Sybil-Resistant DHT Routing

George Danezis[1], Chris Lesniewski-Laas[2],
M. Frans Kaashoek[2], and Ross Anderson[1]

[1] University of Cambridge, Computer Laboratory,
15 J J Thomson Avenue, Cambridge CB3 0FD,
United Kingdom
{George.Danezis, Ross.Andreson}@cl.cam.ac.uk
[2] MIT Computer Science and Artificial Intelligence Laboratory,
The Stata Center, Building 32,
32 Vassar Street, Cambridge, MA 02139, USA
ctl@mit.edu, kaashoek@csail.mit.edu

Abstract. Distributed Hash Tables (DHTs) are very efficient distributed systems for routing, but at the same time vulnerable to disruptive nodes. Designers of such systems want them used in open networks, where an adversary can perform a sybil attack by introducing a large number of corrupt nodes in the network, considerably degrading its performance. We introduce a routing strategy that alleviates some of the effects of such an attack by making sure that lookups are performed using a diverse set of nodes. This ensures that at least some of the nodes queried are good, and hence the search makes forward progress. This strategy makes use of latent social information present in the introduction graph of the network.

1 Introduction

Distributed Hash Tables (or DHTs) [14,15,12,10] are distributed systems that allow efficient lookup of identifiers and routing to the corresponding nodes. They achieve this by imposing on the routing tables of nodes a rigid structure that guarantees quick convergence to a target.

This rigid structure makes DHTs easy to disrupt by a set of malicious nodes that return useless information instead of helping in the routing. An adversary can create a very large number of bogus nodes and flood the DHT network, in order to disrupt it or degrade its performance. This is called a sybil attack [3].

We present a method that lowers the probability an honest user queries a malicious node. The method takes into account the DHT's introduction graph, which describes which node introduced which to the network. We assume that the adversary is connected to the graph at very few points, but that it can create large numbers of virtual "sybils" behind its attachment points. Following a strategy inspired by Advogato [5], our method turns these few corrupt attachment points into trust bottlenecks. We ensure that queries use a diverse set of nodes, thereby minimising our reliance on a localized set of nodes that might be controlled by the adversary. We also show that trying to minimise the number

S. De Capitani di Vimercati et al. (Eds.): ESORICS 2005, LNCS 3679, pp. 305–318, 2005.

of corrupt nodes in honest nodes' routing tables makes a significant difference to the performance of the DHTs.

The security of DHTs, including routing security which is the main concern of our work, has been the subject of discussion in [13,2]. Trust metrics based on social networks were introduced in Advogato [5]. Advogato uses maximum flow in a network to make trust judgments, but there are other proposals, such as Appleseed [16], which use spreading activation models. Our work uses such social network trust metrics to tackle the sybil attack in structured peer-to-peer systems. Sprout [7] is also making use of social network information, to route messages over trusted nodes. We follow the opposite approach and attempt to eliminate trust bottlenecks, thereby trying not to trust any nodes more than others.

2 The Sybil Attack Against DHTs

The basic premise of the *sybil attack* [3] is that an adversary in a peer-to-peer system can easily introduce a very large set of corrupt participants. All of these participants, or *sybils*, are controlled by the adversary; they can be used to compromise security properties of the system or degrade its performance. The latter can be framed in the context of computer security by considering degradation of performance as service denial [9].

A Distributed Hash Table (DHT) is a specialized distributed system that aims to look up identifiers efficiently in order to route messages to and from the corresponding nodes. Our designs will be based on Chord [14], but the principles we will examine (both in terms of understanding the sybil attack and defending against it) are applicable to other systems [15,12,10]. Nodes in Chord arrange themselves into a ring sorted according to their IDs, where each knows its successor. In correct operation, this guarantees that all nodes are reachable. Chord achieves its efficiency by additionally requiring each node to know a small number of other nodes in the network, its *finger table*. Finger nodes are selected to be carefully spaced [4] around the ring address space to ensure that lookups will quickly converge towards a target node.

