u + g_u} \frac{d_u}{2m} = \frac{1}{2m} \sum_u g_u \frac{d_u}{d_u + g_u} < \frac{1}{2m} \sum_u g_u = \frac{g}{2m}$$

Observe that if the step does not cross an attack edge, then the next node is a random honest node, exactly the same as the starting stationary distribution.

By repeating this process, we conclude that a length- $w$  random walk starting at a random honest node has at least a

$$\left(1 - \frac{g}{2m}\right)^w > 1 - \frac{gw}{2m}$$

probability of staying within the honest region, and at most a  $\frac{gw}{2m}$  probability of escaping. We can summarize this as

$$\sum_u p_u^w \pi(u) < \frac{gw}{2m}$$

and thus we have the lemma

$$\sum_u p_u^w d_u < gw$$

Now, recall the above ordering on virtual nodes  $p_{v_1} \geq p_{v_2} \geq \dots \geq p_{v_{2m+g}}$ . Since each social node  $u$  corresponds to  $d_u + g_u$  virtual nodes, we can rewrite  $p_{v_1} + p_{v_2} + \dots + p_{v_{2m+g}}$  as  $\sum_u p_u^w (d_u + g_u)$ . Using the recursive definition of  $p_u^w$  and rearranging terms yields:

$$\begin{aligned} \sum_u p_u^w (d_u + g_u) &= \sum_u \left[ g_u + \sum_{u' \sim u} p_{u'}^{w-1} \right] \\ &= g + \sum_u \sum_{u' \sim u} p_{u'}^{w-1} \\ &= g + \sum_u \sum_{u' \sim u} p_u^{w-1} \\ &= g + \sum_u p_u^{w-1} d_u \\ &< g + g(w-1) = gw \end{aligned}$$

Thus we can conclude that

$$p_{v_1} + p_{v_2} + \dots + p_{v_{2m+g}} < gw$$

and more specifically that

$$ip_{v_i} \leq p_{v_1} + \dots + p_{v_i} < gw$$

Thus we have  $p_{v_i} < \frac{gw}{i}$ . □

**Corollary.** *Since each physical social node has at least one virtual node, if we order the physical social nodes  $\{u_1, \dots, u_n\}$  so that  $p_{u_1} \geq \dots \geq p_{u_n}$ , we will still have  $p_{u_i} \leq \frac{gw}{i}$ .*



**Definition** (winners and losers). Fix an *arbitrary* constant fraction  $0 < \epsilon < 1$ . The  $\epsilon n$  nodes with highest escape probability ( $u_1, \dots, u_{\epsilon n}$  above) are the **loser nodes**, and the rest of the honest nodes are the **winner nodes**.

**Corollary.** *If  $g \ll n/w$ , then for all winner nodes  $u$ , the escape probability is small:*

$$p_u^w < \frac{gw}{\epsilon n} = \mathcal{O}\left(\frac{gw}{n}\right) \ll 1$$

Analogously, we can define winner and loser *virtual* nodes, so that there are up to  $2\epsilon m$  loser virtual nodes, and winner virtual nodes have escape probability less than  $\frac{gw}{2\epsilon m} = \mathcal{O}\left(\frac{gw}{m}\right)$ . This definition is a bit tighter, since it uses  $m$  instead of  $n$ .

Our arbitrary fraction  $\epsilon$  dividing “winners” from “losers” appears only in this analysis and not in the Whānau 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.

## 6.1.2 Variation from the uniform distribution

To be useful, the samples returned by random walks should be both honest and chosen uniformly from the virtual nodes. Thus, we are concerned with both the probability of escape into the Sybil region, and the difference between the ideal uniform distribution and the random walk’s distribution.

**Variation from the stationary distribution.** Section 3.2.3 introduced the probability distribution  $P_u^w$  of a  $w$ -step random walk starting at  $u$  and the idea of **mixing time**, defined as the smallest  $w$  such that for all  $u, u'$ :

$$\frac{1}{2}\pi(u') \leq P_u^w(u') \leq \frac{3}{2}\pi(u') \quad (6.1)$$

Of course, our real social network datasets (Section 8.1.2) contain a small number of poorly-connected nodes which do not satisfy (6.1). For the purpose of this analysis, we can simply lump these few nodes in with the losers, and make no guarantees about their performance.<sup>1</sup>

Even for the winners, this definition permits up to 50% variation from the stationary distribution. (Of course, the actual variation may be much less for many nodes.) Thus  $P_u^w$  can be decomposed into a stationary component and an arbitrary positive residue:

$$P_u^w(u') = \frac{1}{2}\pi(u') + \frac{1}{2}\hat{P}_u^w(u')$$

We can think of this as a process which flips a fair coin and then, if heads, draws from the stationary distribution  $\pi$ , or if tails, draws from an adversary-chosen distribution  $\hat{P}_u^w$ .

---

<sup>1</sup>In practice, if there are few attack edges in the vicinity, such nodes might be able to perform lookups perfectly well. If not many random walks reach them, however, they may not be able to successfully insert as many keys into the DHT.

**Stationary distribution versus virtual nodes.** Because honest nodes create virtual nodes for both honest edges and attack edges (they can't tell the difference), and the stationary distribution  $\pi$  considers only honest edges, there is a difference between the stationary distribution and the uniform distribution over honest virtual nodes.

**Definition** (obscure nodes). We will call a virtual node **obscure** if the probability of a non-escaping random walk reaching it is less than  $\frac{1}{2} \cdot \frac{1}{2m+g}$ .

If the virtual node's social node is  $u$ , this requires that

$$\frac{d_u/2m}{(d_u + g_u)/(2m + g)} < \frac{1}{2}$$

and hence requires  $d_u < g_u$ . As a consequence, the number of obscure virtual nodes is at most

$$\sum_{\{u \mid d_u < g_u\}} d_u + g_u < \sum_{\{u \mid d_u < g_u\}} 2g_u \leq 2g$$

and thus the *fraction* of obscure virtual nodes is less than  $\frac{2g}{2m+g} < \frac{g}{m}$ .

Since obscure nodes and loser nodes are correlated with clusters of attack edges, there will be a large overlap between them. However, they are not the same: an obscure node can be a winner and a loser node can be non-obscure. In Whānau, a winner node can perform efficient lookups, and a non-obscure node can reliably insert at least  $k$  key-value records per virtual node into the DHT. (An obscure node may be able to insert a smaller number of keys, or no keys at all.)

### 6.1.3 Quantifying random-walk sample quality

The above two factors — escaping walks and non-uniform distribution — represent deviations from the ideal sampling procedure, which would return a uniformly chosen random virtual node. Using the theorems we have proven, we can quantitatively bound these deviations in three situations of interest:

- A winner virtual node drawing a random non-obscure virtual node.
- A winner virtual node drawing a random non-obscure winner virtual node.
- A winner social node drawing a random non-obscure winner virtual node.

**Definition** (core nodes). We will call a non-obscure winner virtual node a **core node**. The core nodes, which constitute a large majority of the honest nodes, will carry the weight of the proof of Whānau's correctness.

**Drawing a random non-obscure virtual node.** Suppose a winner virtual node sends out a random walk. With probability at least  $1 - \frac{gw}{2\epsilon m}$ , this random walk will stay within the honest region (Section 6.1.1). According to Section 6.1.2, a non-escaping walk has a better

than  $\frac{1}{2} \cdot \frac{1}{2} \left(1 - \frac{g}{m}\right)$  chance of sampling a uniformly random non-obscure virtual node. Thus, the probability of getting a good sample in this case is at least:

$$\varphi > \frac{1}{4} \left(1 - \frac{g}{m} - \frac{gw}{2\epsilon m}\right)$$

**Drawing a random core node.** If we need a randomly chosen non-obscure *winner* sample, we must account for the samples that return loser virtual nodes. Building on the above analysis, this yields a probability of success at least:

$$\psi > \frac{1}{4} \left(1 - \frac{g}{m} - \frac{gw}{2\epsilon m} - \epsilon\right)$$

Finally, we can perform a similar analysis starting from a winner social node instead of a winner virtual node, yielding a probability of success:

$$\xi > \frac{1}{4} \left[1 - \frac{g}{m} - \frac{gw}{\epsilon n} - \epsilon\right]$$

**Good sample probability is  $\Omega(1)$**  — in other words, as long as the sparse cut and fast mixing assumptions are satisfied, the random walk sampling process returns a substantial fraction of good samples. We will be using the values  $\varphi$ ,  $\psi$ , and  $\xi$  as shorthand throughout the analysis. Naturally, these “good sample probabilities” decrease with the number of attack edges  $g$ . Their important property is that if  $g = \mathcal{O}\left(\frac{n}{w}\right)$ , then  $\varphi, \psi, \xi = \Omega(1)$ . More specifically, if  $g \ll n/w$ , then  $\varphi \gtrsim \frac{1}{4}$  and  $\psi \gtrsim \xi \gtrsim \frac{1}{4}(1 - \epsilon)$ .

The above analysis yields very conservative lower bounds on  $\varphi$ ,  $\psi$ , and  $\xi$ . In practice, the fraction of useful samples returned by random walks tends to be quite large when  $g$  is small. However, the above formulas predict that performance will rapidly degrade when  $g$  approaches  $m/w$ . The experiments in Section 8.2 confirm this prediction.

## 6.2 Winner key tables are correct

The first step of proving Whānau’s correctness is to prove that the key tables which SETUP constructs are correct: that is, they contain all honest, non-obscure key-value records immediately following the given ID.

### 6.2.1 Preliminary definitions

**Definition.** Let  $\mathcal{I}$  be the disjoint union of all the core nodes’ *intermediate* tables:

$$\mathcal{I} \stackrel{\text{def}}{=} \bigsqcup_{\text{core } v} v.\text{intermediate}$$

Intuitively, we expect non-obscure honest nodes’ records to be heavily represented in  $\mathcal{I}$ .

**Definition** (distance metric  $d_{xy}$ ). Recall from Section 5.1.1 that Whānau has no *a priori* notion of distance between two keys. However, with the definition of  $\mathcal{I}$ , we can construct

an *a posteriori* distance metric. Let  $\mathcal{I}_{xy} \stackrel{\text{def}}{=} \{z \in \mathcal{I} \mid x \preceq z \prec y\}$  be all the records (honest and Sybil) in  $\mathcal{I}$  on the arc  $[x, y)$ . Then define

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

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

## 6.2.2 SLICE-SAMPLE returns good sample records

Recall that SETUP (Figure 5-2) uses the SLICE subroutine, which calls SLICE-SAMPLE  $r_k$  times, to find all the honest records in  $\mathcal{I}$  immediately following an ID  $x$ . Consider an arbitrary successor key  $y \in \mathcal{I}$ . If the  $r_i$  and  $r_k$  parameters are sufficiently large, and  $d_{xy}$  is sufficiently small, then we will show that  $y$  will almost certainly be returned by some call to SLICE-SAMPLE. Thus, any winner node  $v$ 's table  $v.keys[i]$  will ultimately contain all records  $y$  close enough to the ID  $x = v.ids[i]$ .

**Lemma.** *Assume that  $1 \ll r_i \ll 2km$ . For any core node  $v$ , ID  $x$ , and honest, non-obscure key  $y$  such that  $d_{xy} < \frac{1}{r_i}$ ,*

$$\text{Prob}[y \in v.\text{SLICE-SAMPLE}(x)] \gtrsim \frac{\varphi r_i}{2ekm}$$

*Proof.* SLICE-SAMPLE queries  $v$  for the first key in  $v.intermediate$  following  $x$ . (For this proof, we assume that  $t = 1$ .)  $y$  will be returned if  $y$  is present in  $v.intermediate$  and if no other key between  $x$  and  $y$  is present:

$$\begin{aligned} \text{Prob}[y \in v.\text{SLICE-SAMPLE}(x)] &= \text{Prob}[y \in v.intermediate] \\ &\times \text{Prob}[\nexists z \in v.intermediate \mid x \preceq z \prec y] \end{aligned}$$

$v.intermediate$  is constructed by taking  $r_i$  random walks from  $v$  and drawing a random put-queue entry from each sample node. Each random walk's probability of sampling a uniform non-obscure node is at least  $\varphi$ ; the probability that node's put-queue contains  $y$  is at least  $\frac{1}{2m}$ <sup>2</sup>; and the probability of drawing the key  $y$  from that node's put-queue is  $\frac{1}{k}$ . Thus, each walk has a  $\frac{\varphi}{2km}$  chance of sampling  $y$ . With  $r_i \ll 2km$  walks, the probability that  $y$  is in  $v.intermediate$  is:

$$\text{Prob}[y \in v.intermediate] \gtrsim 1 - \left(1 - \frac{\varphi}{2km}\right)^{r_i} \approx \frac{\varphi r_i}{2km} \quad (6.2)$$

---

<sup>2</sup>We can approximate using  $2m$  (instead of  $2m + g$ ) non-obscure nodes here because we are assuming up to  $2g$  obscure nodes.

If  $r_i \ll 2km$ , then  $v.intermediate$  is a random sample of  $\mathcal{I}$ .<sup>3</sup> Consequently, each element of  $v.intermediate$  has a  $d_{xy} \in [0, 1)$  chance of being between  $x$  and  $y$ . Thus, the probability that  $y$  is the first element after  $x$  is:

$$\text{Prob}[\#z \in v.intermediate \mid x \preceq z \prec y] \approx (1 - d_{xy})^{r_i}$$

Under the given assumption  $d_{xy} < \frac{1}{r_i} \ll 1$ , we have

$$\text{Prob}[\#z \in v.intermediate \mid x \preceq z \prec y] \geq \left(1 - \frac{1}{r_i}\right)^{r_i} \approx e^{-1} \quad (6.3)$$

Multiplying results (6.2) and (6.3) yields the desired probability:

$$\text{Prob}[y \in v.SLICE-SAMPLE(x)] \geq \frac{\varphi r_i}{2km} \cdot e^{-1} = \frac{\varphi r_i}{2ekm} \quad \square$$

### 6.2.3 Aggregating $r_k$ samples yields complete key tables

This section's main lemma shows that  $v.keys[i]$  collects a contiguous slice of honest keys in  $\mathcal{I}$ — those with distance less than  $1/r_i$  to  $v.ids[i]$ .

**Lemma.** *Assume that  $1 \ll r_k, r_i \ll 2km$ . For any winner virtual node  $v$ , ID  $x = v.ids[i]$ , and honest, non-obscure  $y \in \mathcal{I}$  such that  $d_{xy} < \frac{1}{r_i}$ ,*

$$\text{Prob}[y \in v.SLICE(i, r_k)] \geq 1 - e^{-\frac{\varphi \psi r_k r_i}{2ekm}} \quad (6.4)$$

*Thus, after running SETUP,  $y \in v.keys[i]$  with high probability if  $r_k r_i \gg 2km$ .*

*Proof.* SLICE takes  $r_k$  walks and sends a SLICE-SAMPLE query to each sampled node. It then collects the resulting records together into the key table.

