ReDS: A Framework for Reputation-Enhanced DHTs

Ruj Akavipat, Mahdi N. Al-Ameen, Apu Kapadia, *Member*, *IEEE*, Zahid Rahman, Roman Schlegel, *Member*, *IEEE*, and Matthew Wright

Abstract—Distributed hash tables (DHTs), such as Chord and Kademlia, offer an efficient means to locate resources in peer-to-peer networks. Unfortunately, malicious nodes on a lookup path can easily subvert such queries. Several systems, including Halo (based on Chord) and Kad (based on Kademlia), mitigate such attacks by using redundant lookup queries. Much greater assurance can be provided; we present Reputation for Directory Services (ReDS), a framework for enhancing lookups in redundant DHTs by tracking how well other nodes service lookup requests. We describe how the ReDS technique can be applied to virtually any redundant DHT including Halo and Kad. We also study the collaborative identification and removal of bad lookup paths in a way that does not rely on the sharing of reputation scores, and we show that such sharing is vulnerable to attacks that make it unsuitable for most applications of ReDS. Through extensive simulations, we demonstrate that ReDS improves lookup success rates for Halo and Kad by 80 percent or more over a wide range of conditions, even against strategic attackers attempting to game their reputation scores and in the presence of node churn.

Index Terms—Peer-to-peer, DHTs, security, reputation, distributed systems, systems and software, reliability, availability

1 Introduction

Over the past several years peer-to-peer (P2P) systems have been gaining popularity and mainstream acceptance. For example, Skype, the popular P2P-based system, had 55 million concurrent online users in April 2013.[1] BitTorrent (http://www.bittorrent.com/) and even botnets are large P2P systems that must achieve decentralized coordination to locate resources. For example, in Skype, one must be able to locate the current IP addresses of contacts, and in a distributed storage system, one must be able to locate the IP address of a node hosting a particular file. One class of solutions called *distributed hash tables* (*DHTs*) maps resources onto nodes in the P2P network and provides a "put-get" abstraction where resources can be stored (put) in the network and subsequently retrieved (get). The key idea in DHTs is that each peer maintains a routing table with only a few entries and yet any resource can be located by routing queries through a few nodes, where "few" usually corresponds to a number logarithmic in the number of nodes in the network. Chord [1], CAN [2], Pastry [3], and Kademlia [4] are examples of DHTs with these properties.

DHTs provide several important properties, such as scalable location of nodes and services, but do not protect against malicious peers manipulating *lookups*, the operations used to locate resources in the system. For example, an attacker may want to undermine the system's operations by providing fake lookup results for nonexistent peers or to make his own peers the end point of lookups so as to pollute the network's files and services. Such an attacker can easily manipulate much of the system's activity. In Chord, for example, only about 10 percent of malicious nodes in the network can subvert more than 50 percent of the searches [5]. Halo is a system that exploits the deterministic mapping of routing-table entries to nodes in Chord to provide a "high-assurance locate" through redundant searches [5]. Several other DHTs, such as Salsa [6], Cyclone [7], and NISAN [8], also utilize redundant searching to tolerate malicious nodes in the network. Kad (based on Kademlia) is an example of a nondeterministic DHT that also incorporates redundancy into the protocol; routing-table entries are a function of nodes encountered in the system and are not easily predictable.

While all these techniques are able to improve the success of lookups by a combination of redundancy and diversity of the redundant lookup paths, they still allow a nontrivial failure rate while incurring substantial overhead for redundancy. For example, Halo still has a failure rate as high as 5-6 percent for 20 percent malicious nodes utilizing a logarithmic number of redundant lookups in the size of the network (e.g., 13 lookups in a network of 10,000 nodes). We show that Kad has a nontrivial failure rate of 17-21 percent with only 10 percent malicious nodes, even with high redundancy.

1. Skype Numerology. http://skypenumerology.blogspot.com/.

- R. Akavipat is with the Department of Computer Engineering, Mahidol University, Salaya, Nakornpathom, Thailand.
- M. N. Al-Ameen and M. Wright are with the Department of Computer Science and Engineering, University of Texas at Arlington, 500 UTA Blvd., Room 640, Arlington, TX 76019.
- A. Kapadia and Z. Rahman are with the School of Informatics and Computing, Indiana University Bloomington, 150 S Woodlawn Ave, Bloomington, IN 47405.
- R. Schlegel is with Corporate Research, ABB Switzerland Ltd., CH-5405 Baden-Dättwil, Switzerland.

Manuscript received 16 Sept. 2012; revised 26 Aug. 2013; accepted 27 Aug. 2013; published online 6 Sept. 2013.
Recommended for acceptance by X. Li, P. McDaniel, R. Poovendran, and G. Wang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2012-09-0936.
Digital Object Identifier no. 10.1109/TPDS.2013.231.

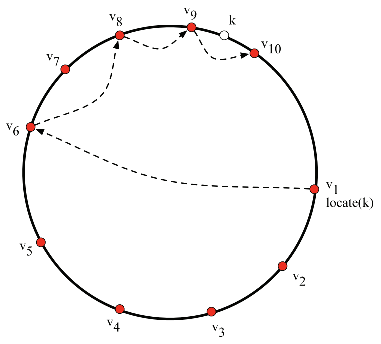
We investigate an approach to improve DHT lookups in the face of malicious peers. Our central observation is simple: if a node uses redundant lookups and tracks which nodes gave accurate results, then it can use this information to improve the success rate of lookups that traverse it.

Our design approach based on this observation, Reputation for Directory Services (ReDS), includes two novel features. First, the querying node uses the redundancy in the lookups and structure of the DHT to infer the honesty and reliability of nodes throughout the lookup path, even though direct observation is not possible. Second, peers employ *collaborative boosting*, in which each node involved in the lookup can improve the success of the route by picking the next hop based on a form of constrained local boosting.