Lookups can happen in two ways: either recursively or iteratively. In a *recursive* lookup the initiator looks up a particular ID by asking the finger with the closest ID to the target node. The finger node will in turn ask one of its fingers, and this procedure is repeated until the target node is located and the answer propagated back. Iterative routing relies on the initiator of a lookup to query the finger with the closest ID to the target, which in turn returns one of its fingers. The initiating node can then perform further lookups itself, using the additional information until the target node is located. Our sybil resistant lookup strategies will implement a variant of the iterative method, giving the requesting node the most flexibility. Each iteration returns a set of nodes instead of just one.

An adversary can participate in a Chord network by introducing nodes it controls. These malicious nodes take their respective places in the ring structure and populate other nodes' finger tables. The objective of the adversary nodes is

to disrupt lookups as much as possible: make them fail if possible or make them very slow otherwise. Two basic strategies malicious nodes can use to sabotage lookups are:

- **Non-cooperation.** Malicious nodes do not provide any information to other nodes. They fail to look up nodes, and just forget about their successors: they return no information. As a result, requests are slower, and the structure of the ring is fractured.
- **Flooding.** Malicious nodes, when prompted for a request, provide another malicious node as the reply. This sends the requesting node in a wild goose chase [13], never successfully finishing its request.

Both non-cooperation and flooding can lead to a standard Chord lookup failing. Using the standard Chord strategy a node looking up a target ID tries to make 'progress': the next hop is chosen from among the nodes discovered between the current hop and the target ID. If all the known nodes in this region are non-cooperating, the lookup will fail. Similarly, flooding nodes will provide a set of corrupt virtual nodes with IDs closer to the target, yet never reaching it. In both cases there will be no answer to the query.

In this paper we will attempt to protect DHTs, and a variant of Chord in particular, against random flooding attacks. Our threat model is based on an adversary that aims to disrupt as many queries as possible through the network, and positions its nodes, at random around the Chord ring. Note that targeted attacks could be more easily accomplished by concentrating the dishonest nodes on particular regions of the ring [13]. Targeted attacks that aim to maximally disrupt queries from or to specific nodes are beyond the scope of our study.

In order to address these attacks, we need to modify the iterative Chord lookup. When choosing a next help, our variant will take into account the sources of information about the previous hops, and strive to avoid relying on a single trust bottleneck.

3 The Bootstrap Graph Model

The traditional peer-to-peer model, within which the sybil attack was formulated, views the network as an undifferentiated set of nodes, each with an individual ID. The attacker controls some fraction of these nodes and can cheaply introduce new ones until the network is flooded. Once the fraction of bad nodes exceeds approximately 25%, the system is unable to reliably route queries to the correct ID [2]. So one proposed solution to the sybil attack is to rate limit new nodes joining and to impose a centralized admission control system.

The bootstrap graph adds new elements to the peer-to-peer model that might help tackling the sybil attack without any centralized authority. In most peer-to-peer systems, a new node needs to have a first point of contact with the network in order to join; thus, the nodes in the network must have some previous off-line relationship. We call the set of these relationships the network's introduction graph, or "bootstrap graph". Figure 1 provides an example of such a graph, in which nodes joined in the order of their label numbers.

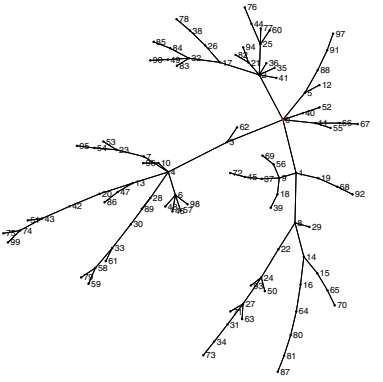

Fig. 1. Example of a bootstrap graph

Now, the goal of a secure overlay network is to enable Alice and Bob, nodes in the network, to communicate with each other if there is any path of good nodes between them in the bootstrap graph. In the DHT context, "communicate with each other" means to be able to look up each others' IDs. This is a good security criterion because if there is no such path between Alice and Bob, then they might as well be in separate DHTs: the only nodes common to the two graph components are malicious, so connecting them would require some out-of-band mechanism, i.e. a new bootstrap graph link.