The  $r_k$  walks each return a random core node with probability  $\psi$ . Given that  $r_k \ll 2km$ , there will be very few repeats. The previous lemma showed that each SLICE-SAMPLE query to a core node has at least  $\frac{\varphi r_i}{2ekm}$  chance of returning  $y$ . Thus, the probability of not seeing  $y$  is at most:

$$\text{Prob}[y \notin v.SLICE(i, r_k)] \lesssim \left(1 - \psi \cdot \frac{\varphi r_i}{2ekm}\right)^{r_k} \approx e^{-\frac{\varphi \psi r_k r_i}{2ekm}} \quad \square$$

We can intuitively interpret this result as follows: to get a complete key table with high probability, we need  $r_k r_i = \Omega(km \log km)$ . It is no coincidence that this looks like the solution to the Coupon Collector's Problem [85] (as in Section 4.1): the SLICE subroutine "examines"  $r_k r_i$  random elements from  $\mathcal{I}$ , and it must examine the entire set of  $|\mathcal{I}| \approx 2km$  honest records to collect all the needed records.

<sup>3</sup>In addition to  $r_i \ll 2km$ , this also assumes that the adversary cannot change his responses to SAMPLE-RECORD queries after observing the partially-built intermediate tables — this would actually permit an adaptive attack in which the adversary inserted the key  $y - 1$  for every key  $y \in v.intermediate$ . However, this attack is easily prevented by hiding the intermediate tables until they are fully constructed, or by forcing all participants to commit to their keys before SETUP starts.

### 6.3 Layer-zero IDs are evenly distributed

The next lemma shows that layer-zero finger tables are correct, meaning that layer-zero finger IDs are spread uniformly over  $\mathcal{I}$ , so that any sufficiently large range of the key space will contain at least one finger. As a consequence, key clustering attacks are not effective against Whānau. The result follows naturally from the way fingers and IDs are chosen: the finger table entries are sampled randomly from the social network, and each of their layer-zero IDs is drawn randomly from the keys in  $\mathcal{I}$ .

**Lemma.** *Assume  $r_f \ll 2km$ . For any winner  $v$  and any keys  $x, y$ ,*

$$\text{Prob}[\nexists \text{core } f \in v.\text{FINGERS}(0, r_f) \mid x \preceq f.\text{ids}[0] \prec y] \lesssim (1 - \psi d_{xy})^{r_f} \quad (6.5)$$

*Thus, after running SETUP, any given range  $[x, y)$  with  $d_{xy} > \frac{1}{\psi r_f}$  is expected to contain at least one finger ID in  $v.\text{fingers}[0]$ .*

*Proof.* Each element  $f \in v.\text{FINGERS}(0, r_f)$  is a random walk sample, and thus has at least  $\psi$  probability of being a random core node. If so, then CHOOSE-ID line 2 drew  $f$ 's layer-zero ID randomly from  $f.\text{intermediate}$ . Since  $\mathcal{I}$  is the disjoint union of all core nodes' *intermediate* tables, a random core node's layer-zero ID is thus a random element of  $\mathcal{I}$ .

By its definition,  $d_{xy}$  is the probability that a random element of  $\mathcal{I}$  falls in  $[x, y)$ . Therefore, each ID in the table has an independent  $\psi d_{xy}$  chance of falling in  $[x, y)$ . Since there are  $r_f$  such IDs, the probability that *no* ID falls within the range is at most  $(1 - \psi d_{xy})^{r_f}$ .  $\square$

From this result, we can see that the larger finger tables are, the likelier that nodes can find a layer-zero finger in any small range of keys. In any winner's finger table, we expect to find approximately  $\psi r_f d_{xy}$  core fingers with IDs in the range  $[x, y)$ .

### 6.4 Layers are immune to clustering

The adversary may attack the finger tables by clustering its Sybil IDs. CHOOSE-ID line 4 causes honest nodes to respond by clustering their IDs on the same keys. The third and final main lemma will show that regardless of the adversary's ID clustering strategy, it will not be able to dominate the majority of layers.

**Preliminary definitions.** Assume a given  $r_f \gg 1$ . Fix any two keys  $x, y$  which are sufficiently far apart that we expect at least one layer-zero finger ID in  $[x, y)$  with high probability (Section 6.3). Let  $\gamma_i$  ("good fingers") be the average (over core nodes' finger tables) of the number of honest core fingers with layer- $i$  IDs in  $[x, y)$ . Likewise, let  $\beta_i$  ("bad fingers") be the average number of Sybil fingers in  $[x, y)$ .

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

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

*Proof.* The average core node's finger table contains at least  $\psi r_f$  random core nodes.<sup>4</sup> Each of these core nodes drew its layer- $(i + 1)$  ID from its own layer- $i$  finger table. (CHOOSE-ID, line 4.) By the definitions of  $\gamma_i$  and  $\beta_i$ , a random layer- $i$  ID from a random core node's finger table falls in  $[x, y)$  with probability at least  $\frac{\gamma_i + \beta_i}{r_f}$ . Thus, the average number of good fingers with layer- $(i + 1)$  IDs in  $[x, y)$  is

$$\gamma_{i+1} \geq \frac{\gamma_i + \beta_i}{r_f} \cdot \psi r_f = \psi(\gamma_i + \beta_i) \quad \square$$

**Corollary.** Let the density  $\rho_i$  of good fingers in layer  $i$  be  $\rho_i \stackrel{\text{def}}{=} \frac{\gamma_i}{\gamma_i + \beta_i}$ . Then

$$\prod_{i=0}^{\ell-1} \rho_i \geq \frac{\psi^{\ell-1}}{r_f}$$

*Proof.* By the previous lemma,  $\rho_i \geq \psi \frac{\gamma_{i-1} + \beta_{i-1}}{\gamma_i + \beta_i}$ . Cancelling numerators and denominators,

$$\prod_{i=0}^{\ell-1} \rho_i \geq \frac{\gamma_0}{\gamma_0 + \beta_0} \cdot \psi \frac{\gamma_0 + \beta_0}{\gamma_1 + \beta_1} \cdot \psi \frac{\gamma_1 + \beta_1}{\gamma_2 + \beta_2} \cdots \psi \frac{\gamma_{\ell-2} + \beta_{\ell-2}}{\gamma_{\ell-1} + \beta_{\ell-1}} = \psi^{\ell-1} \frac{\gamma_0}{\gamma_{\ell-1} + \beta_{\ell-1}}$$

Because each finger table can contain no more than  $r_f$  fingers,  $\gamma_{\ell-1} + \beta_{\ell-1} \leq r_f$ . Also, since we assumed the range is big enough to contain at least one core layer-zero finger,  $\gamma_0 \geq 1$ . Therefore,  $\prod \rho_i \geq \psi^{\ell-1} \frac{1}{r_f}$ .  $\square$

Because the density of winner fingers  $\rho_i$  is bounded below, this result means that the adversary's scope to affect  $\rho_i$  is limited. The adversary may strategically choose any values of  $\beta_i$  between zero and  $(1 - \psi)r_f$ . However, the adversary's strategy is limited by the fact that if it doubles  $\beta_i$  in order to halve the density of good nodes in layer  $i$ , this will necessarily cause the density of good nodes in layer  $i + 1$  to double.<sup>5</sup>

**Lemma.** Given  $r_f \gg 1$ , the average layer's density of winner fingers is at least

$$\bar{\rho} \stackrel{\text{def}}{=} \frac{1}{\ell} \sum_{i=0}^{\ell-1} \rho_i \geq \frac{\psi}{e} (\psi r_f)^{-\frac{1}{\ell}}$$

*Proof.* By multiplying out terms,  $(\sum \rho_i)^\ell > \ell! \prod \rho_i$ . Substituting in Stirling's approximation  $\ell! > \left(\frac{\ell}{e}\right)^\ell$  and the above lemma's lower bound for  $\prod \rho_i$  yields

$$\left(\sum \rho_i\right)^\ell \geq \left(\frac{\ell}{e}\right)^\ell \frac{\psi^{\ell-1}}{r_f}$$

<sup>4</sup>More precisely, by the Chernoff bound, the variable deviates little from  $\psi r_f$ , and with very high probability is greater than  $\frac{1}{2}\psi r_f$ . Substituting the latter expression doesn't change the analysis.

<sup>5</sup>It turns out that the adversary's optimal strategy is to attack all layers equally, starting with a small degree of clustering  $\beta_0$  and increasing  $\beta_i$  by a constant factor for each successive layer.

Thus,

$$\sum_{i=0}^{\ell-1} \rho_i \gtrsim \ell \cdot \frac{\psi}{e} (\psi r_f)^{-\frac{1}{\ell}} \quad \square$$

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

$$\ell = \log \psi r_f \quad (6.6)$$

Beyond this point, increasing the number of layers yields rapidly diminishing returns: doubling  $\ell$  improves  $\bar{\rho}$  by 65%, and doubling  $\ell$  again improves  $\bar{\rho}$  only by another 28%. Thus, for most values of  $\psi \in [0, 1]$ , the optimal  $\ell \approx \log r_f$ .

### 6.4.1 Special case: relatively weak Sybil attacks

The above analysis can be refined in the specific situation that  $g \ll \frac{m}{w \log m}$ . In this case,  $1 - \psi \ll \frac{1}{\log r_f}$ . Returning to the lemma, the number  $\gamma_i + \beta_i$  of finger IDs in a range  $[x, y]$  has three components:

1. Sybil fingers that have chosen their IDs in that range.
2. Honest non-core fingers, whose IDs may be assumed to be adversary-controlled.
3. Honest core fingers whose random IDs fell into the range.

The first two components are limited to a  $1 - \psi$  fraction of core nodes' finger table entries. Thus,  $\gamma_{i+1} + \beta_{i+1} \lesssim (1 - \psi)r_f + \psi(\gamma_i + \beta_i)$ . Expanding the recursion yields  $\gamma_{\ell-1} + \beta_{\ell-1} \lesssim \ell(1 - \psi)r_f + \gamma_0$ . If we plug this expression into the above analysis, we find that the optimal number of layers is now given by

$$\ell \approx \log \ell \psi (1 - \psi) r_f = \mathcal{O}(\log(1 - \psi) r_f),$$

a function of the average number  $(1 - \psi)r_f$  of Sybil entries in a finger table. Thus, with a small escape probability  $1 - \psi = \mathcal{O}(1/r_f^c)$ , a high density of good fingers is achieved with only  $\ell = \mathcal{O}((1 - c) \log r_f)$  layers. As a special case, if  $1 - \psi \ll 1/r_f$  (i.e., the average finger table contains less than one Sybil entry), then one layer should suffice.

## 6.5 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  $\kappa = \frac{2ekm}{\varphi\psi^2}$ . Suppose that we pick  $r_k, r_f, r_i$ , and  $\ell$  so that  $1 \ll r_k, r_f, r_i \ll km$  and (6.6), (6.7) are satisfied. Then, run SETUP to build routing tables.*

$$r_k r_f > \frac{r_k r_i}{\psi} > \kappa \quad (6.7)$$



Now let any winner node run LOOKUP on any valid key  $y$ . Then, a single iteration of TRY succeeds with probability better than  $\text{Prob}[\text{success}] > \frac{1}{20}\psi\xi = \Omega(1)$ .

The value  $\kappa$  is the aggregate storage capacity  $2km$  of the DHT times an overhead factor  $\frac{e}{\varphi\psi^2}$  which represents the extra work required to protect against Sybil attacks. When  $g = \mathcal{O}\left(\frac{m}{w}\right)$ , this overhead factor is  $\mathcal{O}(1)$ .

The formula (6.7) may be interpreted to mean that both  $r_k r_i$  and  $r_k r_f$  must be  $\Omega(\kappa)$ : the first so that SLICE-SAMPLE is called enough times to collect every successor key into the key tables, and the second so that key table slices are larger than the distance between fingers. These would both need to be true even with no adversary.

*Proof.* Suppose that the TRY request is sent to a core node; the probability of this happening on a given iteration is  $\xi$ . Let  $x \in \mathcal{I}$  be a key whose distance to the target key  $y$  is  $d_{xy} = \frac{1}{\psi r_f}$ , the average distance between core fingers.

First, substitute the chosen  $d_{xy}$  into (6.5). By the lemma, the probability that there is an honest finger  $x_h \in [x, y]$  is at least  $1 - 1/e$ . TRY line 3 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  $\frac{1}{\ell}$ , and the probability of getting an honest finger from the range is  $\rho_i$  from Section 6.4. 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 (6.6) was satisfied, Section 6.4 showed that the probability of success is at least  $\bar{\rho} \gtrsim \frac{\psi}{e^2}$ .

Finally, if the chosen finger  $f$  is honest, the only question remaining is whether the target key is in  $f$ 's key table. Substituting  $d_{x_f y} < d_{xy}$  and (6.7) into equation (6.4) yields  $\text{Prob}[y \in f.\text{keys}] \gtrsim 1 - 1/e$ . Therefore, when  $f.\text{QUERY}(y)$  checks  $f$ 's key table, it succeeds with probability at least  $1 - 1/e$ .

A TRY iteration will succeed if four conditions hold:

- (1) The TRY request is sent to a core node
- (2)  $x_f \in [x, y]$
- (3) CHOOSE-FINGER returns a winning finger  $f$
- (4)  $y \in f.\text{keys}$

Combining the probabilities calculated above for each of these events yields the total success probability

$$\xi \cdot \left(1 - \frac{1}{e}\right) \cdot \frac{\psi}{e^2} \cdot \left(1 - \frac{1}{e}\right) > \frac{1}{20}\psi\xi \quad \square$$

**Corollary.** *The expected number of queries sent by LOOKUP is bounded by  $\frac{20}{\psi\xi} = \mathcal{O}(1)$ . With high probability, the maximum number of queries is  $\mathcal{O}(\log m)$ .*

## 6.6 Routing tables are small

Any table size parameters which satisfy (6.6) and (6.7) will, by the above proof, provide fast lookups. However, it makes sense to balance the parameters to use the minimum resources to achieve a given lookup performance.

Each (virtual) node has  $S = r_i + \ell(r_f + r_k)$  table entries in total. To minimize  $S$  subject to (6.7), nodes should set  $r_k \approx r_f \approx \sqrt{\kappa}$  and  $r_i \approx \psi\sqrt{\kappa}$ . Therefore, the optimal total table size is, as expected:

$$S \approx \sqrt{\kappa} \log \kappa = \mathcal{O}\left(\sqrt{km} \log km\right)$$

As one might expect for a one-hop DHT, the optimal parameters set the finger tables and key tables to the same size. The logarithmic factor in the total table size comes from the need to maintain  $\mathcal{O}(\log km)$  layers to protect against clustering attacks. If the number of attack edges is small, Section 6.4.1 indicates that multiple layers are unnecessary. This is consistent with the experimental data in Section 8.3.