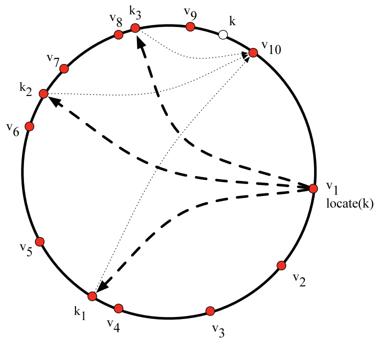
We note that quite a bit of work has been done on P2P reputation systems (Hoffman et al. [9] provide an extensive survey). However, this prior work mainly addresses the "free-rider problem" in which some users unfairly use resources provided by peers without providing any resources themselves. This issue is orthogonal to our work, which instead leverages reputation to detect malicious behavior that aims to undermine DHT routing. While we are able to leverage some of the findings of other work on reputation systems, identifying malicious behavior in the DHT routing layer presents unique design challenges that we address.

Contributions. Preliminary results on ReDS were published as a work-in-progress paper examining ReDS in the context of Salsa [10] and a workshop paper examining Halo-ReDS under a limited adversary model [11]. Here, we make the following additional significant contributions:

- We show that reputation can be applied to a variety of redundant DHT-based directory services to improve lookup success rates. We specifically describe *Halo-ReDS* and *Kad-ReDS*, which are implementations for Halo (a deterministic DHT) and Kad (a nondeterministic DHT).
- Building on our approach of using a *reputation tree* to make statistical inferences about where malicious nodes reside in the DHT, we show how nodes along a lookup path can make use of their local reputation trees for *collaborative boosting*. We show a dramatic improvement in success rates for this mode.
- Through analysis and simulation, we study the behavior of Halo-ReDS and Kad-ReDS under adaptive adversaries who attack only some fraction of the time in an effort to game the reputation system. In particular, we show that attackers are limited to attacking at a low, ineffective rate.
- We evaluate the performance of Halo-ReDS and Kad-ReDS under churn and show that while adaptive inference suffers with churn, collaborative ReDS is more robust.
- We examine the possibility of sharing reputation scores and show how such sharing can be attacked through slandering and self-promotion attacks. Further, we identify a new "use-based" attack on shared reputation that would greatly undermine most ReDS systems, leading us to recommend using only firsthand observations.



(a) A Chord lookup progressing through the DHT from querying node v_1 to target key k, with each hop cutting the remaining distance to the destination at least by half.



(b) Using the Halo technique, node v_1 performs three redundant "knuckle searches" (each of these is a regular Chord lookup, abstracted by dark arrows) yielding knuckles k_1, k_2, and k_3, which directly provide the location of v_{10} (indicated by lighter arrows).

Fig. 1. Overview of the lookup process in Chord and Halo.

2 System and Attack Models

2.1 System Model

In a DHT objects indexed by keys are stored on and retrieved from peers using put (key, object) and get (key) operations, respectively. In both operations, the DHT must first perform a *lookup* operation to map the key to an *owner* node o that stores the object. We describe the rest of the system model in the context of Halo and Kad.

Halo and Chord. In Chord, nodes are assigned to a virtual address space that is organized in a ring (see Fig. 1). For example, the address space could correspond to the output of SHA-1, and the next address after $2^{160} - 1$ is 0 again. IDs can be issued to nodes via a central authority along with a certificate [12] (we discuss how attackers can be prevented from manipulating their IDs in Section 2.2). Resources, such as files, can be assigned virtual addresses (the resource's key) based on the hash values of their filenames. A resource's owner is the *clockwise-closest* in the virtual address space, i.e., the node with the lowest ID greater than the target ID, modulo the size of the ID space. Each node maintains a routing table of nodes called "fingers" that are at exponentially increasing distances from itself. When a node receives a locate request for a target key t, it redirects the query to the closest finger to t. This process results in efficient lookups with $O(\log n)$ hops requiring only $O(\log n)$ storage at each node, where n is the total number of nodes in the DHT (see Fig. 1a).

While Chord has properties that promote stability under independent node failures, lookups are easily subverted. Simply adding redundancy to lookups does not help much, as lookups often converge to the same nodes. Halo takes advantage of the fact that each node v occurs in $O(\log n)$ other nodes' finger tables [5]. These nodes are called the "knuckles" of v. Searching for those knuckles instead of the target effectively disentangles the redundant searches (see Fig. 1b). Note that because of the clockwise-closest relation, if a redundant search yields multiple candidates for the target's owner, the closest one (that is responsive) is picked. Thus, as long as one of the lookups in a redundant search returns the correct answer, the correct owner is obtained.

Kad. Kad is a widely deployed DHT based on Kademlia [4]. Distances in the Kad ID space are measured by computing the XOR of two IDs and taking the output as an integer. In Kad, each node maintains a routing table comprised of a "k-bucket" for each exponentially increasing interval of ID space from the node. Each k-bucket includes up to k nodes from the corresponding ID range and is dynamically populated by new nodes encountered in each put-get operation, resulting in nondeterministic routing table entries. More specifically, the jth k-bucket of a node contains learned nodes for which it shares the first j bits of the ID and has a different $j+1$-st bit. If a k-bucket is full, then the least recently seen node is evicted to make space for a new node.

Kad lookups proceed *iteratively*, where each node contacts α nodes at each step and receives the β closest results from each of them. A short list of k nodes is maintained by the querying node, and the list is updated with the $\alpha \times \beta$ results returned at every step. At the next step, the querying node contacts the closest α unqueried nodes drawn from the short list. Kad ensures $O(\log n)$ lookup steps by moving at least one bit closer to the target ID with each iteration.

In Kad, a "resource" is stored on r different nodes (called *replica roots*) around the key such that their Kad ID falls within a specified distance, the *search tolerance* δ. Typically, $r = 10$ and δ is set so that the Kad ID of a replica root agrees at least in the first 8 bits of the key.

2.2 Attack Model

Malicious nodes in the system may attempt to subvert lookup operations, for example, by dropping or misdirecting lookups. The adversary's goal could be to cause peers to use attacker-controlled nodes, for example, as a way to spread disinformation, spam (e.g., a marketing message in an MP3 or video file), or malware. Adversaries may also seek to censor access to content through denial-of-service or degradation-of-service attacks, in which lookup results lead to invalid or incorrect nodes. The attacker's most effective strategy to achieve these ends in a P2P network is to control a large number of peers in the system (or virtual peers by controlling its location in the address space).