A "flood fill" routing mechanism provides a proof of concept: if Alice flood-fills the bootstrap graph with her query, it will eventually reach every connected good node regardless of the actions of the adversary. However, this is a very inefficient solution, and it is not resiliant to nodes in the path failing. We'd like to solve this problem using less storage and communication cost than flood filling (which is quadratic in the number of nodes).[1]

The methods presented in this paper assume that the bootstrap graph is a tree, which is typical for current DHTs. We analyze the case of an adversary which has managed to convince one honest node to allow it to join the network, perhaps by social engineering. The adversary can then introduce a large number of sybils into the network via this attachment point. The adversary spends much less effort per sybil than it spent obtaining the attachment point. In section 4.3 we examine what happens when a set of sybils are attached to the honest bootstrap graph at more than one points.

In this paper, we only concern ourselves with routing security, i.e. resolving a particular ID to a node. We assume that if we actually reach the target node, we will be able to verify that it owns the target ID, e.g. using self-certifying IDs [1,8]. We won't discuss the security of data stored on nodes, or the mechanics of data block migration as nodes join and leave the network; these must be left to other security layers.

[1] Insecure DHTs like Chord achieve polylogarithmic cost; it is an open question whether this is possible for a secure overlay.

3.1 Efficient Bootstrap Path Calculation

When the bootstrap graph is a tree, there is an efficient decentralized algorithm for calculating the shortest path between any two nodes. This means that no node is required to know the totality of the bootstrap tree at any time, and the path is constructed as a natural side effect of node lookups.

Each node stores, in addition to IDs and addresses, the path from itself to each node it knows. This includes the node's Chord-ring successor, its predecessor, its finger nodes, and the connections it has in the bootstrap tree. The same is true for all resolutions: the current hop returns not only the ID and address of the next hop, but also the path of bootstrap links from the current hop to the next hop. The path from the querier to the next hop may be computed as the concatenation of the querier-to-current-hop path and the current-hop-to-next-hop path, with any loops removed. In this way, a querier can compute the path from itself to any other node it discovers.

Nodes that first join the network not only have to discover their successor node, predecessor node, and fingers, but also the paths to them. This is done as a side effect of the joining protocol. The new node asks the node it uses to join the network for the IDs of the nodes, and as a result also gets their paths. This allows it to compute its paths to them.

When the bootstrap graph is a tree, nodes can join paths and eliminate all cycles, therefore guaranteeing that they know the shortest path. This is convenient but not necessary for the security properties we will describe next, and extending our algorithms for discovering paths in generic graphs should be possible.

4 Reducing the Impact of the Sybil Attack

Our objective is to devise a resolution strategy that will always succeed, and provide better performance than the standard Chord strategy when under a sybil attack. First we shall deal with the issue of failed queries, then we shall assess the efficiency of our approach.

We modify the standard Chord iterative strategy in the following two ways.

1. A node, when queried, returns all nodes that it knows about, and not simply the closest to the target. The node returns its successor, fingers, and connections in the bootstrap tree. Such a modification requires more bandwidth per query, but does not add any latency to standard iterative lookups.
2. Having a set of nodes returned by each query allows the initiator to be in full control of the resolution, and be able to schedule lookups to maximize efficiency and minimize the potential for disruption from corrupt nodes. A number of query strategies can be used to establish which nodes should be queried and in which order.

We will first look at the standard Chord query strategy that selects nodes according to *closeness* in ID space, and then present a radically different query strategy that routes according to trust *diversity*. These two extremes can be combined, as in the *mixed* and *zig-zag* strategies, to provide fast yet robust lookups.

4.1 Query Scheduling Strategies

The aim of a node that wants to perform a lookup is to select hops that might provide more information about the address of the target ID, and that are not malicious. The basic Chord strategy, which we will call *closeness routing*, is extremely effective at ensuring the first, but does not take into account the second issue (corrupt nodes): given a set of known nodes, Chord chooses to query the one whose ID is closest to the target. When all nodes are honest, the worst case for performance is that the last hop's successor is the only node closer to the target ID. On the other hand, when some nodes are liars, the ring structure is effectively broken. Thus, looking only at nodes between the initiator and the target of the lookup is not guaranteed to succeed. To address this, we need an alternative resolution strategy.