Solving for  $k$  in the above formula yields an expression for the “storage quota” per virtual node in terms of the resource budget:

$$k = \mathcal{O}\left(\frac{S^2}{m \log^2 S}\right)$$

This formula shows that the number of key-value pairs each node can store grows quadratically with Whānau’s resource budget, as long as  $S < m$ . Once every node’s finger table is large enough to store pointers to every other node, the quota only continues to grow linearly with resources, as one would expect.

# Chapter 7

## Implementations

As of this dissertation’s publication, there exist three full implementations of the Whānau protocol:

1. A distributed Python implementation used to evaluate Whānau on the PlanetLab global-scale research testbed; [31]
2. A distributed Java implementation, part of the Whānau-SIP project [40]; and
3. An abstract protocol simulator, written in C++, which evaluates large-scale protocol instances in a single machine’s physical memory.

This chapter outlines the main characteristics of the first and third implementations above. The second implementation is described in [40].

### 7.1 Distributed implementation

The first distributed Whānau implementation consists of 1,104 source lines of Python code, using the standard libraries and the RPyC [10] RPC library. Of this code, 314 lines implement the Whānau protocol, 249 lines implement RPC-related wrappers, and the remaining 541 lines implement a comprehensive test harness.

When a user starts a node, the user provides it with a list of its social neighbor nodes’ public keys and current IP addresses. The new node uses these bootstrap links to join the next scheduled SETUP round. The lock-step portion of the SETUP protocol uses message passing over UDP for each of its  $\ell+1$  phases, and the SLICE-SAMPLE and LOOKUP queries are implemented using RPC (also over UDP). Random walks for SETUP are generated using the efficient systolic mixing process described in Section 9.2, since the precise number of random walks needed can be predicted in advance. On the other hand, nodes generate random walks for LOOKUP on-the-fly by making recursive RPC queries to random social neighbors.

The Python implementation contains a trivial instant messaging (IM) client as a demonstration application. The IM client stores its current IP address into the DHT. When a user wants to send a message to another user (identified by a public key), the IM client looks up the target user’s contact information in the DHT and verifies the returned record using the

key. If the record is authentic, the IM application sends the IM body as a UDP packet to the IP address contained in the record. The IP application periodically invokes SETUP, causing Whānau to rebuild its routing tables and to incorporate nodes which join or leave.

This implementation was evaluated by running it on the PlanetLab global network testbed [31]. PlanetLab’s resources were sufficient to run up to a total of 4,000 virtual nodes on 400 physical, geographically-dispersed machines — large-scale enough to meaningfully exercise Whānau’s capabilities. Unfortunately, at scales smaller than this, the Whānau protocol’s behavior approximately reduces to simple broadcast. Given this practical limitation, the PlanetLab deployment could not be expected to produce insightful scaling results. However, it demonstrated that Whānau works on a real network with churn, varying delays, packet loss, and so on.

## 7.2 In-memory simulator

Whānau’s behavior is most interesting at large scales — with small numbers of nodes, both Whānau and simple flooding-based protocols perform adequately. Real social networks have millions of users, and social network snapshot datasets have been collected containing millions of nodes [83]. Unfortunately, PlanetLab has insufficient resources to run millions of instances of the Python implementation.

The Whānau protocol simulator addresses this problem by abstracting away all the details of network communication and synchronization, and packing a compact representation of millions of nodes’ state into a single physical machine’s RAM. It directly implements the protocol as described in Figures 5-2 and 5-3, takes a static social network as input, and provides knobs to experiment with Whānau’s different parameters. Because existing peer-to-peer simulators don’t scale to millions of nodes, this simulator was written in custom C++ (3,234 source lines of code, using Boost and STL) and implements many Whānau-specific optimizations to reduce memory consumption and running time. The most important optimizations are:

- Nodes are represented as packed data structures and network RPCs are simulated simply as C++ method calls. Messages sent by each node are counted, but underlying network latencies, bandwidth limits, congestion, and so on are not simulated.
- The social graph has a very compact memory footprint: nodes are identified with small integer indices, and social links are stored using a compressed sparse row representation.
- DHT keys are 64-bit integers instead of 20-byte cryptographic hashes.
- Finger tables, key tables, and message queues are stored using compact vectors instead of dynamic tree, hash table, or queue data structures.
- Attributes such as node IDs are only stored once, in the node, instead of being stored redundantly in every finger table pointing to that node.
- To save both space and time, nodes’ finger and key tables are generated lazily as required, instead of pre-generating all nodes’ tables.

- To save even more space, after a node’s finger and key tables are generated and used, the tables are discarded. The PRNG state used to generate the tables is memoized so that subsequent queries to the same node will re-generate the identical tables.
- Finally, the simulator implements both forward and “reverse” random walks. Reverse random walks (which are symmetric with forward random walks) are used to determine which nodes *could have* collected a given key  $x$  into their intermediate tables. This enables the simulator to avoid generating intermediate tables for those nodes which *could not* have helped to propagate  $x$  to another node’s key table, saving orders of magnitude of running time for simulating a lookup of  $x$ .

Since the simulator abstracts away so many details of the real implementation, its primary purpose is to validate the correctness, performance, and security properties of the Whānau protocol using large social network datasets, and to investigate how these properties vary with scale. The simulator’s results and the Python implementation’s results were broadly consistent at small scales. At large scales, it was not possible to cross-validate the simulator against the Python implementation, since the latter could not support so many nodes. However, as the next chapter will show, the simulator’s results were consistent with the theoretical analysis in Chapter 6.



# Chapter 8

## Simulations and experiments

This chapter investigates several empirical questions using the Whānau implementations described in the previous chapter:

1. How closely do real-world social networks match the Chapter 3’s assumptions?
2. How does Whānau’s performance change when under a denial-of-service attack?
3. Is the layered-IDs technique actually necessary for Whānau’s security?
4. How does Whānau’s performance vary with the total number of honest nodes?
5. How does Whānau’s performance behave under real wide-area network conditions?

To summarize, the results show that:

1. Real-world social networks exhibit the fast-mixing property upon which Whānau relies, and random walks are likely to return good samples.
2. The Whānau protocol can handle clustering attacks, with measured performance matching Chapter 6’s predictions for a broad range of routing table sizes and attacker strengths.
3. Layered IDs are essential for the Whānau protocol to handle clustering attacks efficiently: when under attack, setting  $\ell = 1$  causes Whānau’s performance to plummet.
4. Asymptotically, Whānau’s performance scales similarly to insecure one-hop DHTs. To maintain constant-time lookups, table sizes must scale as  $\mathcal{O}(\sqrt{n})$ .
5. When tested on PlanetLab, the Python implementation provides good lookup performance despite moderate levels of failures and churn.

It is well-established that previous structured DHTs cannot tolerate Sybil attacks in which the adversary can create many pseudonyms (*e.g.*, by generating fresh public keys). For example, Castro *et al.* introduced an important set of secure DHT routing techniques in [36]; their analysis showed that their DHT failed when more than approximately 25% of the nodes were Sybils. On the other hand, Whānau’s behavior does not change at all if the

adversary creates unlimited numbers of pseudonyms. For this reason, there isn't a natural apples-to-apples comparison of Whānau's Sybil-resistance with previous DHTs. However, it is possible to evaluate Whānau's Sybil-resistance in absolute terms by measuring the performance degradation caused by a simulated Sybil attack.

Non-adversarial settings do permit direct comparison between Whānau and other DHTs. In this case, simulations show that Whānau's performance scales like any other insecure one-hop DHT, so (ignoring constant overhead factors such as cryptography) adding Sybil-proof security is "free". Also, similar to other (non-locality-aware) one-hop DHTs, the lookup latency is one network round-trip. Of course, caveats apply: for some applications, cryptographic overhead might be significant; other applications might be affected by the mandatory delay Whānau imposes on inserting new keys into the DHT. Broadly speaking, though, the evaluation shows that Whānau's performance, even under a powerful Sybil attack, is comparable to previous one-hop DHTs' performance when not under any attack.

## 8.1 Real-world social networks fit the model's assumptions

Whānau nodes bootstrap from an externally-provided social network to build their routing tables. Chapter 6's analysis depends upon certain assumptions about this social network: it must be fast mixing, and must have a sparse cut separating the honest region from the Sybil region. This section evaluates how closely several real-world social networks conform to this model.

### 8.1.1 Large-scale data sets

All of this chapter's simulations use social network graphs extracted from Flickr, LiveJournal, YouTube, and DBLP [2, 5, 7, 15], which have also been used in other studies [83, 121]. These networks correspond to real-world users and their social connections. The Flickr, LiveJournal, and YouTube graphs represent explicitly-created friend links; the LiveJournal graph was estimated to cover 95.4% of the site's users in Dec 2006, and the Flickr graph 26.9% in Jan 2007. On the other hand, the DBLP graph represents implicit social connections: an edge in the DBLP graph signifies that two people co-authored a research paper published in a computer science journal or conference. While these four large graphs are derived from different underlying processes and exhibit a range of gross characteristics such as average degree and clustering coefficients, we will see that they all appear to be suitable inputs to the Whānau protocol.

The raw input graphs were preprocessed by discarding unconnected nodes and transforming directed edges into undirected edges. (The majority of edges were already symmetric.) The resulting graphs' basic properties are shown in Table 8.1. The node degrees follow power law distributions, with exponents between 1.6 and 2 [83].



	$n$ =#nodes	$m$ =#edges	average degree
Flickr	1,624,992	15,476,835	19.05
LiveJournal	5,189,809	48,688,097	18.76
YouTube	1,134,890	2,987,624	5.27
DBLP	511,163	1,871,070	7.32

**Table 8.1:** Properties of the input data sets.

## 8.1.2 Measuring social network mixing properties

For Whānau’s correctness, it is important that the given 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 virtual node.

**Computing the random walk distribution matrix.** To test the fast-mixing property, define the distribution  $p_{ij}$  of random walks as follows: for a given starting edge (virtual node)  $i$ , and for each ending edge (virtual node)  $j$ , define  $p_{ij}$  as the probability that a walk of length  $w$  starting at  $i$  will end at  $j$ . Ideally, the values  $p_{ij}$  should all be very close to  $\frac{1}{2m}$ , yielding a uniform distribution over edges. For short random walks (small  $w$ ), the matrix  $p_{ij}$  will always be sparse (yielding a lumpy distribution), but as the walk length  $w$  increases, the matrix will always approach the uniform ideal. However, for a fast-mixing network,  $p_{ij}$  will approximate the ideal distribution  $\frac{1}{2m}$  after only a relatively small number of random walk steps  $w = \mathcal{O}(\log n)$ .

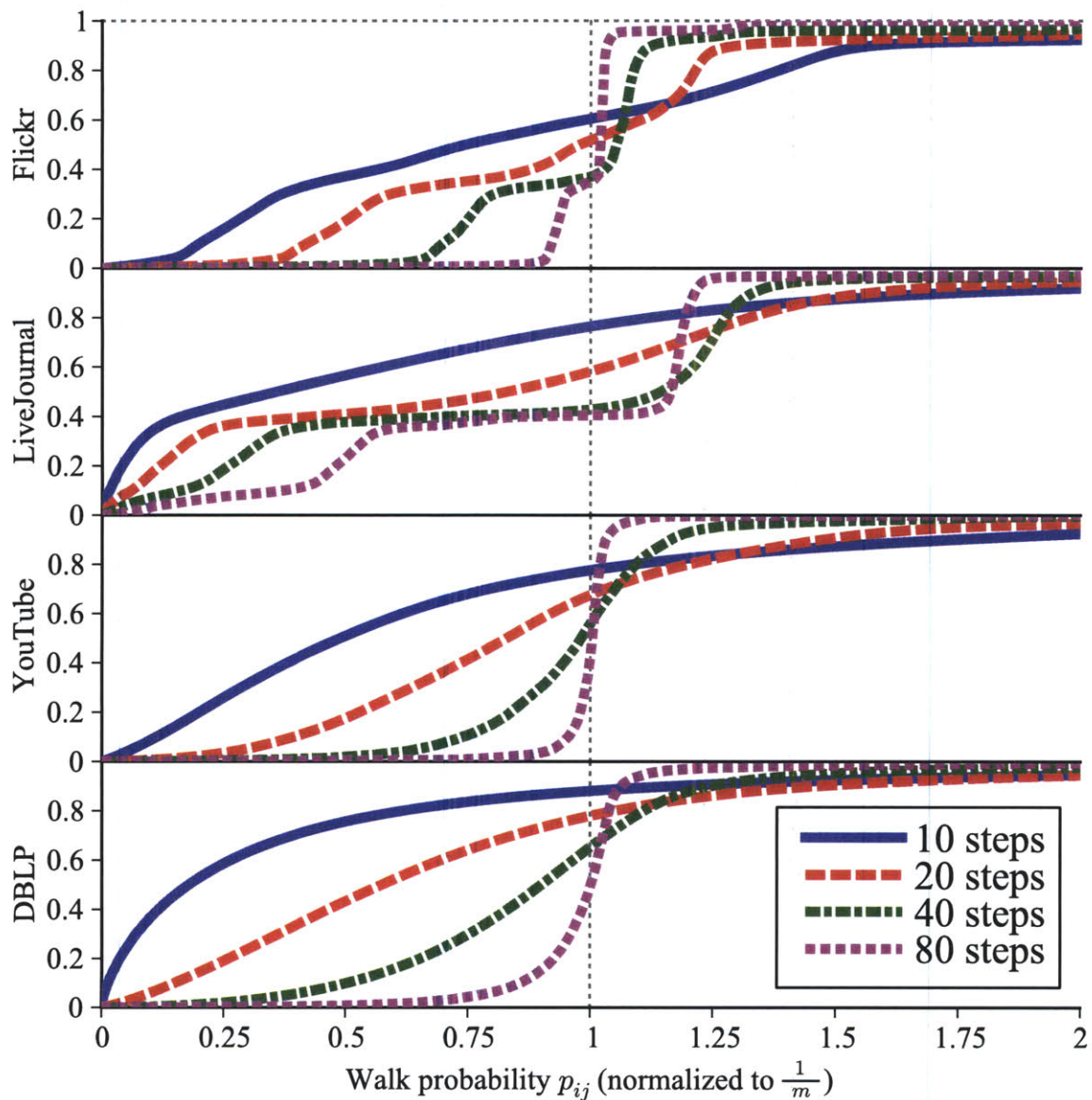
Measuring the difference between the matrix  $p_{ij}$  and the uniform matrix  $\frac{1}{2m}$  would tell us how well-mixed a random walk of length  $w$  is. Unfortunately, computing the entire matrix  $p_{ij}$  (for all  $m$  possible starting edges  $i$ ) was too expensive, requiring  $\mathcal{O}(wn^2)$  time. Instead, the matrix was approximated by sampling 100 random edges from each input graph.<sup>1</sup> For each sampled starting edge  $i$ ,  $p_{ij}$  was computed for all  $m$  end edges  $j$ .

**Results.** Figure 8-1 plots the resulting cumulative distributions of  $p_{ij}$  for increasing values of  $w$ . To compare the different social graphs, the CDFs are normalized so that they have the same mean. Thus, for all graphs,  $p_{ij} = \frac{1}{2m}$  corresponds to the ideal vertical line at 1.