To prevent the number of malicious peers from growing without bound, we can leverage decentralized, social network-based anti-Sybil techniques, such as SybilInfer [13] and SybilLimit [14]. Such defenses limit an adversary to a small number of malicious nodes for each connection he can socially engineer to honest users in the social network. We, thus, expect that the attacker may be able to socially engineer enough connections to inject a constant fraction (e.g., up to 20 percent) of the total number of peers into the network without detection. This requires that users must be at least somewhat vigilant against connecting to strangers. *Interaction graphs* have recently been proposed to ensure that connections represent some mutual activity [15].

We note that guarantees on the fraction of malicious peers are also required for reasons orthogonal to lookup routing. For example, if fraction f of the nodes are malicious, then f of the *correctly routed* lookups will find a malicious owner node that can provide malicious or spam content or deny the request. Thus, effective Sybil defense is necessary regardless of its impact on lookup routing.

An attacker who controls a significant fraction of malicious nodes would seek to both manipulate lookup results and deceive reputation systems. Thus, our design must account for both types of attacks. We assume that lookup operations going through a malicious node will be manipulated to map the key to the closest malicious node instead of the actual owner. We also assume the attackers can choose to attack only a fraction of the time in an attempt to evade detection. For an *attack rate* a, adversaries will attempt to compromise a particular lookup(t) with probability a. We assume powerful adversaries who can coordinate their attacks by exchanging information in real time to flag t as a target that should be attacked or not.

A standard assumption we make is that malicious nodes cannot control their placement in the ring, and thus, malicious nodes are distributed uniformly at random in the address space. To ensure that an attacker cannot manipulate its location in the ID space, the central authority should issue the node a random nonce and make the ID of a node x be a function of x's public key PU_x and the nonce N, for example, $ID_x = H(PU_x, N)$, where H is a cryptographic hash function such as SHA-1. Further, the authority must limit the number of ID requests from any entity by, for example, requiring a valid credit card number, verifying a valid mobile phone number by text message, or using a Sybil-resistant admission-control protocol [16]. Peers can verify the virtual addresses of nodes by checking signatures; for example, Myrmic [17] provides such assurances for Kademlia.

We assume attackers will attempt to pollute routing tables to increase the fraction of attackers in the tables. In the context of Halo, control and regular lookups follow the same process, and our technique for collaborative boosting provides high success rates under malicious behavior and, thus, ensures minimal chances of attackers gaining any extra influence in the system. Since routing tables are updated during regular lookup operations in Kad, we describe a new attack on Kad routing tables and show how our approach effectively limits routing table pollution in Kad (these experiments are detailed in Section 6 of the online supplement, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.231.

3 ReDS Design

We now describe ReDS, our approach to augmenting DHTs, like Halo and Kad, so that nodes can utilize the successes and failures of individual lookups in a redundant search to infer malicious nodes. ReDS then directs lookups to avoid these nodes in two steps: 1) the originator of a lookup picks the best possible start nodes ("local boosting"), and 2) each node involved in the lookup selects high-performing fingers ("collaborative boosting"), thereby avoiding malicious fingers at every step.

We begin with a brief overview of the ReDS idea and then explain how we apply this approach to Halo and Kad.

3.1 Overview

ReDS can be applied to DHTs that meet these requirements:

1. Redundant lookups can be performed with diverse lookup paths.
2. Every querying peer has a choice of peers for each entry (i.e., finger) in its routing table. We call the set of peers available at each routing table entry the k-bucket, adapting the terminology from Kademlia.
3. Given a set of targets based on the redundant search, it is possible to select the correct target node within that set if it exists.
4. The success or failure of a particular sublookup can be linked to the first finger in that lookup.

The first requirement is the basis for systems like Halo to provide robust lookups against moderately strong attackers. Without redundancy and path diversity, the system's lookup success rate will be unacceptably low [5], [6], and ReDS may not be able to distinguish between honest and malicious peers. For the second requirement, it is important that the choices for each k-bucket all meet the basic routing requirements of the DHT. In particular, any node in the k-bucket can cut the remaining distance to a target by half. This requirement allows for preferential selection of nodes within a k-bucket by avoiding malicious fingers while maintaining path-length guarantees of lookups.

The third requirement allows a querying peer to assess the set of targets returned by the various sublookups and pick an honest target if it exists in the returned set. In most DHTs, where ownership of a resource is by nodes who are closest to the resource's key, the closest target to the search key can be considered to be a success. If the closest node in the target set is honest, then it is guaranteed to be selected. ReDS treats all targets that are not the closest to the target as a "failed" search.[2] Note that we do not base reputation scoring on application-level content or services, but only on the information available to the routing mechanism.

The fourth requirement allows for a mechanism to attribute successes or failures of a sublookup to particular fingers. In some DHTs (e.g., Halo), this mapping is obvious because each sublookup proceeds independently, but in other DHTs (e.g., Kademlia), redundancy is built into the search, and greater care is needed to avoid misattribution.

2. In all DHTs that we know of, the owner of the key being looked up will be closest peer to that key in the ID space. In Kademlia and Kad, there may be multiple owners, but the closest peer should be one of them.

A system that meets these requirements, or can be modified to meet them (e.g., Halo [5] modifies Chord [1] to meet the first requirement), can apply ReDS as follows: First, each peer should track the success rates of their own lookups through each finger. The success rate through each finger is used to calculate a reputation score for that finger. Second, the requesting peer should use these reputation scores to pick the peer with the highest reputation from each k-bucket for subsequent lookups. With enough reputation information, the failure rate at each hop in the lookup is expected to drop significantly because all nodes in the k-bucket must be malicious to subvert a lookup. ReDS, thus, *boosts* the lookup success rate for that peer.