We have assumed that the set of bad nodes are connected to the rest of the bootstrap network through a single good node. We therefore expect that this single good node, along with the bad node it is directly connected to, will always be in the bootstrap graph path from the quering node to the bad nodes. An intrusion detection approach could be used to detect them — this will not be the strategy we chose to implement since nodes that are not always misbehaving might fool it. Instead we will try to balance the number of requests going to bad nodes by making sure that not too much 'trust' is put on any particular node when answering queries. For the purpose of routing we will consider that a node is trusted if it is on the path of the bootstap graph from the initiator of the request to the queried node. The core of our sybil defense mechanism consists of distributing queries around the network in such a way that no small set of nodes is predominantly present on the paths of the queries.

Diversity routing is the purest form of this strategy, and choses nodes to query as following:

1. For each ID lookup the initiator keeps a record of nodes queried. A histogram is computed of the frequency with which each node in the network has been on the path of the queries so far. This can be thought as a 'trust profile' of this particular lookup at any time (Fig 2, step (1)).
2. A node proceeds by answering the follwing question: which node is to be queried next to get more information concerning the node looked up, given the trust profile so far? We associate with each candidate node the 'trust profile' the lookup would have if it was to be used (Fig 2, step (2)).
3. Then the different trust profiles are compared to each other in order to assess which one increases the least the trust put on a single or a small set of nodes. This can be done by sorting the 'trust profile' for each candidate by descending order, and creating a 'trust list' of their values: the first value would be the number of paths the most trusted node was on, and so forth. Then the candidate nodes can be ranked by sorting lexicographically their respective sorted 'trust lists' (Fig 2, step (3)). We then chose the smallest element as the next node to query.

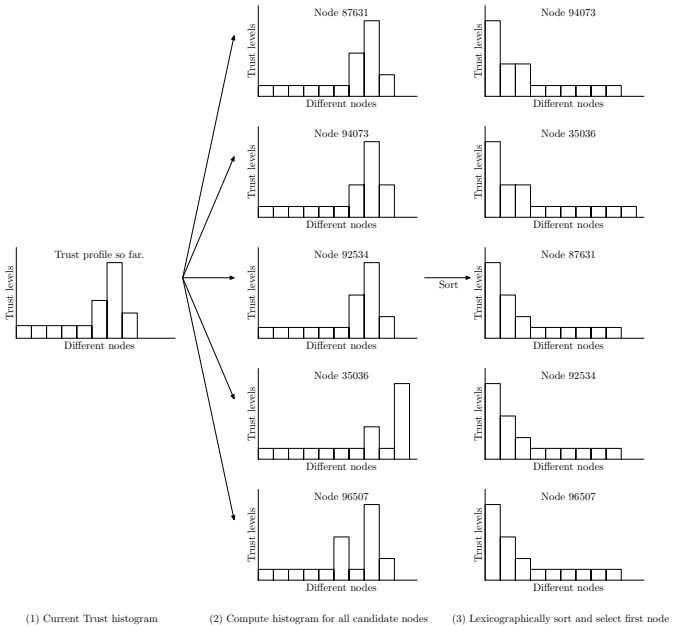

Fig. 2. Illustrating a step of the diversity routing node selection

Table 1. Number of queries to satisfy 100 lookups in closeness and diversity routing (100 good nodes)

Number of bad nodes	*Closeness*	*Diversity*
1	373	552
50	1400	1359
100	3183	2610
200	6977	4807
400	18434	12543

Figure 2 illustrates a step of the diversity routing strategy. Strarting with the nodes that are known to the querying node, this strategy is repeated, until the target ID node is found.