As expected, as the number of steps increased from 10 to 80, each CDF approached the ideal uniform distribution. The bulk of  $p_{ij}$  values were near  $\frac{1}{2m}$ , indicating that most walks return a close-to-perfectly-random node. However, the narrow residual tails at the top and bottom of each CDF indicate that a small number of nodes are over- or under-sampled by random walks and therefore may not be able to contribute useful work for the DHT protocol. More importantly, the few nodes which are under-sampled (because they are poorly connected to the bulk of the social network) cannot reliably insert records into the DHT. Naturally, the number of such under-sampled nodes decreases with  $w$ .

The CDFs shown for  $w = 10$  are apparently far from the ideal distributions, but there are two reasons to prefer smaller values of  $w$ . First, the amount of bandwidth consumed

<sup>1</sup>The measured results were broadly similar when 100 random starting *users* (social nodes) were sampled instead of 100 random starting edges (virtual nodes).



**Figure 8-1:** Mixing properties of social graphs. Each line shows a CDF of the probability that a  $w$ -step random walk ends on a particular virtual node (social edge). The X axis normalizes the average probability  $\frac{1}{2m}$  to 1 so that social networks of various sizes can be compared directly. The level of non-uniformity of short random walks is given by the difference between each CDF and the perfectly uniform CDF (dotted vertical line). As predicted, as random walk length increases, the sampled distribution quickly becomes more uniform.

scales as  $w$ . Second, larger values of  $w$  increase the chance that a random walk will return a Sybil node. Section 8.2 will show that Whānau works well even when the distribution of random walks is not perfect: the protocol requires only that random walks hit the majority of virtual nodes with non-negligible probability. The main negative effect of a non-ideal distribution is not lookup failure, but load balance skew, since each node does an amount of work proportional to the number of random walks ending there.

### 8.1.3 Measuring escape probabilities

Recall Section 3.3's prediction that when a fast-mixing social network has a sparse cut between the honest nodes and Sybil nodes, random walks will have a low probability of escaping from the honest region, and are thus a powerful tool to protect against Sybil attacks. To confirm that this approach works with real-world social networks, the escape probability of the Flickr social network was measured with varying numbers of attack edges.

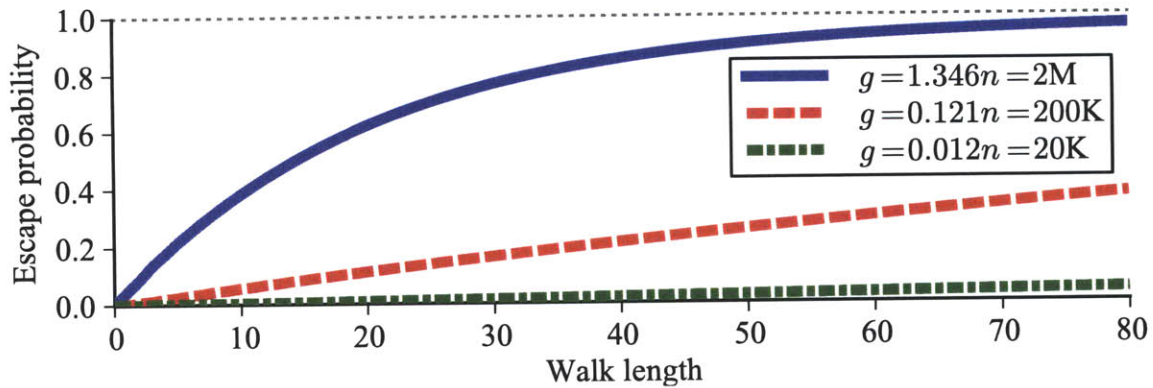
**Generating attack edges.** To generate a social network instance with  $g$  attack edges, randomly chosen honest nodes were marked as Sybils until there were at least  $g$  edges between marked nodes and non-marked nodes. Then, any honest nodes which were connected only to Sybil nodes were removed. For example, for the Flickr network, in the generated 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 (originally honest nodes in the input graph).

This algorithm for generating test instances is a good model for an adversary that is able to corrupt a random subset of honest nodes, either by exploiting bugs in users' software, or by convincing credulous users to create many social connections to Sybils. However, an artifact of the algorithm is that increasing the number of attack edges actually consumes honest nodes. As a result, it is not possible to generate test instances with  $g/m$  ratios substantially greater than 1.

The Sybil nodes generated by this algorithm tend to be distributed evenly over the social network. This might not be a realistic assumption if the attacker targeted a particular cluster of users, or if unsophisticated users naturally tend to cluster with other unsophisticated users. While social networks with clustered attack edges would exhibit different dynamics (*i.e.*, more loser nodes), the majority of winner nodes would experience much better escape probabilities than in the non-clustered case. Therefore, evenly-distributed Sybil nodes represent a conservative benchmark, demonstrating Whānau's worst-case performance.

**Results.** Figure 8-2 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 taken and with the number of attack edges. When there are few attack edges, random walks are likely to stay within the honest region.

When the number of attack edges is greater than the number of honest nodes (Figure 8-2, top line), the adversary has convinced essentially all of the system's users to form links with 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



**Figure 8-2:** Escape probability on the Flickr network.

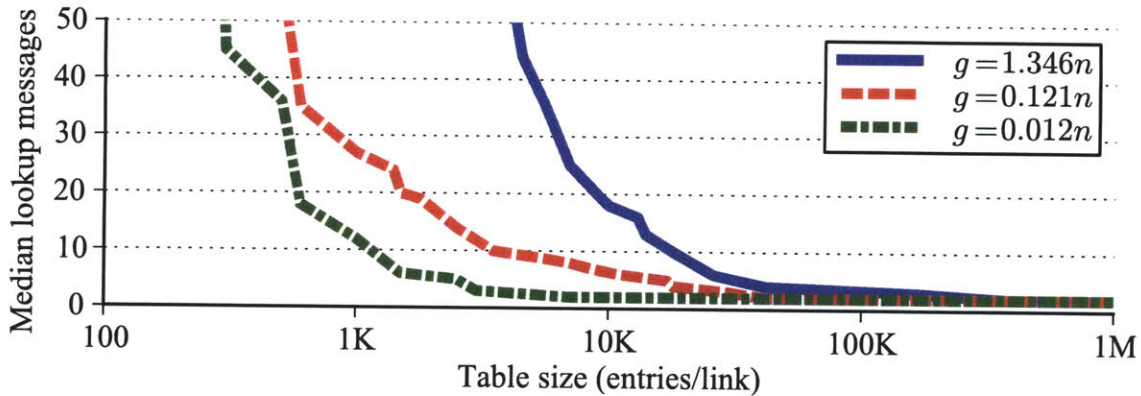
each user has an average of 1.35 links to the adversary, and random walks of length 40 are 90% Sybils. On the other hand, random walks of length 10 will return 60% honest nodes, although those honest nodes will be less uniformly distributed than a longer random walk.

## 8.2 Performance under clustering attack

To evaluate Whānau’s resistance against the Sybil attack, instances of the protocol were simulated using a range of table sizes, numbers of layers, and adversary strengths. For each instance, a random set of honest starting nodes were instructed to look up randomly chosen target keys, recording the number of messages used by the LOOKUP procedure. Chapter 6’s analysis predicted that the number of messages would be  $\mathcal{O}(1)$  as long as  $g \ll n/w$ . Since the simulations used a fixed  $w = 10$ , LOOKUP was expected to require few messages when the number of attack edges was less than 10% of the number of honest nodes. Increasing the table size was also expected to reduce the number of LOOKUP messages.

**Simulated Sybil attacker behavior.** Social network instances with varying numbers of attack edges were generated using the algorithm given in Section 8.1.3. The simulated adversary employed an ID clustering attack on the honest nodes’ finger tables, choosing all of its IDs to immediately precede the target key. It is likely that ID clustering is the most effective attack against Whānau (given a certain number of attack edges), since it attempts to fill the finger tables which honest nodes require for efficient routing with misleading information. Random walks which escape from the honest region always return Sybil identities. Simulated Sybil nodes reply to RPC requests (SAMPLE-RECORD, SLICE-SAMPLE, TRY, QUERY) with bogus data — of course, honest nodes can detect and discard these bad records, but they still waste the honest nodes’ resources.

In a real-world deployment of Whānau, it is only possible for an adversary to target a small fraction of honest keys using an ID clustering attack: in order to increase the number of Sybil IDs near a particular key, the adversary must move some Sybil IDs away from other keys. However, the Whānau simulator permits the adversary to change its IDs between *every* LOOKUP operation: that is, the simulated adversary can start over from scratch and



**Figure 8-3:** Number of messages used by LOOKUP decreases as table size increases (Flickr social network).

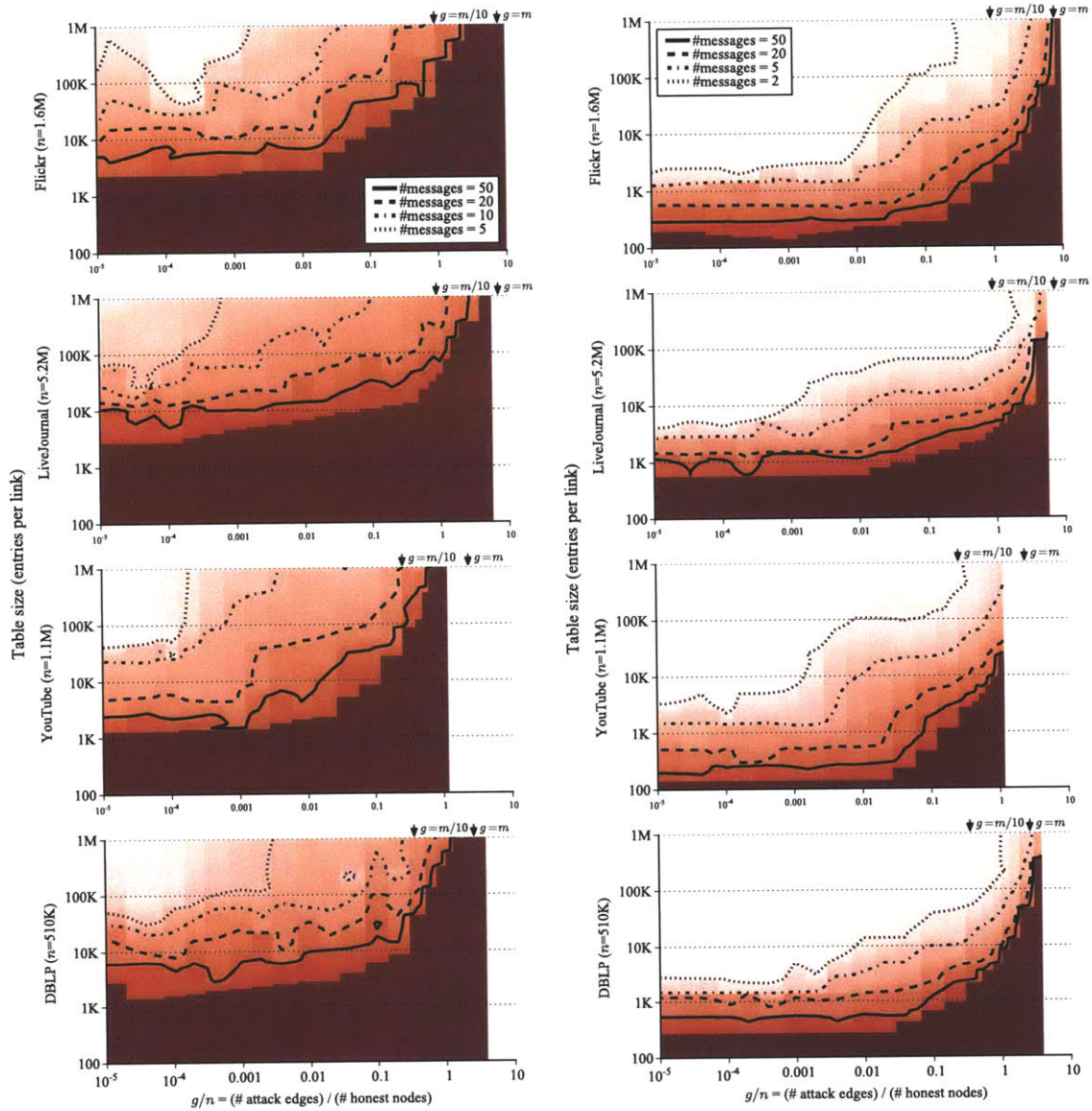
adapt its attack to the chosen target key. The results therefore show Whānau’s worst case performance, and *not* the average case performance for random target keys.

**Results.** Figure 8-3 plots the number of messages required by LOOKUP versus table size for various numbers of attack edges in the Flickr social network. The bandwidth, computation, and storage resources consumed by SETUP are proportional to a node’s routing table size; following the policy that resources scale with node degree (Section 3.3.2), table sizes are measured 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 (key and intermediate tables).

As expected, the number of LOOKUP 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. For the Flickr social network, 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  $\geq 10,000$  table entries per link.

Figure 8-4 shows the full data set of which Figure 8-3 is a narrow slice. Figure 8-4(a) shows the number of messages required for 100% of the test lookups to succeed. Of course, most lookups succeeded with far fewer messages than this upper bound. Figure 8-4(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 example, with 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.

The X-axis of each plot was normalized by the number of honest nodes in each network so that the results from different datasets could be compared directly. The theoretical analysis (Chapter 6) predicted that, in general, Whānau’s performance would drop sharply (LOOKUP messages would grow exponentially) when  $g > n/w$ . However, in all the ob-



(a) Max messages required for all lookups to succeed. (b) Median messages required for lookups to succeed.

**Figure 8-4:** 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. Arrows indicate where  $g = m/w$  and  $g = m$ ; when  $g \gg m/w$ , LOOKUP performance degraded rapidly. The plots' right edges do not line up because it was not always possible to create a social network instance with  $g = 10n$  (see Section 8.1.3).

served data, this transition actually occurred in the higher range  $m/w < g < m$ . In other words, due to the conservative approximations adopted by the analysis, the predictions were a bit too pessimistic: Whānau functioned well until a substantial fraction of all edges were attack edges.

When the number of attack edges  $g$  was below  $n/w$ , observed LOOKUP performance was more a function of table size (which must always be at least  $\Omega(\sqrt{m})$  for Whānau to function) than of  $g$ . Thus, Whānau’s performance was insensitive to relatively small numbers of attack edges.

## 8.3 Impact of layered IDs

Section 8.2 showed that Whānau handles ID clustering attacks. Much of Whānau’s complexity comes from its introduction of layered IDs; it is natural to wonder whether this new technique is indeed strictly necessary for Whānau’s attack resistance.

For the plots in Figure 8-4, several different numbers of layers were simulated for each table size, and the best-performing results were shown.<sup>2</sup> The implicit assumption is that honest nodes know *a priori* how many layers to use. Section 6.4 predicted that the optimal number of layers is a function of the chosen finger table size.