We distinguish between *local boosting*, in which a single peer uses ReDS to improve its own lookups, and *collaborative boosting*, in which all of the honest peers use ReDS to improve both their own lookups and others' lookups through them. Note that boosting is helpful to the peer's own lookups, so it is incentive-compatible to assume that all peers would collaborate in this way.

To better illustrate the ReDS design and show its generality, we now describe how ReDS can be applied to Halo and Kad, two very different DHTs—Halo has a deterministic mapping of nodes to routing tables, whereas Kad has a nondeterministic mapping. Also, redundant searches are performed independently in Halo, whereas the results of redundant searches in Kad are combined iteratively.

3.2 Halo-ReDS

Given the general description of ReDS above, we now explain how we adapt Halo to be suitable for ReDS. Halo was designed to have robust lookups through redundancy and path diversity [5], thus meeting our first requirement. Lookups in Halo are mostly *recursive* in that the querying node simply asks its fingers for the knuckles of a target, and success or failure in finding a knuckle can easily be traced to a finger. Thus, Halo meets our third requirement. The main challenge in applying ReDS to Halo is that Halo (which is based on Chord) has a fixed set of fingers without any choices. This violates our second requirement that each peer have a choice of fingers in a k-bucket.

We, therefore, create k-buckets, one for each location where a finger would be in the Chord ring. The k-bucket for a given finger's key $(v + 2^i)$ includes that key's owner (the original Chord finger) and the $k - 1$ predecessors of that node. When the closest finger for a target key t is requested from node u, u checks its reputation scores for the fingers in the k-bucket closest to t and selects the best finger in the bucket for that request. With this modification, we can apply ReDS as described above. Additional system details and an analysis of resultant path lengths are described in Section 3 of the online supplement—we show that the path lengths increase by at most one hop on average.

Halo-ReDS computes reputation scores as the fraction of lookups that have succeeded through a particular finger. A more sophisticated approach called "A-Boost" maintains these fractions for various target regions in a *reputation tree* to improve selection based on a target region. The intuition behind this approach is that if all of the lookups through a finger fail for one part of the ID space but usually succeed elsewhere, then the failing part of the ID space is likely to

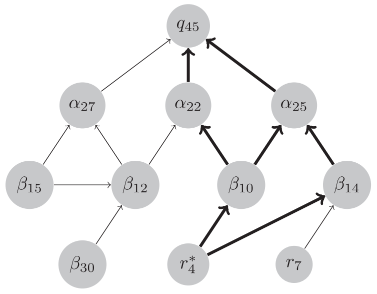


Fig. 2. Lookup graph for a lookup initiated by node q_{45} for key. Subscripts denote the XOR distance of the node from the lookup target key, i.e., $XOR(q_{45}, key) = 45$. Edges from a node are directed toward the node from which they are returned during the lookup process. Three successful paths from a correct replica root r_4 are shown in bold.

represent a malicious node further along in the lookup. Thus, with additional per-finger tracking, fingers can be avoided on just the regions of ID space for which they fail. See Section 2 of the online supplement for more details on A-Boost.

3.3 Kad-ReDS

Unlike Halo, Kad inherently has k-buckets that we can easily leverage to satisfy the second requirement for applying ReDS, which is choice of fingers. Kad lookups are also inherently redundant over diverse paths, as each step of the lookup includes multiple peers and each peer's routing table is set through opportunistic connections. This meets our first requirement. The main issue with applying ReDS to Kad is that Kad is an *iterative* DHT, in which each step of the lookup involves asking multiple peers for their fingers that are closer to the target. The querying node selects the closest few of these peers to continue the lookup. Some results between peers may overlap. Thus, mapping the lookup results to the fingers in the querying node's k-bucket (our third requirement) is nontrivial.

To address this issue, we have the querying node construct a *lookup graph* (see Fig. 2) that tracks the paths used to find the target owner during the lookup process. We describe the exact algorithm in Section 4 of the online supplement. The main insight is that once a target node has been determined (r_4), the querying node (q_{45}) can determine the lookup paths (indicated by the darker edges) whose nodes will be credited with $+1$ to their reputation score (a node on multiple such paths will get multiple credits). We assume that attackers will try to game this process by polluting routing tables but eventually providing the correct target at the end of the lookup to result in positive reputation for the malicious nodes. We explain the details of this attack and its limitations in Sections 4 and 6 of the online supplement.

3.4 Handling Churn

In our simulations (see Section 4), we evaluate ReDS under a dynamic network with churn. In large P2P systems, peers leave and rejoin the system at irregular intervals. This churn makes relying on predictions based on past behavior inaccurate at larger time scales. The attacker can also modulate the behavior of his peers to manipulate the reputation system. We explored techniques to cope with churn, such as exponentially weighted moving average (EWMA), but as we show in Section 4.2, it is effective to update the scores with equal weight to older and newer results. EWMA is still recommended to deal with oscillation attacks, as described in our security analysis in Section 8 of the online supplement. We also considered various exploitation-versus-exploration tradeoffs, but deterministically picking the node with the highest A-Boost score provided the best results. Since all members of a given k-bucket have the same initial reputation scores, one of them will be picked at random at the beginning. If the selected node loses reputation because of failed searches, another node is picked. Eventually, all nodes are explored if they all display failures, and thus, we found both exploitation and exploration taking place on an as-needed basis. Our findings for exploration validate analysis in Section 8 of the online supplement.

3.5 Shared Reputation Scores

An intuitive idea for improving ReDS is for peers to share reputation information with each other. Shared reputation is beneficial for and even a central part of many reputation systems in the context of free-rider prevention [9]. We, thus, explore how shared reputation could work in ReDS and how well it would work.

Unlike reputation systems in many contexts, ReDS peers cannot make use of reputation information shared by arbitrarily selected peers. They can only use reputation from nodes who share the same fingers. We, thus, aim to identify and maintain a list of the nodes with shared fingers and regularly share reputation information with them.