The above strategy tries to distribute queries across the network, taking into account bottlenecks that might indicate a sybil attack point. Adversary nodes that introduce a large number of sybils will aquire high values in the trust profile and nodes behind them will not be used until other nodes in the network are queried. Yet the diversity strategy does not make by itself any progress towards the target node. Note that sybil nodes are not excluded but a balance is maintained between queries to sybils and other nodes.

We expect this strategy to be more efficient than pure Chord when there are more sybils than honest nodes. In particular it will always yield the answer to a query, even if it has to ask the whole network. Still this strategy remains terribly inefficient under normal circompstances. Table 1 shows the number of nodes queried to satisfy 100 random ID requests, where there are 1, 50, 100, 200 and 400 sybil nodes flooding 100 honest nodes.

4.2 Mixed Strategies

In order to maintain some efficiency we need to introduce some bias to choose nodes that are closer to the target. Two strategies have been assessed.

A first approach is to provide a balance between the closeness and the diversity of nodes, and we call this *mixed routing*. This can easily be implemented: given the rank c_i of a node according to closeness, and the rank d_i of the node according to the strategy that provides diversity, and a balance factor $b \in [0, 1]$, we calculate the new rank r_i:

$$r_i = bc_i + (1 - b)d_i \tag{1}$$

Nodes can then be sorted according to r_i in decending order, and the first one chosen to be queried next. Table 2 illustrates the number of queries required to satisfy 100 random lookups in a network of 100 good nodes flooded by 1, 50, 100, 200 and 400 nodes. It is clear that this mixture strategy balances the two key factors, closeness and trust diversity. It provides better results than either of the pure strategies for a lower number of sybils (100 and 200) but does not perform better than the diversity strategy in case there are a lot more sybils than honest nodes.

The best results have been achieved using *zig-zag routing*. Instead of trying to select diverse yet close to the target nodes, as mixed routing attempts to do, the closeness strategies and diversity strategies are alternatively employed. First a node that is close to the target is queried, then a node that is diverse is chosen, and so forth. With each set of diversity routing the pool of known nodes becomes more likely to contain honest nodes, and then the step of closeness routing selects the closest node and queries it for the target.

As table 2 shows zig-zag routing outperforms closeness as the number of malicious nodes grows, as well as mixed routing. Zig-zag routing is also easier to analyze. In the absence of any malicious nodes the lookup will take at most

Table 2. Number of queries to satisfy 100 lookups in closeness, mixed and zig-zag routing (100 good nodes)

Number of bad nodes	*Closeness*	*Mixted ($b = 0.2$)*	*Zig-Zag*	*Good entries in finger table*
1	373	1696	510	99%
50	1400	2172	1291	65%
100	3183	2358	2104	46%
200	6977	4842	3606	30%
400	18434	15110	7004	20%

double the amount of queries to resolve, than using the standard Chord strategy. This is due to the fact that one in two steps implements the Chord strategy. When there are malicious nodes in the network the diversity step ensures that the pool of known nodes retains its quality: it makes sure mostly honest nodes are queried to populate it. On the other hand the closeness step ensures progress towards the target ID, by choosing out of a the pool of known nodes, the closest to the target.

Simulations were run and the histograms describing how many queries (i.e. steps of the iterative routing strategy) were necessary to satisfy 100 requests are plotted in Figure 3. Note that a significant number of requests are satisfied by few (< 10) queries even when a lot of sybils are introduced in the system. Zig-zag routing retains this property as the sybils multiply, while closeness routing becomes increasingly inefficient.

Note that under extreme flooding, the zig-zag strategy (and any strategy based on bootstrap graphs) will be following the bootstrap graph to route between two honest nodes. This makes them fragile against node churn, that could even be the result of malice, and heavy sybil attacks. Providing routing security under such extreme conditions is beyond our scope.

4.3 The Effects of Increased Infiltration

In our analysis so far we have assumed that the set of sybils nodes are attached to the honest part of the bootstrap graph at one honest node only. We briefly assess how our most effective defense mechanism, the zig-zag strategy, handles sybils being attached to multiple points of the bootstrap graph, or in other words an adversary that has fooled more honest nodes.