This section uses simulated data to evaluate the performance impact of layered IDs, and to investigate how nodes should choose the number of layers. The results were found to be consistent with theoretical predictions.

### 8.3.1 Layers are necessary for attack resistance

As in Section 8.2, Whānau was simulated using social networks with a range of attacker strengths. The number of layers was varied from 1 to 10 while holding the total routing table size (summed over all layers) at a constant 100,000 entries per social link, and the median number of LOOKUP messages was measured.

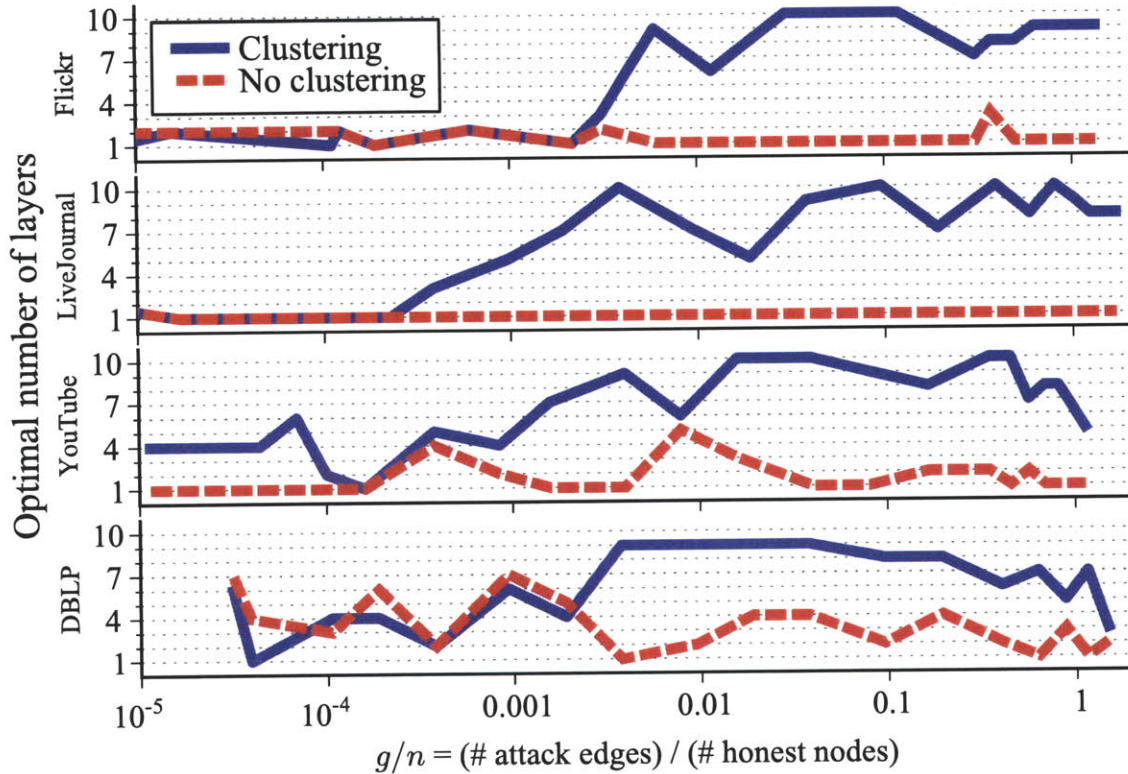
The results of this experiment show, for a given attacker strength, whether honest nodes are better off spending their available resources on more layers or on bigger per-layer routing tables. If the layered ID technique provided no benefit, we would expect Whānau to perform best with only one layer, because multiple layers come at the cost of smaller per-layer tables.

Based on the analysis in Section 6.4.1, we would expect that for small-scale attacks ( $g \ll m/w$ ), one layer is always best. For more large-scale clustering attacks, we expect more layers to be better. Even for very large-scale attacks ( $g \approx m$ ), adding more layers quickly yields diminishing returns, and so we only simulated numbers of layers between 1 and 10.

The solid lines in Figure 8-5 show the results of these simulations using the ID clustering attack described in Section 8.2. When the number of attack edges was small ( $< 0.1\%$  of  $n$ ), the smallest median number of LOOKUP messages was achieved by using small

---

<sup>2</sup>Of course, the total resources consumed by SETUP was held constant while the number of layers was varied.



**Figure 8-5:** Optimal layers versus attacker power. The resource budget was fixed at 100K table entries per link.

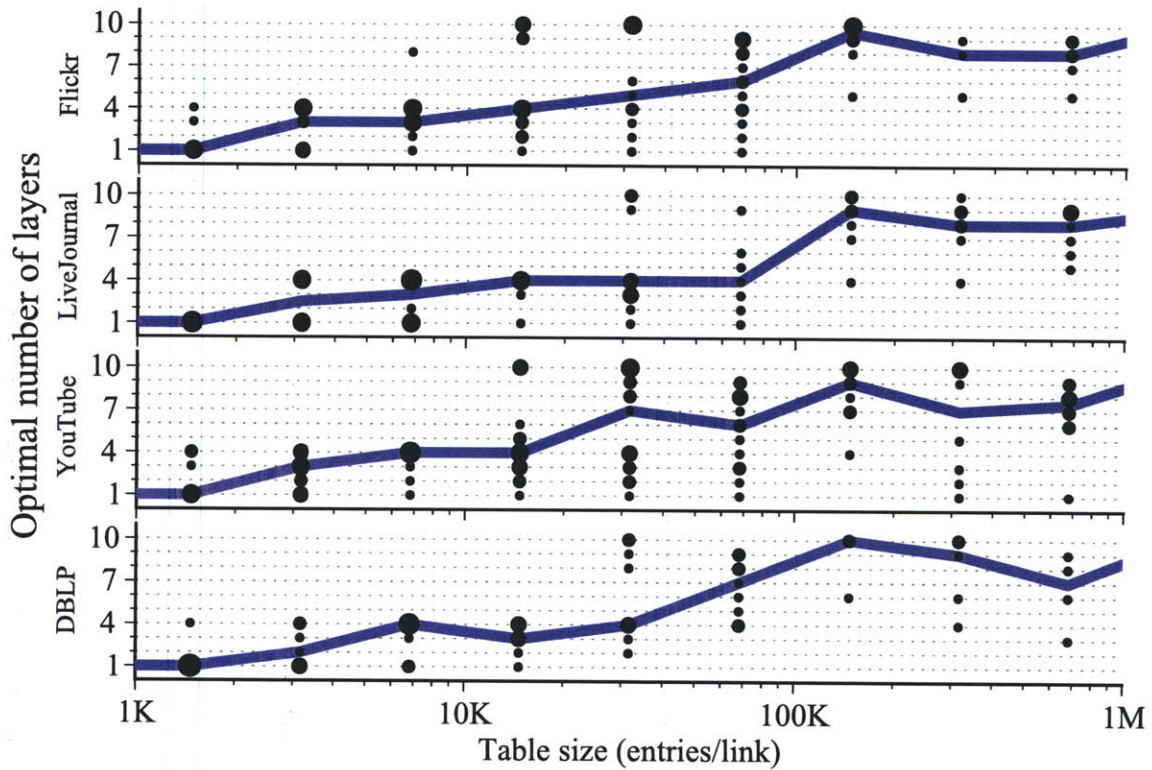
numbers of layers. In other words, weak clustering attacks can be defended against without using layers, and in this case it's best to spend all SETUP resources on bigger routing tables.

However, with larger numbers of attack edges, the data show that small numbers of layers result in many LOOKUP failures, while larger numbers of layers yield good LOOKUP performance. For example, using the Flickr social network, layers become important when the number of attack edges exceeds 5,000 (0.3% of  $n$ ); for  $g > 20,000$ , a constant number of layers (around 8) yields the best performance. At high attack ratios (around  $g/n \gtrsim 1$ ), the measured data became noisy because LOOKUP performance degraded regardless of the choice of layers.

As a control, the dashed lines in Figure 8-5 show the same Whānau 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. The difference between the solid and dashed lines clearly shows that, while multiple layers are strictly necessary to protect against powerful ID clustering attacks, they are not necessary against weak or naïve attacks.

The observed trends were clearer for the larger graphs (Flickr and LiveJournal) than for the smaller graphs (YouTube and DBLP). The experiment's fixed SETUP resource budget (100,000 table entries per virtual node) was very large in comparison to the smaller graphs' sizes, and therefore the differences in performance between small numbers of layers are not as substantial. In the raw data, the same trends were observed to hold for different fixed resource budgets.





**Figure 8-6:** Optimal layers versus resource budget. Each dot represents a table size / attacker power instance. Larger dots correspond to multiple instances with different attacker powers but the same optimal number of layers. The trend line passes through the median point for each table size; this is a reasonable choice for number of layers, given that the attacker power is unknown to honest nodes.

### 8.3.2 Optimal number of layers increases with table size

The above data showed that layers improve Whānau’s resistance against powerful ID clustering 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 have a resource cost, the optimal number of layers would be expected to depend on the honest nodes’ resource budget. If the budgeted number of table entries is large compared to  $\sqrt{km}$ , then increasing the number of layers is the best way to protect against powerful adversaries. On the other hand, if the budgeted number of table entries is relatively small, then no number of layers can protect against a powerful attack; thus, nodes should choose a smaller number of layers to reduce overhead and perform well in the case of no attack or a weak attack.

This hypothesis was tested by re-analyzing the data collected for Section 8.2. For each given total table size, the optimal number of layers (yielding the lowest median LOOKUP messages) was computed over a range of attack strengths. The results are shown in Figure 8-6. Each dot represents a simulated instance with a different number of attack edges.

The overall trend is clear: at small SETUP resource budgets, optimal performance was achieved with fewer layers. At large SETUP resource budgets, more layers was better.

At intermediate resource budgets, the optimal number of layers varied strongly with the number of attack edges  $g$ ; given that  $g$  is unknown to honest nodes, it is reasonable to choose a moderate number of layers. The median trend lines shown in Figure 8-6 appear to be consistent with the predicted logarithmic relationship (Section 6.4) between finger table size (roughly proportional to SETUP resource budget) and optimal number of layers.

Since honest nodes certainly know their own finger table size  $r_f$ , it's reasonable to set the number of layers as a function of  $r_f$ . This may not be perfectly optimal for every attacker strength  $g$ ; however, Whānau's performance is not very sensitive to small changes in the number of layers. Thus, a rough estimate is sufficient to get good performance over a wide range of situations.

## 8.4 Performance versus number of nodes

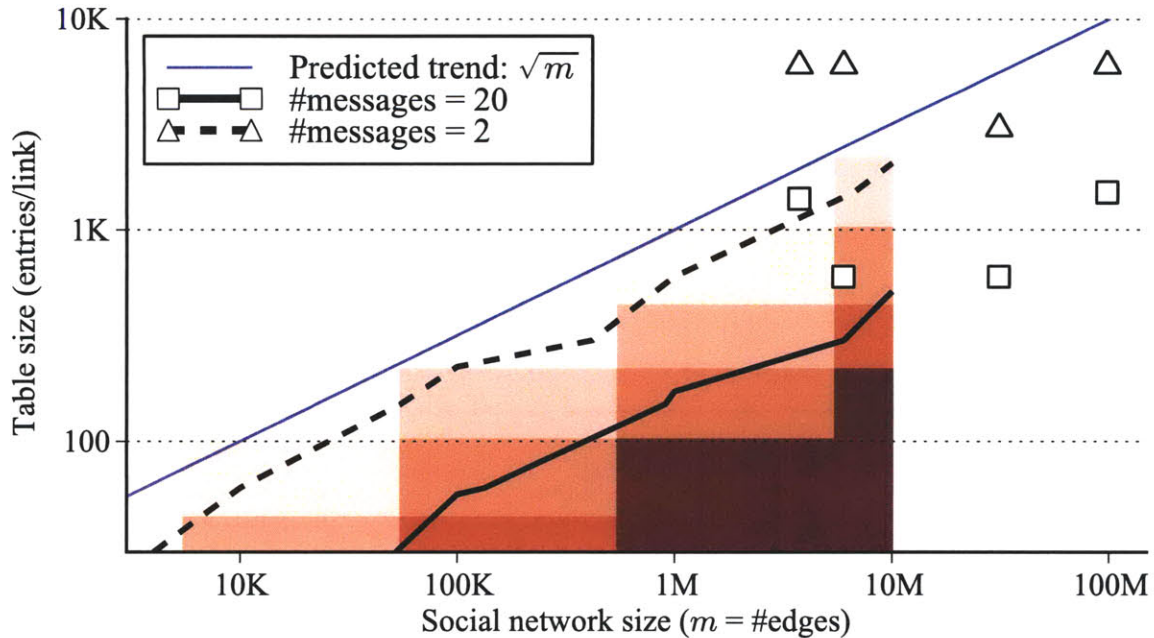
Whānau was designed as a one-hop DHT; it is thus important to confirm that its performance scales similarly to insecure one-hop DHTs such as Kelips [63]. The real social network datasets introduced in Section 8.1.1 don't cover a wide range of different sizes. To overcome this limitation, synthetic social networks with varying numbers of nodes were generated using the standard technique of preferential attachment [29], yielding power-law degree distributions with average degree 10 and exponents close to 2. For each network, the Whānau protocol was simulated for various table sizes and layers, as in the preceding sections. Since the goal was to demonstrate that Whānau reduces to a standard one-hop DHT in the non-adversarial case, there was no simulated attack.

Figure 8-7 plots the median number of LOOKUP messages versus SETUP resource budget and social network size. For any one-hop DHT, we would expect that, to hold the number of LOOKUP messages to a constant  $\mathcal{O}(1)$ , the required table size should scale as  $\mathcal{O}(\sqrt{m})$ : the blue line shows this predicted trend. The heat map and its contours (black lines) show simulated results for the 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 the four real-world datasets alongside the synthetic networks for comparison. While each real network has idiosyncratic features of its own, it is clear that the required table sizes follow the  $\mathcal{O}(\sqrt{m})$  scaling trend expected of a one-hop DHT.

These simulations confirm that Whānau's asymptotic scaling behavior is the same as previous one-hop DHTs in the non-adversarial case. However, in the adversarial case, there are two important caveats:

- Layers will be required to protect against clustering attacks, increasing the required table size from  $\mathcal{O}(\sqrt{m})$  to  $\mathcal{O}(\sqrt{m} \log m)$ .
- Cryptographic authentication will be required to preserve integrity, possibly increasing the constant factor hidden in the big-O notation.