Specifically, we worked out a shared reputation scheme for Halo, which has deterministic finger selection. We expect that shared reputation in Kad will be significantly harder, since k-buckets are populated opportunistically. As we show in our experiments and analysis, shared reputation is ineffective in the face of malicious reputation sharing, and thus, we do not attempt to devise a scheme for Kad. In the context of Halo, we can define two nodes that share the same finger f to be *joint knuckles* of each other for finger f. In this approach, each node maintains a list of joint knuckles for each of its fingers. The list can be maintained by periodically performing the knuckle search on each finger, which is already a Halo primitive. The node then incorporates the scores of these joint knuckles into a score for its finger. We divide the scheme into two phases: I) sharing reputation scores with joint knuckles, and II) calculating the shared reputation scores.

Phase I: *Sharing.* We divide time into epochs based on the assumption of loosely synchronized clocks (e.g., with the Network Time Protocol (NTP)). At the beginning of each epoch t_i, a node compares its firsthand reputation score with its score from epoch t_{i-1}. If the score has changed, it broadcasts the updated score to its joint knuckles.

Phase II: *Calculating reputation.* After receiving all updated scores, the node can calculate the shared reputation score for each of its fingers. The average reputation score is vulnerable to self-promotion and slandering attacks. Wagner [18] goes into great detail on aggregation methods that are suitable for security applications, finding that median is a strong solution. We note, however, that median *always* fails when the number of attackers is more than 50 percent. Having so many attackers among one's joint knuckles is possible due to the small sample size. Since we have our own reputation information collected from local observations, we can do better.

In Section 7.1 of the online supplement, we describe a novel scheme for reputation aggregation called "Drop-Off," in which scores close to the node's own local scores are more likely to be considered for a final aggregation step. The basic idea of Drop-Off is to probabilistically select scores based on their closeness to the node's local score and apply the median to the selected scores. We find that Drop-Off performs better than median in our setting. The simulation results presented in Section 4.4 bear out our analytical findings, but they also show the limited benefits of sharing overall. Further, we present an attack on shared reputation for ReDS in Section 7.2 of the online supplement.

3.6 Communication Overhead

To withstand malicious behavior, various approaches have advocated the use of redundant lookups in DHTs [5], [6], [7], [8], [12]. Redundant lookups add overhead to lookups by a constant factor equal to the amount of redundancy used. ReDS can, for a particular robust DHT, provide similar security assurances at lower levels of redundancy, or much better security assurances at the same level of redundancy. We mainly focus on the latter in this paper, but as an example of the former, we note that a recursive version of Halo can also provide high assurance with up to 22 percent attackers but with the *square* of the typical redundancy (e.g., 169 for a redundancy of 13).

ReDS biases lookups toward honest nodes and away from attackers, which puts more of the system load on the honest nodes. Note that in a system with no attackers, however, the load due to the honest nodes would already rest entirely on the honest nodes. The load due to attackers that are not servicing requests due to low reputation can be addressed by solutions to the freeloader problem [9].

As described in Section 3 of the online supplement, the use of predecessors in the k-buckets leads to minimal additional hops in the lookup path (e.g., 0.15 hops per lookup). Finally, while the use of shared reputation increases communication overhead, we show that shared reputation does not offer any security advantages and, thus, advocate against its use. There are no other communication overheads. Thus, ReDS significantly outperforms existing approaches at similar or reduced levels of communication overhead.

4 Experimental Evaluation

We now describe our experimental setup and present results from extensive simulations of Halo-ReDS and Kad-ReDS.

4.1 Experimental Setup

We built simulators for both Halo-ReDS and Kad-ReDS in Java. Each simulator includes the basic lookup mechanism of the network,[3] the A-Boost reputation tree for each node, collaborative boosting, a model for node churn, and attacker models specific to the network (see Section 2.2). For A-Boost, the A-Boost scores are used only by the querying node. For collaborative boosting, A-Boost scores are used at intermediate nodes as well.

Setup for Halo-ReDS. All our simulations for Halo-ReDS were run for networks with 1,000 nodes.[4] In our experiments, we use a redundancy of 10 as suggested for regular Halo with 1,000 nodes in the network [5].

Setup for Kad-ReDS. Most of our simulations for Kad-ReDS were run for networks with 10,000 nodes. The largest simulation was run for 100,000 nodes. In Kad and Kad-ReDS, we initialize the system with n_l lookups per node to populate the k-buckets. We used $k = 10$ and $\alpha = 7$ redundancy for most simulations.

Routing-table pollution. As discussed in Section 3.3, we impose an attacker model on Kad that includes routing-table pollution, and we modify the bucket replacement policy to replace low reputation nodes. For Halo-ReDS, we do not add a separate layer of activity in our simulations for control lookups, because a control lookup operates exactly the same as a regular lookup; when success rates are high for lookups, routing tables will face minimal pollution.

Churn. To evaluate how the network handles node churn, we add and remove nodes probabilistically after each lookup (which are treated as atomic operations). The probability of a given node joining or leaving after a given lookup is set based on the intended churn rate for that simulation run. For example, in a simulation with $n = 1{,}000$ nodes, a colluding fraction of $c = 20\%$, $l = 250$ training lookups, and a churn rate of $r = 25\%$ over the whole simulation (i.e., in a network with 1,000 nodes, on average 250 nodes leave the network and 250 new nodes join the network over the course of the simulation), the probabilities for a single node are $p_{leave} = p_{join} = 0.00125$, calculated as

$$p = \frac{1}{\frac{l \cdot (1-c) \cdot n}{r \cdot n}} = \frac{r}{l \cdot (1-c)},$$

where the $(1-c)$ stems from the fact that only honest nodes do training lookups.

Nodes chosen for lookups. In a simulation, every honest node is selected as a querying node in turn, ordered according to a random permutation that is repeated as necessary. For A-Boost and collaborative boosting, this helps to build the reputation trees of all the honest nodes. Nodes use the *deterministic maximum score* as described in Section 3.4 to select fingers for routing.

Shared reputation. When shared reputation is used, all honest nodes are trained and queried similarly, with the addition that nodes gather shared reputation information from joint knuckles when evaluating fingers. Shared reputation is only available in Halo-ReDS, as described in Section 3.5.