We performed 100 requests in the DHT, made of 100 good nodes, using the zig-zag and the closeness routing strategies, and record how many nodes have been queried to answer them. We repeated the experiment for 100 and 200 additional bad nodes in the network, connected to $1, 10, 20 \ldots 90, 100$ distinct good nodes. The rest of the bad nodes were only introduced by these 'attached' bad nodes. Figure 4 summarises the results.

In the experiment with 100 bad nodes (Figure 4, black lines) we observe that the zig-zag strategy outperforms the standard closeness strategy until more than 80 bad nodes have infiltrated the network. For higher values closeness (represented by the straight line) outperforms the zig-zag strategy, which is due to the overhead it introduces: it only makes progress in one out of two steps.

On the other hand as the overall number of sybils increases, as in our experiment with 200 bad nodes (Figure 4, dotted lines), our zig-zag strategy outperforms the standard closeness strategy even when the adversary has managed to infiltrate and connect to every single one of the 100 honest nodes. It is encouraging that the number of requests 'stabilises' for more than 30 infiltrated nodes, and infiltrating more of them does not seem to degrade the performance of the network further. An adversary that chooses to infiltrate more and more honest nodes will therefore experience diminishing returns.

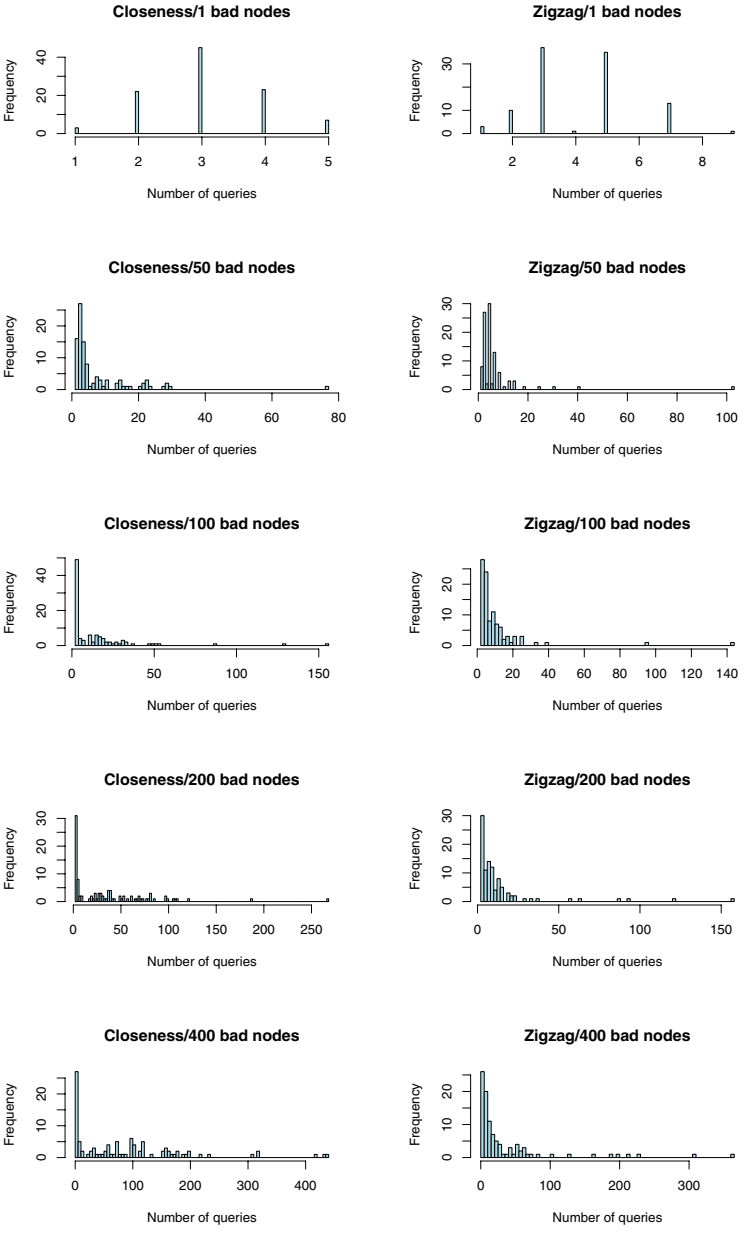

Fig. 3. Closeness and Zig-Zag routing

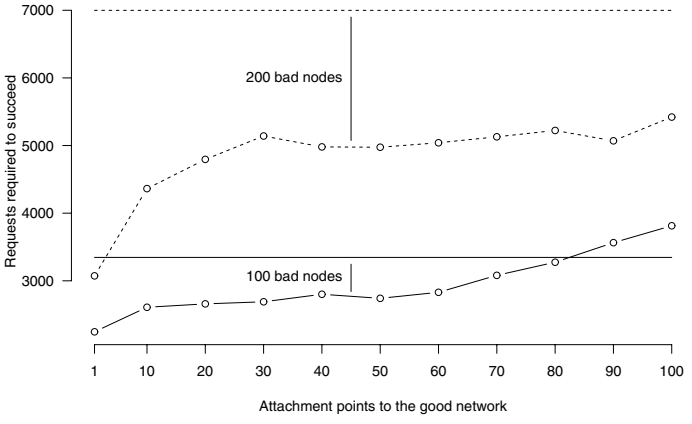

Fig. 4. The number of requests required to satisfy a query increase with the number of sybil nodes attached to the honest network. The network used was composed of 100 honest nodes, and 1 to 100 attachment points for 100 and 200 sybils. In each case the horisontal line denotes the standard closeness strategy, while the other set of points denotes the zig-zag strategy under increased infiltration.

These initial simulation results indicate that there is an optimal number or percentage of infiltrated nodes to disrupt the network, after which the adversary does not get much advantage. Still we cannot do away with our initial assumption that the bootstrap graph should contain a connected components with all honest nodes, otherwise an adversary would be able to split the network in many separate ones to better attack it.

4.4 Balanced Finger Tables