This means that Whānau's scaling in Sybil scenarios differs (by an  $\mathcal{O}(\log m)$  factor) from



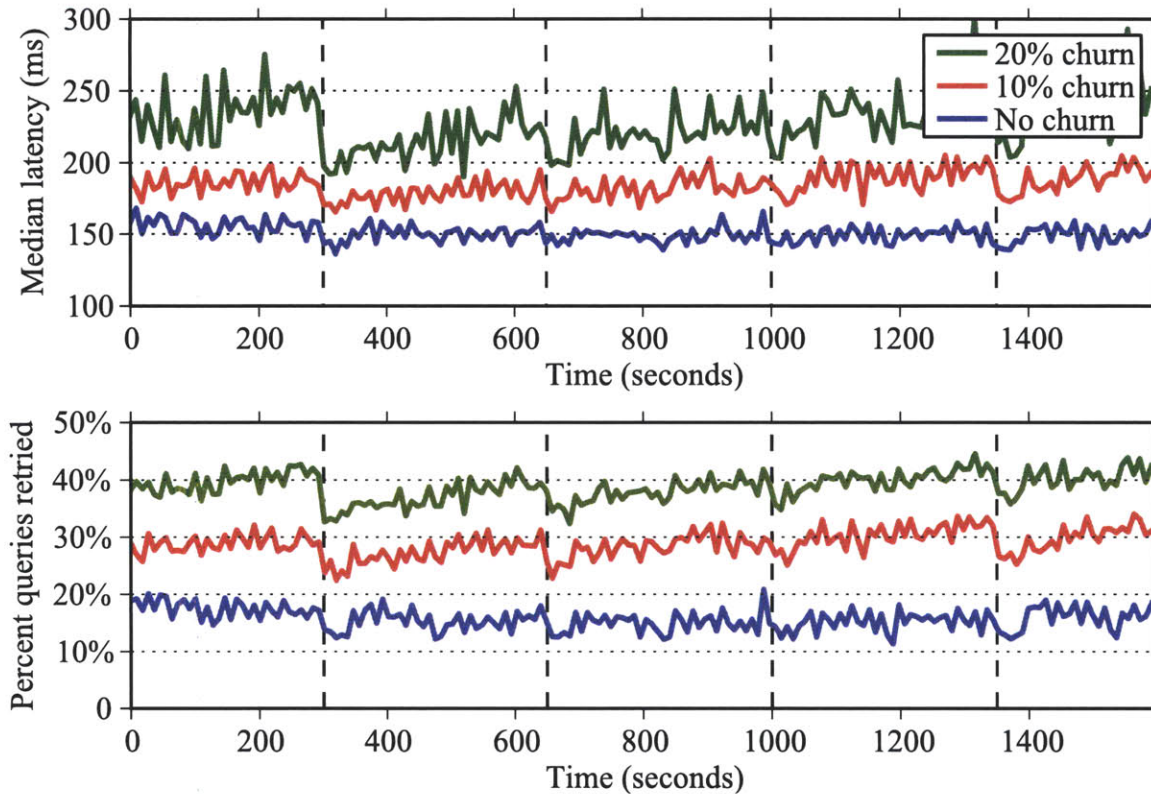
**Figure 8-7:** Number of messages used by LOOKUP, versus system size and table size. The heat map and contour lines show data from synthetic networks, while the markers show that real-world social networks fall roughly onto the same contours. Whānau scales like previous one-hop DHTs.

its scaling in non-Sybil scenarios. Given that previous DHTs would not operate at all in Sybil scenarios, this overhead seems a reasonable price for guaranteed availability.

## 8.5 Performance on PlanetLab with node churn

The example instant-messaging application described in Section 7.1 ran on 400 machines in the PlanetLab public distributed testbed. Each physical machine ran 10 independent Whānau instances, each with 5 social neighbors, yielding a total of 20,000 virtual nodes in the system. 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.

Node churn was simulated by inducing node failure and recovery events according to a Poisson process. These events occurred at an average rate of two per second, but the average node downtime was varied to yield different severities of churn: at any given time, approximately 10% or 20% of the virtual nodes were offline. (In addition 10% and 20% failures, an instance without injected churn was simulated as a control.) Lookup latency was expected to increase over time as some finger nodes became unavailable due to churn, causing some lookups to require multiple retries. Latency was also expected to go back down whenever the IM application re-ran SETUP, building new routing tables to reflect the current state of the network.



**Figure 8-8:** 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 LOOKUP retry frequency slowly increases until SETUP runs again, at which point it reverts to the baseline.

Figure 8-8 plots the lookup latency and retries for these experiments, and shows that Whānau’s performance is largely unaffected by modest node churn. The median latency was approximately a single network round-trip within PlanetLab, and increased gradually with greater churn. As expected, the fraction of LOOKUP 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 was too small to test Whānau’s asymptotic behavior, it did demonstrate two important points: (1) the Whānau Python implementation functions on PlanetLab under realistic wide-area network conditions, and (2) Whānau’s simple approach for maintaining routing tables is sufficient to handle reasonable levels of churn.

# Chapter 9

## Extending Whānau

The previous chapters described the Whānau protocol abstractly, in order to focus on its efficiency and resistance to denial-of-service attacks. This chapter extends the core protocol in various ways to address real-world problems such as exploiting locality, using bandwidth efficiently, preserving privacy, and supporting a wide range of practical applications.

### 9.1 Taking advantage of network locality

As Section 8.5 showed, Whānau’s typical LOOKUP latency is one round-trip to a random PlanetLab node, on the order of 150-200 milliseconds. Some applications might want to reduce this latency by taking advantage of routing table redundancy to send queries to finger nodes with shorter round-trip times (RTTs). [44, 62] However, it’s important to implement network locality carefully: simply querying the closest known fingers would be vulnerable to an adversary that can manipulate its Sybil nodes’ RTTs to appear closer than honest nodes. A “quick fix” would be to perform locality-aware and locality-oblivious lookups in parallel; then, even if the locality-aware queries were co-opted by the adversary, the slow-but-steady parallel lookup would still complete successfully. However, this approach has the disadvantage that it confers no locality benefit when under attack.

If LOOKUP’s latency is much more important than its bandwidth usage, a more subtle technique can be used even when under attack. In base Whānau, CHOOSE-FINGER line 4 (Figure 5-3) chooses a uniformly random finger from a set of candidates; a substantial fraction of the candidates are honest nodes. TRY line 6 then sends a query message to the finger, waiting for a fixed timeout before giving up and retrying.

Suppose, instead, that average RTTs were measured for every finger table entry, and the query timeout was set to the chosen finger node’s average RTT times two. With this adaptive timeout, queries to honest fingers would still usually succeed, but the latency cost of a timed-out query to a malicious finger would be proportional to that finger’s average RTT. As a consequence, Sybil nodes which minimized their RTTs in order to attract more queries would also reduce the amount of damage they can do to the queries’ latencies.

To take advantage of this property, a Whānau implementation can modify CHOOSE-FINGER line 4 to weight each candidate finger’s probability by the inverse of its average RTT, *i.e.*,  $\text{Prob}[f] \propto 1/RTT(f)$ . As an example of how this might work, suppose that the

candidate set contains five Sybils and two honest nodes, all with average RTTs of 100 milliseconds. Since each candidate would have equal probability of being chosen, LOOKUP's expected latency would be approximately one second — five 200-millisecond timeouts and retries. On the other hand, suppose that, due to the adversary's manipulations, the Sybil fingers' RTTs all appear to decrease by a factor of ten, to 10 milliseconds. This would increase their probability of being chosen by a factor of ten, so that LOOKUP would now require fifty timeouts and retries. However, since each timeout would now only be 20 milliseconds, LOOKUP's latency would still be approximately one second.

The above example would appear to indicate that the new strategy wouldn't change LOOKUP's latency, but it actually indicates that the *Sybil nodes* cannot manipulate their own RTTs to increase the latency. In contrast, honest candidates with low RTTs would now be much more likely to be queried. For instance, imagine modifying the previous example so that one of the two honest finger nodes has an RTT of 10 milliseconds; this would increase its probability of being queried by a factor of ten. The overall LOOKUP latency would thus improve from 1 second to only 100 milliseconds, despite the large number of Sybil fingers. In general, this strategy should produce much lower latencies whenever there is substantial diversity amongst honest nodes' RTTs.

The cost of this improved LOOKUP latency is substantially more bandwidth usage, since many more messages are likely to be sent per lookup. This bandwidth cost can be traded off against LOOKUP latency by choosing a candidate weighting function intermediate between the original uniform weights and the new  $1/\text{RTT}$  weights. To take advantage of a greater diversity of candidate fingers, nodes must also increase the size of finger tables and key tables beyond the minimum required for correctness, consuming more SETUP resources. The set of candidate fingers for a given target key grows quadratically with table size.

## 9.2 Efficient random walks using a 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 recursive or iterative RPC to compute a random walk, as suggested by the RANDOM-WALK subroutine in Figure 5-2, is not very efficient: it uses  $2w$  messages per random node returned.

A better approach, used by the Python implementation, is to batch-compute a large number  $r$  of random 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 synchronous time step, each node randomly shuffles its pool and divides it equally amongst its social neighbors (using a reliable transport). For the next time step, the node combines the messages it received from each of its neighbors to create a new pool, and repeats the process. After  $w$  such mixing steps, each node's pool is a randomly shuffled assortment of addresses.

**Theorem.** *If  $r$  is sufficiently large, the systolic mixing process closely approximates sending out  $r$  random walks from each node.*

*Proof sketch.* In the unmodified random walk process, a degree- $d$  node which receives  $r$  requests for random walks of length  $w$  will deliver an expected  $r/d$  requests for random

walks of length  $w - 1$  to each of its neighbors. The actual number of requests sent to each neighbor follows the binomial distribution  $B(r, 1/d)$ , which is sharply peaked around  $r/d$  for large  $r$ , with deviation less than  $\sqrt{r/d}$ . On the other hand, the systolic process delivers *exactly*  $r/d$  random walks of length  $w - 1$  to each neighbor. Thus, each step introduces a small error, but even after  $w$  error-accumulating steps, the difference between the systolic process and the unmodified random walk process remains negligible for large  $r$ .  $\square$

### 9.2.1 Using the systolic process in Whānau

The main advantage of this systolic protocol is that a degree- $d$  node need only send and process  $dw$  messages in total, instead of  $rw$  messages using RPCs. The systolic technique balances random-walk load predictably and uniformly over the nodes, unlike the RPC technique.

The SETUP protocol is defined such that the number of random walks needed is perfectly predictable ahead of time, given the table size parameters. The systolic mixing process is thus easily incorporated as a separate initial phase. While this adds  $w$  steps to the SETUP latency, it actually decreases the individual latency of each subsequent step by a factor of  $w$  (since they no longer have to wait for random walk RPCs to return).

Incorporating the systolic process into the LOOKUP phase would appear to be more difficult than the SETUP phase, but it is actually simpler. There is no requirement that the random walks used by separate runs of LOOKUP be statistically independent from each other. Thus, SETUP can simply pre-compute a small additional pool of random walks which is reused by every run of LOOKUP. This pool should be large enough to accommodate the maximum number of retries needed, which, if the routing tables are sufficiently large, should be only  $\mathcal{O}(\log km)$ .

### 9.2.2 Using symmetry to protect against denial-of-service

The systolic mixing process has the interesting property that it is completely symmetric and reversible: the set of random walks terminating at a node  $u$  has an identical distribution to the set of random walks starting at  $u$ . Consequently, they share the same good sampling properties: relatively few of the random walks terminating at an honest node can originate in the Sybil region.

This feature can be used to distribute “request tokens” to protect nodes against request-flooding denial-of-service attacks. A request token is a random nonce which can only be used a limited number of times. Every RPC request must include a token, and requests containing invalid or expired tokens may be ignored.

Before the systolic mixing process’s first step, each node generates its own set of request tokens and stores them in a local table of unused tokens. It appends one token to each of the  $r$  initial address records in its mixing pool; these tokens are then mixed alongside the addresses, ending up at random nodes. Since the mixing process is symmetric, the adversary can only obtain a limited fraction of each honest node’s tokens. Thus, this technique strictly curtails the scope of a denial-of-service attack based on overloading honest nodes with bogus requests.

## 9.3 Supporting very many or very few keys per node

The Whānau protocol described in this dissertation handles  $1 \lesssim k \lesssim m$  efficiently, where  $k$  is the number of keys per honest node. This base protocol requires various minor tweaks in order to handle the extreme cases outside this range efficiently. As this section will show, the  $k \ll 1$  and  $k \gg m$  cases are actually relatively simple to handle as compared with the  $1 \leq k \leq m$  case, which requires Whānau’s full complexity.

### 9.3.1 Many keys per node

Consider how to solve the case where every node needs to store a very large number of keys  $k > m$  into the DHT. Any structured DHT, secure or not, requires at least  $k = \Omega(m)$  resources per node just to transmit and store all these keys once. This observation enables a simpler DHT design: without budging the asymptotic resource usage at all, each DHT node could use  $\mathcal{O}(m)$  bandwidth to collect a (nearly) complete list of all other honest nodes. Having replicated such a list, consistent hashing [71] could be used to distribute records directly to key tables, avoiding the need for *intermediate* tables and SLICE-SAMPLE.

Returning to the base Whānau protocol, the analysis in Section 6.2 breaks down when  $k > m$ : more than  $m$  calls to SLICE-SAMPLE will tend to have many repeats, and thus cannot be treated as independent trials. To recover this property, a Whānau implementation can simply split each old virtual node into approximately  $\sqrt{\frac{k}{m}} > 1$  new virtual nodes.

This yields a total of  $2m \cdot \sqrt{\frac{k}{m}} = 2\sqrt{km}$  virtual nodes storing approximately  $\sqrt{km}$  keys each. By Chapter 6’s analysis, the SETUP resource budget per (new) virtual node should be  $\mathcal{O}(\sqrt{km} \log km)$ , or  $\mathcal{O}(\log km)$  per stored key. This is an optimal result, since any secure DHT would need to replicate records by a logarithmic factor to ensure at least one honest replica for each record.

### 9.3.2 Fewer keys than nodes

Now consider the other case,  $k < 1$ , where only a subset of nodes are storing any key-value records into the system. In the extreme limit  $k = 1/2m$ , only a single honest node attempts to store a single key-value record into the system. Conceptually, handling this case is trivial: the single key-value record can be flooded to all honest nodes over the social network at a cost of only  $\mathcal{O}(1)$  bandwidth per node.

Whānau can be straightforwardly adapted to handle the case  $k < 1$  by adopting the systolic mixing process described above (Section 9.2) to transport key-value records from put-queues to intermediate tables. Nodes with empty put-queues simply initialize their pools with null entries, which can be omitted on the wire and thus cost nothing to store and transmit. All other details and parameters of the Whānau protocol remain unchanged.

In the extreme case  $k = 1/2m$ , with one record stored in the DHT, this approach essentially reduces to social flooding; the modified Whānau protocol smoothly adapts to larger  $k$ . However, this approach works only for small numbers of attack edges  $g = \mathcal{O}(km/w)$ ; otherwise, the adversary can flood the network with its own key-value records, which the



honest nodes must store. Sybil-proofness will generally require the table sizes at each node to be at least  $\Omega(\sqrt{km} + gw \log km)$ .

## 9.4 Routing over the social network

Section 3.1.1 described how to use Whānau for rendezvous, by storing a client’s current IP address into the DHT under its user’s persistent identifier (*e.g.*, a username or public key). To send a message to the user, another client can perform a lookup on this persistent identifier to retrieve the current IP address, and then send a packet to this address.