Attack rate. Attack rate a is the probability with which malicious nodes decide to attack a lookup (see Section 2.2).

3. We use "network" to indicate the underlying DHT, i.e., Chord or Kad.

4. Larger networks can be simulated, but take an impractically long time to finish since each node in the network must build up reputation information through lookups.

Sampling. For some of our results, we used a *continuous simulation mode*, in which the network is sampled at regular intervals as the simulation progresses. This allowed us to monitor the evolution of the failure rate as nodes learn more information about the network and as nodes join and leave. To achieve this, we conducted alternating phases of n_t *training lookups*, during which reputation scores evolve under normal system operation, and n_l *probing lookups*, during which the reputation system is frozen and the lookup failure rate is recorded. In both phases, the attacker is attacking lookups, but reputation scores are frozen in the probing phase. Probing can, thus, be thought of as taking a snapshot of the state of the network. One set of training lookups and probing lookups is a *slot*. We then took the failure rate achieved in the steady state as the final result. Since continuous simulations show changes over time, they represent a single (often very long) simulation run.

In other (noncontinuous) simulations, we simply ran a long training phase and then a single probing phase at the end. Each data point in these graphs corresponds to an average value with standard error bars from n_i different instantiations of the DHT, where we typically set $n_i = 10$.

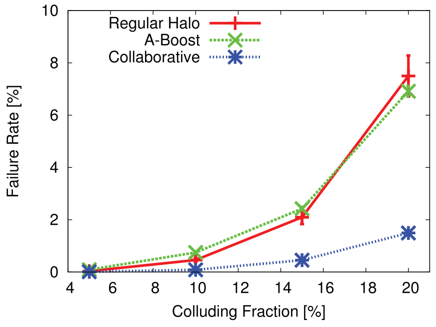
4.2 Attack Success under Churn

One issue with reputation in a DHT is that as nodes join and leave the network (i.e., in the presence of churn), reputation information becomes stale. We seek to determine whether ReDS performance reaches a reliable steady state as churn continues to affect the system.

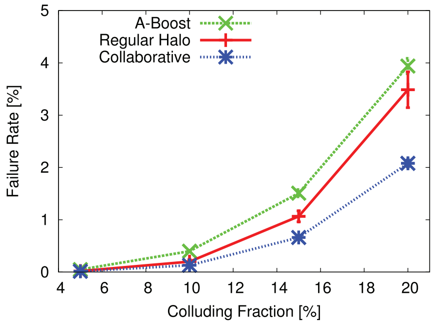
Halo-ReDS. We ran Halo-ReDS experiments in continuous simulation mode, in which all nodes were replaced on average once every 800 lookups (160 slots)—by the time a node has done 800 lookups, each node in the network has been replaced once on average. We, then, let this simulation continue for 20,000 lookups per node (4,000 slots, 20 million total lookups), and calculate the average failure rate over the latter half of the simulation (i.e., the latter 2,000 slots) to get a *steady-state* value for the failure rate (we include a graph showing the time evolution of failure rate to a steady state in Section 5 of the online supplement).

Comparison of different Halo schemes. We now compare the failure rates of regular Halo, A-Boost, and collaborative boosting for different colluding rates using noncontinuous simulations. Fig. 3a shows the failure rate for these schemes for attack rate $a = 1.0$. Due to churn, A-Boost performs roughly the same as regular Halo. Collaborative boosting, however, significantly reduces the failure rate, down to 1.5 percent for a colluding fraction of 20 percent, an improvement of 79 percent over regular Halo and 73 percent over A-Boost.

The results for an attack rate of $a = 0.5$ are shown in Fig. 3b. Note that in such a scenario there is only half as much information available to the reputation system about attackers. The failure rate of regular Halo is now approximately half of what it was for an attack rate of $a = 1.0$, because only half of all lookups are attacked. A-Boost now performs worse than regular Halo. Collaborative boosting, however, performs much better than regular Halo and A-Boost, reducing the failure rate by 40 and 50 percent compared to regular Halo and A-Boost, respectively.



(a) Attack rate $a = 1.0$. A-Boost fails to improve over regular Halo due to node churn. Collaborative boosting, however, performs significantly better throughout.



(b) Attack rate $a = 0.5$. A-Boost performs worse than regular Halo, while collaborative boosting performs significantly better than either regular Halo or A-Boost.

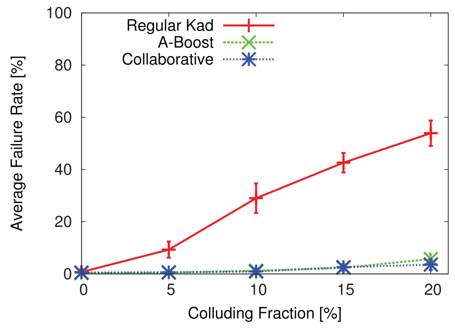
Fig. 3. *Halo.* These graphs compare the failure rates for different Halo algorithms and colluding fractions.

A-Boost suffers under churn because it adapts slowly to changes beyond its fingers. The reputation trees of nodes that did not see the change in their fingers' fingers (and further down the tree) do not account for these churn events. In collaborative boosting, however, nodes can rely on their fingers to adapt to changes further down the lookup paths. This greatly improves the speed at which lookup paths are modified to address churn events.