The diversity based strategy, and the zig-zag strategies, make sure that a fair share of good nodes are selected to answer a query. Yet the quality of the information provided, even by good nodes, decreases as more sybils are introduced in the network. The reason for this is that their finger tables are populated with an increasing number of sybils that will not contribute to answering the queries.

The solution to this is to select fingers according to a strategy that ensure that good nodes are still present. A variant of the strategy based on the ordered 'trust lists' described above in the context of lookups can be used for that purpose.

Chord fingers are distributed around the ring in a manner that maximizes the efficiency of lookups. The original design is very deterministic and requires nodes to pick fingers half way across the ring, a quarter across, an eight, and so forth. The idea behind this distribution is that nodes will know mostly about their immediate successors, but also some far away nodes to make far lookups efficient.

We propose a sybil-resistant finger distribution: A set of 32 fingers are selected with an exponential distribution around the ring. This reflects the Chord

Table 3. Number of queries to satisfy 100 lookups in closeness, mixed and zig-zag routing (100 good nodes) with diverse routing tables

Number of bad nodes	*Closeness*	*Mixted ($b = 0.2$)*	*Zig-Zag*	*Good entries in finger table*
1	309	1634	413	99%
50	809	1962	725	66%
100	1519	1503	1056	53%
200	3581	1801	1400	37%
400	8762	5873	3627	26%

paradigm, of discovering more about the immediate environment. Out of these fingers the 16 are selected in the following manner: the successor is first used to create a 'trust profile' of the table. Then all the candidate fingers are assessed to find out which would least increase the trust put into a small set of nodes, using the 'trust list' strategy described above. The procedure is repeated with all selected fingers contributing to the 'trust profile', until 16 fingers have been chosen.

Requests routed using these more trust 'balanced' finger tables exhibit a better performance as illustated in table 3. Note that the percentage of corrupt finger entries is lower than in table 2, when the strategy described was not in use. The efficiency results are positively correlated to the degree of finger table corrution.