### 9.4.1 Recursive routing

In the above “iterative routing” approach, Whānau’s CHOOSE-FINGER subroutine finds an honest finger storing the appropriate identifier-to-address binding, and LOOKUP queries the finger for the address. In an alternative “recursive routing” approach, the source client could simply send the persistent identifier and message to the chosen finger. The finger node would forward the message on to the destination client using the stored address.

The recursive approach has the advantage of decreasing the typical message latency from three hops (source to finger, finger to source, source to destination) to two hops (source to finger, finger to destination). Moreover, if the finger maintains a persistent open connection with the destination instead of simply storing its address, this approach can avoid any connection setup latency (*e.g.*, to set up transport layer security [21]). Finally, by sending messages over persistent connections, this technique can be used to route messages to nodes behind firewalls and NATs (which would block packets sent using the iterative technique).

On the other hand, the recursive routing technique increases the load on finger nodes, since they must forward message payloads instead of simply answering short routing queries. If the destination node does not send an acknowledgement message to the source node, then the source must send via multiple fingers in parallel, since some of the fingers may be Sybils. Whānau guarantees that CHOOSE-FINGER returns a substantial fraction of good fingers; thus, the source can either send the whole payload via each finger, or, more efficiently, it can split the payload into erasure-coded fragments to be reassembled at the destination node.

### 9.4.2 Tunneling over social links

In Whānau, every key table entry and every finger table entry was created by taking a  $w$ -hop random walk on the social network. If the random walk recorded its entire path, then this could also be stored into the routing tables and later used to forward messages by retracing the same path over the social network. Using this observation, Whānau can be extended to route messages using only social network connections: simply forward the message from source to finger using random-walk hops stored in the source’s finger table, and then forward the message from finger to destination using random-walk hops stored in the finger node’s key table.

In an overlay network setting, this approach would generally increase routing latency and load as compared with direct messages; it would have the (minor) benefit of reducing the number of direct network connections each node must establish. However, in a *non*-overlay setting, where there is no underlying IP network, this technique enables Whānau to be a standalone routing protocol. For example, Whānau can be used for Internet inter-AS routing, or for routing on mobile or vehicular ad-hoc networks, if the connections between these nodes have the same good properties as social networks (Section 3.2). While Whānau’s randomly generated routes are not as efficient as shortest-path routes, they could be used as reliable fallback routes, or used to carry critical control-plane messages.

## 9.5 Preserving friend list privacy

Many ways of implementing random walks reveal which nodes are connected to which other nodes. For example, if an implementation simply uses recursive RPCs, an attacker can learn all of a node’s  $d$  social neighbors using only  $\mathcal{O}(d)$  calls to RANDOM-WALK(1).

In many cases, users’ “friend lists” are sensitive private information. Whānau’s design fundamentally requires that social contacts maintain open communication channels with each other; thus, an eavesdropping adversary can always use traffic analysis to guess a node’s social neighbors. However, it is worthwhile to examine techniques to protect friend lists against adversaries that cannot observe all network traffic, but can make Byzantine RPC requests or send Byzantine messages.

Of course, a first step is to respond only to random walk requests from a node’s social neighbors. (In normal Whānau operation, no non-neighbor nodes would try to call RANDOM-WALK.) While this limits the attack, it may not be sufficient, since a user may not want her own friends (or friends of friends) to learn her friend list.

A more complete approach would use and extend the tunneling technique described above (Section 9.4.2), in which each finger and key table entry is associated with a source route used to send messages over the social network. The most simple form of source route, a list of globally unique node IDs, would compromise friend list privacy. However, since each entry of the source route is used only by a single node to forward messages to the next hop (one of the node’s neighbors), there’s no need to store globally unique IDs: a locally-unique neighbor identifier is just as good. Thus, privacy can be preserved by replacing each source route entry with a locally-valid identifier meaningful only to the previous node on the route. For instance:

- The next hop’s global ID, encrypted under a key known only to the current hop.
- A randomly-generated local neighbor ID. The current hop would map this to a next hop using a locally-stored table.
- A small-integer index into the current hop’s list of neighbors.

As long as all Whānau messages are forwarded via the social network using such source routes, any of these three techniques would protect the identities of a node’s social neighbors. The latter technique would have the additional advantage of using less space to store

the source routes than a list of unique node IDs. However, it may require extra bookkeeping to ensure stable local indices when adding or removing social network links.

## 9.6 Handling large, durable data blocks

Whānau replicates each stored key-value record over  $\mathcal{O}(\sqrt{km} \log km)$  different nodes in order to provide the degree of redundancy needed for one-hop routing. If each node stores a few small records, as in a rendezvous application (Section 3.1.1), this replication overhead is similar to any one-hop DHT's resource requirements. However, some applications, such as filesystems [45, 89, 111] or Web caches [53, 55, 67], need to store *large, variable-sized* chunks of data into the DHT. Such applications probably do not want such a high replication factor for bulk data storage, and would rather choose their own replication factors independent of the routing protocol.

An additional layer of indirection can provide Whānau this functionality, which was omitted from the core protocol for simplicity. To store a large data block, a client should choose a small number of replica nodes, copy the block to each replica, and insert a list of the replicas' IP addresses into the DHT. Whānau will replicate this list many times, but a list of a few IP addresses should be much smaller than the original data block.

Applications can choose replica nodes according to different policies. For example, a node might choose to store block replicas only on its social neighbor nodes (as in Friendstore [112]), or it might choose to use an external service provider to store some or all of the replicas. If the application needs to balance the load over all honest nodes, then nodes can choose replicas by taking random walks on the social network. Because this latter policy requires choosing  $r/\xi$  replicas in order to yield approximately  $r$  honest replicas, the security overhead grows with the number of attack edges. When  $g \ll n/w$ , this overhead percentage is proportional to  $g$ , analogous to other DHTs where the overhead percentage is proportional to the number of malicious nodes.

Whānau periodically discards and rebuilds its tables; therefore, in order for a record to be *durable*, it must be re-inserted into the DHT with every SETUP round. In the above block-storage protocol, if a block's replicas are Whānau nodes, they can be given responsibility for re-inserting the IP address list record into the DHT. As long as at least one honest replica is up during every SETUP round, the record will remain available. This means that the originating node need not stay online to ensure its data's persistence.

## 9.7 Handling dynamic record updates and insertions

Whānau's most obvious limitation is its lack of a dynamic INSERT operation; because SETUP builds a static set of routing tables, new record insertions aren't visible until the following SETUP round completes. The protocol can be extended to support updating the value associated with a key, but not inserting new keys. However, it is possible to trade off LOOKUP bandwidth against the key insertion delay.

### 9.7.1 Protocols for updating values

To implement an UPDATE operation, the DHT must keep track of which nodes store replicas of each record. The simplest implementation of mutable records adopts the layer of indirection described above (Section 9.6), which stores a list of the block’s replicas’ IP addresses into the core DHT. To modify a block’s contents, a client can simply send an update message to each of the replica nodes; nothing in the core DHT needs to change. To ensure that only authorized clients can mutate the value, the update message should include a signature or other authenticator (*e.g.*, a hashed secret [95]). Readers and writers can implement a quorum protocol to handle unavailable or malicious replica nodes.

The indirection-layer approach imposes an additional hop on lookups; applications with tighter latency requirements may prefer to spend the resources to update all  $\mathcal{O}(\sqrt{km} \log km)$  Whānau replicas of the record. To accomplish this, the originating node must keep back-pointers to every copy of the record; it does this by storing a local table entry for every node which collected the record into its intermediate table using SAMPLE-RECORD. Similarly, those nodes will store a local table entry for every node which collected the record into its key table using SLICE-SAMPLE. To update the record’s value, the originating node simply broadcasts the new value to all the replicas using these back-pointers.

As with all Whānau routing tables, attackers must be prevented from overloading these back-pointer tables. A simple solution is to employ the systolic random walk technique from Section 9.2, which limits the number of random walks ending at any honest node.

### 9.7.2 Challenges to inserting keys

While supporting an UPDATE operation is straightforward, a secure, efficient, and dynamic INSERT operation remains a challenging open problem.

If honest nodes have unlimited resources, a similar back-pointer approach can be used to insert new keys. Each node would store a log of every SAMPLE-RECORD or SLICE-SAMPLE request it served in the previous SETUP round. Whenever newly inserted keys would have resulted in different replies to these requests, the node would re-send an updated reply.

This technique yields perfectly up-to-date key tables. If the new keys are distributed similarly to the initial set of keys, the key tables remain well-balanced as well. Unfortunately, if the adversary causes newly inserted keys to be clustered (either by inserting clustered keys himself, or by causing honest nodes to insert clustered keys), then some key tables can grow arbitrarily large. If the targeted honest nodes cannot handle the additional load, they will be forced to drop some records, including possibly some honest records.

An adversary with few attack edges ( $gw \ll r_k = \mathcal{O}(\sqrt{km})$ ) cannot insert enough keys to unbalance the key tables by himself. Additionally, in some applications, honest nodes can ensure an even key distribution by choosing random keys or by salting their input keys. If both conditions hold, then Whānau can provide an INSERT operation supporting up to  $\mathcal{O}(km)$  dynamic key insertions between SETUP rounds. However, this version of the protocol is much less Sybil-proof than the static version ( $gw = \mathcal{O}(\sqrt{km})$  versus  $gw = \mathcal{O}(n)$ ).

In summary, it is currently possible to implement an “optimistic” INSERT which attempts to dynamically insert a new record, and succeeds if there are few ( $\mathcal{O}(\sqrt{km})$ ) attack edges and if new honest keys follow a similar distribution to existing keys. In the worst case, newly stored keys will be unavailable until SETUP rebuilds the routing tables. A stronger dynamic INSERT protocol is a goal for future versions of Whānau.

### 9.7.3 Trading bandwidth against key insert latency

Whānau’s SETUP bandwidth usage can be traded off against responsiveness to churn: for example, running SETUP twice as frequently halves the latency from key insertion to key visibility. Using the observation that the DHT capacity scales with the square of the table size, it is also possible to trade off LOOKUP bandwidth usage against responsiveness to churn.

Consider running SETUP every  $T$  seconds using table size  $r = \mathcal{O}(\sqrt{km} \log km)$ , yielding a total system capacity of  $km$  keys. Now, suppose instead that nodes run SETUP every  $T/2$  seconds using  $r/2$  resources, but they save the last four instances of the routing tables. In the latter case, *each* instance of the DHT has capacity  $km/4$ , but since there are four independent instances, the total capacity remained the same. The total resource usage per unit time also remains the same in both cases, but the responsiveness to churn doubles in the latter case, since SETUP runs twice as often. New keys can be inserted into any instance of the DHT with available space.

This scaling trick might appear to be getting better insert latency for free, but there is a price: since the multiple DHT instances are independent, lookups must be performed in all instances in parallel. Thus, the bandwidth used by LOOKUP increases proportionally to the number of saved instances. In order to halve the key insert latency, LOOKUP bandwidth must quadruple.



# Chapter 10

## Conclusion

Over the first decade of this century, research in structured distributed hash tables has made great progress toward the goal of an open, collaborative Internet, in which peers cooperate to provide the infrastructure services they use. However, this vision faces a well-known obstacle: the Sybil attack, in which a malicious entity creates many independent identities and joins the system repeatedly in order to degrade its performance. Previous designs prevented the Sybil attack by requiring some trusted central authority to control admission to the DHT. It was natural to wonder: was a central authority an inherent requirement? Or, was it possible to design a completely decentralized DHT, where all peers participate on an equal basis?

This dissertation has affirmatively answered this long-open and important question. It described Whānau, a structured DHT protocol which is secure against powerful denial-of-service attacks from an adversary able to create unlimited Sybil identities. The Whānau protocol combined previous techniques — 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  $\mathcal{O}(\sqrt{km} \log km)$  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  $\mathcal{O}(\sqrt{km} \log km)$ , 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.





# Bibliography

- [1] Facebook LiveMessage API. <http://wiki.developers.facebook.com/index.php/LiveMessage>. URL.
- [2] The DBLP computer science bibliography. <http://dblp.uni-trier.de/>. URL.
- [3] Boost C++ libraries. <http://www.boost.org/>. URL.
- [4] Facebook social network. <http://www.facebook.com/>. URL.
- [5] Flickr photo sharing service. <http://www.flickr.com/>. URL.
- [6] LinkedIn social network. <http://www.linkedin.com/>. URL.
- [7] LiveJournal blog service. <http://www.livejournal.com/>. URL.
- [8] MySpace social network. <http://www.myspace.com/>. URL.
- [9] Orkut online community. <http://orkut.com/>. URL.
- [10] RPyC: Remote Python Call. <http://rpyc.wikidot.com/>. URL.
- [11] Tangata Whenua: Māori news and indigenous views. <http://tangatawhenua.com>. URL.
- [12] TruthTweet: who's real, and who's a fake? <http://truthtweet.com/faq/>. URL.
- [13] Twitter. <http://twitter.com/>. URL.
- [14] Vuze BitTorrent client. <http://www.vuze.com/>. URL.
- [15] YouTube video sharing service. <http://www.youtube.com/>. URL.
- [16] Vivienne Adair and Robyn Dixon, ed. The family in Aotearoa/New Zealand. Longman, Auckland, New Zealand, 1998.
- [17] Introduction to Windows peer-to-peer networking. Microsoft TechNet, bb457079, September 2006. URL.
- [18] Peer Name Resolution Protocol. Microsoft TechNet, bb726971, September 2006. URL.

- [19] Mark J. Handley and Eric Rescorla, ed. Internet denial-of-service considerations. RFC 4732, November 2006.
- [20] YouTube hijacking: A RIPE NCC RIS case study. <http://www.ripe.net/news/study-youtube-hijacking.html>, RIPE Network Coordination Centre, February 2008. URL.
- [21] Tim Dierks and Eric Rescorla, ed. The Transport Layer Security (TLS) protocol (version 1.2). RFC 5246, August 2008.
- [22] Secure Hash Standard (SHS). FIPS 180-3, National Institute of Standards and Technology, Gaithersburg, Maryland, October 2008.
- [23] Introducing the Distributed Routing Table in Windows 7. Microsoft Peer-to-Peer Networking blog, November 2008. URL.
- [24] Peer Name Resolution Protocol (PNRP) version 4.0 specification. Microsoft, August 2010. URL.
- [25] Keri Lawson-Te Aho. Definitions of Whānau: A review of selected literature. New Zealand Families Commission, Wellington, New Zealand, April 2010. URL.
- [26] Baruch Awerbuch and Robert Kleinberg. Competitive collaborative learning. *Journal of Computer and System Sciences*, 74(8):1271–1288, Elsevier, December 2008.
- [27] Baruch Awerbuch and Christian Scheideler. Towards scalable and robust overlay networks. *International Workshop on Peer-to-Peer Systems*, Bellevue, Washington, February 2007.
- [28] Randy Baden, Adam Bender, Dave Levin, Rob Sherwood, Neil Spring, and Bobby Bhattacharjee. A secure DHT via the pigeonhole principle. University of Maryland, Technical Report TR-CS-4884, September 2007.
- [29] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509-512, 1999. URL.
- [30] Ingmar Baumgart and Sebastian Mies. S/Kademlia: a practicable approach towards secure key-based routing. *International Conference on Parallel and Distributed Systems*, pages 1–8, Hsinchu, Taiwan, December 2007.
- [31] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating systems support for planetary-scale network services. *Symposium on Networked System Design and Implementation*, San Francisco, California, March 2004.
- [32] Bobby Bhattacharjee, Rodrigo Rodrigues, and Petr Kouznetsov. Secure lookup without (constrained) flooding. *Workshop on Recent Advances on Intrusion-Tolerant Systems*, Lisbon, Portugal, March 2007.