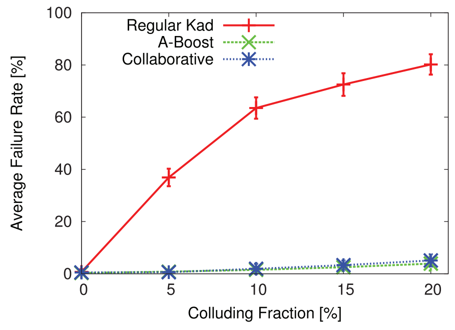
Kad-ReDS. We now compare the performance of Kad and both collaborative and A-boost versions of Kad-ReDS under churn. We present results for $r = 25\%$ churn, and $k = 10$ and $\alpha = 7$ redundancy. Fig. 4a shows the failure rate of regular Kad, A-Boost, and collaborative boosting for an attack rate of $a = 1.0$. The performance of A-Boost and collaborative are almost the same (within the margin of error), significantly reducing the failure rate of Kad down to 4-5 percent from 54 percent for $c = 20\%$.[5]

The results for an attack rate of $a = 0.0$ are shown in Fig. 4b. In our attack model (see Section 3.3), attackers pollute routing tables even when they are not attacking lookups. Fig. 4b shows the failure rate slightly increases for 0 percent attack rate compared to the failure rate for 100 percent attack rate. We further discuss attack effectiveness in Section 4.3. In all scenarios we tested, performance is improved by at least 93.4 percent with Kad-ReDS compared to regular Kad.

5. Fig. 4a can be compared with the results presented in Section 6 of the online supplement, where we see the performance of Kad-ReDS with $k = 10$ and $\alpha = 7$ degrades from 2-3 percent failure rate to 4-5 percent due to the 25 percent churn.

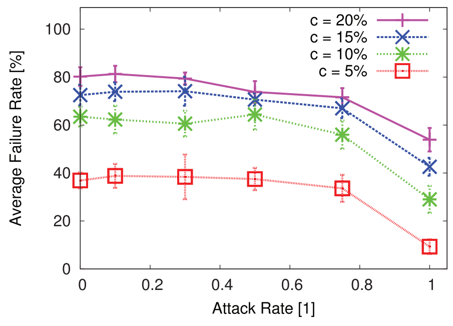


(a) Attack rate of $a = 1.0$.

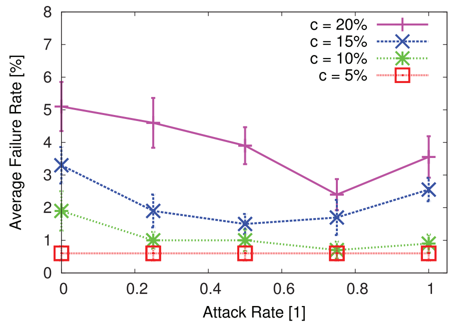


(b) Attack rate of $a = 0.0$.

Fig. 4. *Kad.* Failure rates of different Kad-ReDS schemes for different colluding fractions c. Both A-Boost and Collaborative are much more effective than Kad.



(a) Kad.



(b) Kad-ReDS. y-axis ranges from 0-8%.

Fig. 6. *Kad.* Overall attack effectiveness. Attackers perform best by attacking with low attack rates.

4.3 Overall Attack Effectiveness

In this experiment, we study collaborative boosting by measuring *overall attack effectiveness*, the maximum continuous failure rate that the attacker can achieve when his nodes use a consistent attack rate (i.e., the same in training and probing). This allows us to identify the best that the attacker could do consistently over time.

Halo-ReDS. The results for Halo and A-Boost are shown in Section 5 of the online supplement. In brief, attack effectiveness grows linearly with the attack rate in Halo, which is expected, as there is no reputation system—attacking more has no negative consequences. Attack effectiveness also grows linearly with the attack rate in A-Boost, which is unsurprising when we consider that A-Boost is about as effective as regular Halo under churn.

Fig. 5 shows the overall attack effectiveness that the attacker can achieve against collaborative boosting. We see that increasing the attack rate up to a point results in more lookup failures. Beyond a certain attack rate, for example, at 60 percent for a colluding fraction of $c = 20\%$, the overall failure rate goes back down. Comparing the overall effectiveness of collaborative boosting to A-Boost and regular Halo, ReDS reduces effective failure rates by up to 70 percent for $c = 5\%, 10\%$, and by up to 80 percent for $c = 15\%, 20\%$.

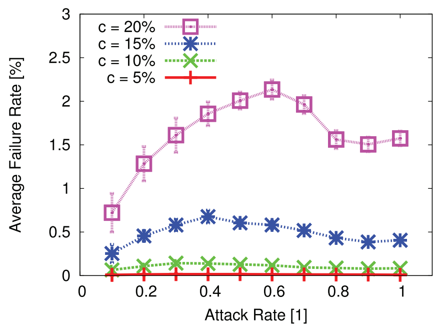


Fig. 5. *Halo-ReDS.* The failure rate for different continuous attack rates. Attacker effectiveness peaks before $a = 1.0$.

The peak in effectiveness comes from the attacker's need to balance exploiting positive reputation to subvert lookups with maintaining those positive reputation values. Despite the ability of attackers to operate at a peak rate, we note that for colluding fractions of 20 percent and below, no matter what rate the attackers attack with, *collaborative boosting limits their effectiveness to below 2.1 percent*.

Kad-ReDS. We similarly examined overall attack effectiveness in Kad. Fig. 6 shows our results. Kad has very different results (see Fig. 6a) from Halo due to routing-table pollution. In particular, for Kad with colluding fraction $c = 20\%$, the failure rate is 80 percent when the attack rate is $a = 0.0$ and drops to 54 percent when $a = 1.0$. The high failure rate with no manipulation of lookups ($a = 0.0$) is due to the effectiveness of routing-table pollution against Kad. As the attack rate increases, routing-table pollution decreases, which leads to the drop in failure rates. This effect on routing-table pollution occurs because the results returned by attacker nodes are always attacker nodes and are generally further away from the target than those returned by honest nodes when the attacker manipulates lookups. So whenever both attacker and honest nodes respond to a lookup, the attacker nodes not only fail to manipulate the lookup result but also do not appear in the later stages of the lookup process. This reduces the malicious nodes' chances of being opportunistically added to k-buckets.

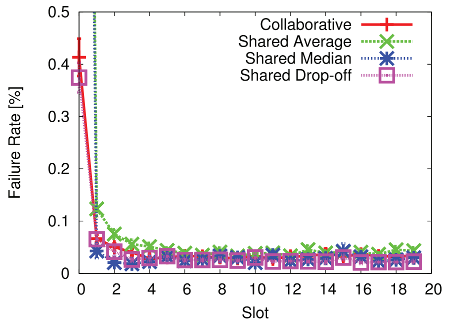


Fig. 7. Shared reputation ($c = 10\%$) at best performs about the same as normal collaborative mode.