5 Conclusions

Distributed Hash Tables are very efficient distributed systems to lookup identifiers, and in the past it has been demonstrated that they can be made robust against node churn, random failures, and fluctuating network conditions [6,11]. We devise strategies that make DHTs resilient to malicious nodes trying to poison lookups by providing inaccurate information. We achieve this by routing queries, not only to make them converge fast to their destinations, but also in a way that minimizes trust bottle necks. This minimizes the amount of poisoned information that honest nodes receive from hostile sybils controlled by the adversary.

The strategies we present have been validated through extensive testing and simulations, whose results have been presented. It is worthwhile noting that routing is still possible, and more efficient than broadcasting, even when honest nodes are in a small minority (our tables illustrate the ratio of 1 honest node to 4 sybils). We have also validated through simulation that our approach would protect the network, even if a large number of sybil nodes manage to infiltrate the network by fooling many honest nodes into introducing them. Furthermore one could describe our approach as 'value free', in that there is no attempt to classify nodes into good and bad: we simply try to spread the queries across all nodes in the trust graph. Intrusion detection strategies could be devised that

correlate the quality of the information provides with nodes, to determine which are the bad nodes. Making such a mechanism strategy proof is a hard problem.

Our algorithms also refrains from making global judgments about nodes: there is no such thing as a good node or a bad node, but only nodes that are connected to the requesting node through different paths. As a result we expect our algorithms to be of use when two mutually hostile groups of nodes decide to form a common DHT. While members of one group might provide poor, or no information, to members of the other group, they would behave properly to each other. Our approach should be able to cope with this model.

Finally we hope that this work contributes to a redefinition of 'identity' when used in a distributed systems security setting, as the position of a party in a social, or other, network, rather than an arbitrary external identifier. The traditional identity based approach requires additional infrastructure to assign identities, such as admission control and public key infrastructures that are expensive and difficult to implement in any network, let alone in a fully decentralized peer-to-peer setting. We have shown that preserving this contextual information can yield simpler and more robust mechanisms for dealing with adversaries.

Acknowledgments. George Danezis and Chris Lesniewski-Laas are supported by the Cambridge-MIT Institute (CMI) project on 'Third generation peer-to-peer networks' and part of this work was done while visiting MIT CSAIL.

References

1. Tuomas Aura, Aarthi Nagarajan, , and Andrei Gurtov. Analysis of the hip base exchange protocol. In *10th Australasian Conference on Information Security and Privacy (ACISP 2005)*, Brisbane, Australia, July 2005.
2. Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *5th Usenix Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.
3. John R. Douceur. The sybil attack. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, Cambridge, Massachusetts, March 2002.
4. J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *32nd ACM Symposium on Theory of Computing*, 2000.
5. Raph Levien. Attack resistant trust metrics. Draft Ph.D. Thesis, at U.C. Berkeley.
6. J Li, J Stribling, TM Gil, R Morris, and F Kaashoek. Comparing the performance of distributed hash tables under churn. In *International Workshop on Peer-to-Peer Systems (IPTPS04)*, 2004.
7. Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. SPROUT: P2P routing with social networks. In *First International Workshop on Peer-to-Peer and Databases (P2P&DB 2004)*, March 2004.
8. David Mazires. *Self-certifying file system.* PhD thesis, MIT, May 2000.
9. Roger M. Needham. Denial of service: an example. *Communications of the ACM*, 37(11):42–46, 1994.

10. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM 01*, San Diego, California, August 2001.
11. S Rhea, D Geels, T Roscoe, and J Kubiatowicz. Handling churn in a dht. In *USENIX Annual Technical Conference*, June 2004.
12. Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Heidelberg, Germany,*, Heidelberg, Germany, 2001.
13. Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS 02)*, Cambridge, Massachusetts, March 2002.
14. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, , and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM 01*, San Diego, California, August 2001.
15. Ben Y. Zhao, John D. Kubiatowicz, , and Anthony D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.
16. Cai-Nicolas Ziegler and Georg Lausen. Spreading activation models for trust propagation. In *IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE '04)*, Taipei, Taiwan, March 29-31 2004,.