- [33] Nikita Borisov. Computational puzzles as Sybil defenses. *International Conference on Peer-to-Peer Computing*, pages 171-176, Cambridge, United Kingdom, September 2006.
- [34] Martin A. Brown. Pakistan hijacks YouTube. Renesys blog, February 2008. URL.
- [35] Kevin R.B. Butler, Sunam Ryu, Patrick Traynor, and Patrick D. McDaniel. Leveraging identity-based cryptography for node ID assignment in structured p2p systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(12):1803–1815, IEEE Press, December 2009.
- [36] Miguel Castro, Peter Druschel, Ayalvadi J. Ganesh, Antony I. T. Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *Symposium on Operating System Design and Implementation*, Boston, Massachusetts, December 2002.
- [37] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.
- [38] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-bandwidth multicast in cooperative environments. *Symposium on Operating Systems Principles*, Bolton Landing, New York, October 2003.
- [39] Alice Cheng and Eric Friedman. Sybilproof reputation mechanisms. *Workshop on the Economics of Peer-to-Peer Systems*, pages 128–132, Philadelphia, Pennsylvania, August 2005.
- [40] Raymond S. Cheng. WhanauSIP: a secure peer-to-peer communications platform. Master’s Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, August 2010.
- [41] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. *Symposium on Networked System Design and Implementation*, San Jose, California, May 2006.
- [42] Fan Chung. *Spectral graph theory*. American Mathematical Society, 1997.
- [43] Chris Cunningham, Brenda Stevenson, and Natasha Tassel. Analysis of the characteristics of Whānau in Aotearoa. New Zealand Ministry of Education, May 2005. URL.
- [44] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *SIGCOMM Conference*, Portland, Oregon, August 2004.

- [45] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. *Symposium on Operating Systems Principles*, Banff, Canada, October 2001.
- [46] George Danezis and Prateek Mittal. SybilInfer: Detecting Sybil nodes using social networks. *Network and Distributed System Security Symposium*, San Diego, California, February 2009.
- [47] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross J. Anderson. Sybil-resistant DHT routing. *esorics*, pages 305-318, 2005.
- [48] John R. Douceur. The Sybil attack. *International Workshop on Peer-to-Peer Systems*, pages 251-260, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, Massachusetts, March 2002.
- [49] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, July 2010.
- [50] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust incentive techniques for peer-to-peer networks. *ACM Conference on Electronic Commerce*, pages 102–111, New York, New York, 2004.
- [51] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, 24(5):1010–1019, May 2006.
- [52] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. Persistent personal names for globally connected mobile devices. *Symposium on Operating System Design and Implementation*, Seattle, Washington, November 2006.
- [53] Michael J. Freedman. Experiences with CoralCDN: A five-year operational view. *Symposium on Networked System Design and Implementation*, San Jose, California, May 2010.
- [54] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. *ACM Conference on Computer and Communications Security*, pages 193-206, Vijayalakshmi Atluri, ed. ACM, Washington, D.C. November 2002.
- [55] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. *Symposium on Networked System Design and Implementation*, San Francisco, California, March 2004.
- [56] Michael J. Freedman, Karthik Lakshminarayanan, and David Mazières. OASIS: Anycast for any service. *Symposium on Networked System Design and Implementation*, San Jose, California, May 2006.

- [57] Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *ACM Transactions on Computer Systems*, 20(1):1–24, February 2002.
- [58] Kevin Fu, Seny Kamara, and Tadayoshi Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. *Network and Distributed System Security Symposium*, San Diego, California, February 2006. URL.
- [59] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y. Zhao. Detecting and characterizing social spam campaigns. *Internet Measurement Conference*, Melbourne, Victoria, Australia, November 2010.
- [60] Roxana Geambasu, Jarret Falkner, Paul Gardner, Tadayoshi Kohno, Arvind Krishnamurthy, and Henry M. Levy. Experiences building security applications on DHTs. University of Washington, Technical Report UW-CSE-09-09-01, 2009.
- [61] Roxana Geambasu, Tadayoshi Kohno, Amit A. Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. *USENIX Security Symposium*, Montréal, Canada, August 2009.
- [62] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. Efficient routing for peer-to-peer overlays. *Symposium on Networked System Design and Implementation*, pages 113-126, San Francisco, California, March 2004.
- [63] Indranil Gupta, Kenneth P. Birman, Prakash Linga, Alan J. Demers, and Robbert van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. *International Workshop on Peer-to-Peer Systems*, pages 160-169, M. Frans Kaashoek and Ion Stoica, ed. Springer Lecture Notes in Computer Science 2735, Berkeley, California, February 2003.
- [64] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A scalable overlay network with practical locality properties. *USENIX Symposium on Internet Technologies and Systems*, Seattle, Washington, March 2003.
- [65] Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Voshall, and Werner Vogels. Dynamo: Amazons highly available key-value store. *Symposium on Operating Systems Principles*, pages 205–220, Bretton Woods, New Hampshire, October 2007.
- [66] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. *International Conference on Very Large Databases*, Berlin, Germany, September 2003.
- [67] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: A decentralized, peer-to-peer Web cache. *ACM Symposium on Principles of Distributed Computing*, Monterey, California, July 2002.

- [68] Laurent Joncheray. Simple active attack against TCP. *USENIX Security Symposium*, Salt Lake City, Utah, June 1995.
- [69] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: scalable secure file sharing on untrusted storage. *Conference on File and Storage Technologies*, pages 29–42, USENIX, San Francisco, California, March 2003.
- [70] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. *International World Wide Web Conference*, pages 640–651, Budapest, Hungary, May 2003.
- [71] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *ACM Symposium on Theory of Computing*, pages 654-663, El Paso, Texas, May 1997.
- [72] Avinash Lakshman and Prashant Malik. Cassandra — A decentralized structured storage system. *ACM SIGOPS Workshop on Large Scale Distributed Systems and Middleware, ACM SIGOPS Workshop on Large Scale Distributed Systems and Middleware*, 2009.
- [73] Chris Lesniewski-Laas. A Sybil-proof one-hop DHT. *Workshop on Social Network Systems*, Glasgow, April 2008.
- [74] Chris Lesniewski-Laas and M. Frans Kaashoek. Whānau: A Sybil-proof distributed hash table. *Symposium on Networked System Design and Implementation*, San Jose, California, May 2010.
- [75] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the Sybil attack. University of Massachusetts Amherst, Amherst, Massachusetts, 2006.
- [76] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek. Bandwidth-efficient management of DHT routing tables. *Symposium on Networked System Design and Implementation*, Boston, Massachusetts, May 2005.
- [77] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. Secure Untrusted Data Repository (SUNDR). *Symposium on Operating System Design and Implementation*, San Francisco, California, December 2004.
- [78] Andrew Loewenstern. BitTorrent DHT protocol. BitTorrent BEP 5, February 2008. URL.
- [79] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. DHT routing using social links. *International Workshop on Peer-to-Peer Systems*, pages 100-111, Geoffrey M. Voelker and Scott Shenker, ed. Springer Lecture Notes in Computer Science 3279, La Jolla, California, February 2004.

- [80] Petar Maymounkov and David Mazières. Kademia: a peer-to-peer information system based on the XOR metric. *International Workshop on Peer-to-Peer Systems*, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, Massachusetts, March 2002.
- [81] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. *Symposium on Operating Systems Principles*, pages 124-139, Kiawah Island, South Carolina, December 1999.
- [82] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr social network. *Workshop on Online Social Networks*, Seattle, Washington, August 2008. URL.
- [83] Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. *Internet Measurement Conference*, pages 29-42, San Diego, California, October 2007. URL.
- [84] Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: Leveraging trust to thwart unwanted communication. *Symposium on Networked System Design and Implementation*, pages 15–30, San Francisco, California, April 2008.
- [85] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, January 2005.
- [86] Abdelaziz Mohaisen, Nicholas Hopper, and Yongdae Kim. Designs to account for trust in social network-based Sybil defenses. *ACM Conference on Computer and Communications Security*, ACM, Chicago, Illinois, October 2010. URL.
- [87] Abdelaziz Mohaisen, Aaram Yun, and Yongdae Kim. Measuring the mixing time of social graphs. *Internet Measurement Conference*, Melbourne, Victoria, Australia, November 2010. URL.
- [88] John C. Moorfield. *Te Aka Māori-English, English-Māori dictionary and index*. Longman/Pearson Education, New Zealand, 2005. URL.
- [89] Athicha Muthitachoen, Robert Morris, Thomer Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. *Symposium on Operating System Design and Implementation*, Boston, Massachusetts, December 2002.
- [90] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web. Stanford InfoLab, Technical Report 66, Stanford InfoLab, Previous number = SIDL-WP-1999-0120, November 1999. URL.
- [91] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122-144, 2004.

- [92] Bogdan C. Popescu, Bruno Crispo, and Andrew S. Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. *Security Protocols Workshop*, pages 213-220, Cambridge, United Kingdom, April 2004.
- [93] Anantha Ramaiah, Randall R. Stewart, and Mitesh Dalal. Improving TCP's robustness to blind in-window attacks. RFC 5961, August 2010.
- [94] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. *SIGCOMM Conference*, pages 161-172, San Diego, California, August 2001.
- [95] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiawicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. OpenDHT: A public DHT service and its uses. *SIGCOMM Conference*, Philadelphia, Pennsylvania, September 2005.
- [96] Matei Ripeanu and Ian T. Foster. Mapping the Gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *International Workshop on Peer-to-Peer Systems*, pages 85-93, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, Massachusetts, March 2002.
- [97] Rodrigo Rodrigues and Barbara Liskov. Rosebud: A scalable Byzantine-fault-tolerant storage architecture. MIT CSAIL, Technical Report TR/932, December 2003.
- [98] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.
- [99] Hosam Rowaihy, William Enck, Patrick McDaniel, and Tom La Porta. Limiting Sybil attacks in structured P2P networks. *IEEE International Conference on Computer Communications*, pages 2596-2600, Anchorage, Alaska, May 2007.
- [100] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Symposium on Operating Systems Principles*, Banff, Canada, October 2001.
- [101] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *International Middleware Conference*, volume 2218, pages 329-350, Rachid Guerraoui, ed. Springer, Heidelberg, Germany, November 2001.
- [102] Christian Scheideler. How to spread adversarial nodes?: Rotate! *ACM Symposium on Theory of Computing*, pages 704-713, Baltimore, Maryland, May 2005.
- [103] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. *IEEE International Conference on Computer Communications*, Barcelona, Spain, April 2006.



- [104] Kundan Singh and Henning Schulzrinne. Peer-to-peer Internet telephony using SIP. Columbia University, Technical Report CUUCS-044-04, October 2004.
- [105] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. *International Workshop on Peer-to-Peer Systems*, pages 261-269, Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, ed. Springer Lecture Notes in Computer Science 2429, Cambridge, Massachusetts, March 2002.
- [106] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Frans Kaashoek, Robert Morris, and John Kubiatoiwicz. Proactive replication for data durability. *International Workshop on Peer-to-Peer Systems*, Santa Barbara, California, February 2006.
- [107] Emil Sit, Robert Morris, and M. Frans Kaashoek. UsenetDHT: A low-overhead design for Usenet. *Symposium on Networked System Design and Implementation*, San Francisco, California, April 2008.
- [108] Alexander Stepanov and Meng Lee. The Standard Template Library. HP Laboratories, Technical Report 95-11(R.1), November 1995.
- [109] Ion Stoica, Dan Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana, and Shelley Zhuang. Internet indirection infrastructure. *SIGCOMM Computer Communications Review*, 32(4):86, ACM, 2002.
- [110] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17-32, 2003.
- [111] Jeremy Stribling, Emil Sit, M. Frans Kaashoek, Jinyang Li, and Robert Morris. Don't give up on distributed file systems. *International Workshop on Peer-to-Peer Systems*, Bellevue, Washington, February 2007.
- [112] Dinh Nguyen Tran, Frank Chiang, and Jinyang Li. Friendstore: Cooperative online backup using trusted nodes. *Workshop on Social Network Systems*, pages 37-42, Glasgow, April 2008.
- [113] Dinh Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, and Sherman S.M. Chow. Improving social-network-based Sybil-resilient node admission control. Brief announcement. *ACM Symposium on Principles of Distributed Computing*, Zürich, Switzerland, July 2010.
- [114] Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient online content rating. *Symposium on Networked System Design and Implementation*, Boston, Massachusetts, April 2009.
- [115] Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. An analysis of social network-based Sybil defenses. *SIGCOMM Conference*, pages 363-374, New Delhi, India, September 2010.

- [116] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, Web publishing system. *USENIX Security Symposium*, pages 59–72, Denver, Colorado, August 2000.
- [117] Tai Walker and Ngāti Porou. Whānau is Whānau. *Blue Skies Report*, volume 8, 06, New Zealand Families Commission, Wellington, New Zealand, July 2006. URL.
- [118] Bing Wei, Ben Y. Zhao, and Alessandra Sala. An user authentication service for Twitter. Poster, University of California, Santa Barbara, 2010.
- [119] Scott Wolchok, Owen S. Hofmann, Nadia Heninger, Edward W. Felten, J. Alex Halderman, Christopher J. Rossbach, Brent Waters, and Emmett Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. *Network and Distributed System Security Symposium*, San Diego, California, February 2010.
- [120] Alexander Yip, Benjie Chen, and Robert Morris. Pastwatch: A distributed version control system. *Symposium on Networked System Design and Implementation*, San Jose, California, May 2006.
- [121] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. A near-optimal social network defense against Sybil attacks. *IEEE Symposium on Security and Privacy*, Oakland, California, May 2008.
- [122] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybil-Guard: Defending against Sybil attacks via social networks. *SIGCOMM Conference*, pages 267-278, Piza, Italy, September 2006.
- [123] Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B. Gibbons, and Feng Xiao. DSybil: Optimal Sybil-resistance for recommendation systems. *IEEE Symposium on Security and Privacy*, Oakland, California, May 2009.
- [124] Ben Y. Zhao, John Kubiawicz, and Anthony D. Joseph. Tapestry: A fault-tolerant wide-area application infrastructure. *SIGCOMM Computer Communications Review*, 32(1):81, 2002.