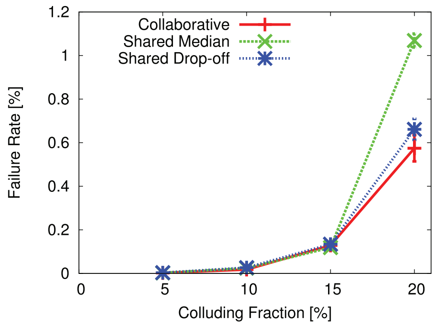


Fig. 8. Shared reputation increases failures for newly joined nodes.

In comparison, Fig. 6b shows how effective Kad-ReDS is in countering the route subversion attack described in Section 3.3. The attackers gain a slight advantage with lower attack rates. The reputation system, however, severely limits the growth of the average failure rate by curbing routing-table pollution. Kad-ReDS maintains a failure rate of no more than 5.1 percent for all attack rates with $c \leq 20\%$, which is a 93 percent improvement over Kad.

4.4 Effectiveness of Shared Reputation

In these experiments, we explored whether sharing reputation values can help lower the failure rate. We studied shared reputation in the context of Halo-ReDS and again simulated malicious nodes attacking at a rate of 1.0 for different fractions of colluding nodes to see how the failure rate evolves. We also looked at different ways of calculating shared reputation: average, median, and the Drop-Off algorithm (see Section 3.5). The values returned by malicious nodes were calculated to maximize the probability that the value was accepted by the requesting node by taking into account the shared reputation algorithm.

Fig. 7 shows the results for 10 percent colluding nodes. We see that while there is a slight difference in the convergence speed at the beginning of the simulation (median and Drop-off shared reputation converging slightly faster), the difference is not statistically significant. For a higher fraction of colluding nodes at 20 percent and for attack rates lower than $a = 1.0$ (not shown), shared reputation fails to improve the failure rate over collaborative in any scenario.

New nodes. Next, we studied whether new nodes joining the system can benefit from shared reputation. Intuitively, shared reputation should be useful for new nodes joining an existing Chord network. A new node does not have any reputation information yet, so asking other nodes in the network for shared reputation helps a node to make routing decisions until it has collected enough observations. In this experiment, we tested the failure rate of nodes newly added to the Chord network, where those nodes rely solely on shared reputation and collaborative boosting. The results do not bear out this intuition, however, Fig. 8 shows that using shared reputation for newly added nodes does not improve the failure rate and can even worsen the failure rate, for example, for colluding fractions of 20 percent. This can be explained by noting that using collaborative boosting necessarily decreases the failure rate, as each node only operates according to its firsthand observations. In shared reputation, however, nodes become susceptible to slandering and self-promotion of malicious nodes.

4.5 Kad-ReDS-Specific Experiments

In Section 6 of the online supplement, we show how routing-table pollution is severely limited by Kad-ReDS. We also show how Kad-ReDS greatly outperforms Kad for various levels of redundancy with and without collaborative boosting.

5 Security Analysis

Due to space limitations, we present our security analysis in Section 8 of the online supplement. To briefly summarize, we find that the oscillation attack is the most important threat to ReDS systems and examine three strategies that the attacker could use. Our results show that such an attacker can be effectively countered by updating scores with an exponentially weighted moving average, parameterized to heavily weight recent behavior, as well as selecting peers with the highest reputation scores instead of attempting to explore other peer choices in the k-bucket. Other attacks on firsthand reputation are limited. With shared reputation, however, we identify a new *use-based attack*, which is particularly relevant in ReDS. We show in analysis that an attacker node could attack a large fraction of lookups with little or no negative cost to its reputation. This attack is a significant reason why we recommend against using shared reputation in ReDS designs.

6 Related Work

In a paper about secure routing in peer-to-peer networks (focusing on Pastry, but generalizable to other protocols), Castro et al. [12] argue that secure routing requires secure assignment of identifiers, secure routing table maintenance, and secure message forwarding. Secure assignment of identifiers is done through the use of a certificate authority (CA), which binds identifiers to IP addresses. Solving the problem of secure routing table maintenance requires modification of the Pastry protocol to introduce additional constrained routing tables. Lastly, secure message forwarding is approached by detecting failed routes and then applying route diversity. Route diversity is achieved by forwarding multiple messages until they reach a node that has the target node for a key in its neighbor set. We argue that ReDS can be used effectively for any system designed along the lines of Castro et al.'s secure routing primitive.

Harvesf and Blough [19] describe a scheme using replica placement to improve the robustness of Chord routing. By placing several replicas of a key uniformly around the Chord network, disjoint routes to the individual replicas are created, which makes it likely that at least one search for one of the replicas will use a route with no compromised nodes. This replication approach is orthogonal to our work (although Kad too uses multiple “replica roots”). ReDS ensures that searches for each replica will succeed with higher probability, and thus, fewer replicas need to be retrieved, or fewer replicas are needed in the first place.

Mickens and Nobel propose Concilium [20], which attempts to distinguish between malicious behavior and network problems and assigns blame to nodes if they are found to subvert searches. It also depends on secure identifiers (e.g., using a CA) like the scheme by Castro et al. [12]. Concilium focuses more on diagnosis and identifying malicious nodes. It requires nodes to perform network tomography as well as propagate “Blame” messages downstream to identify malicious nodes, both of which require coordination. ReDS does not try to implicate and remove bad nodes, but simply avoids them, thereby limiting the cost of false positives and allowing for fast decisions. ReDS also does not require nodes to coordinate reputation information among themselves, reducing overhead and complexity.

Malicious attacks in DHTs can be partially addressed by using the concept of quorums. A quorum is a group of nodes that effectively acts as an atomic unit, replacing individual peers in the DHT. There are several different approaches to create and maintain quorums [21], [22], [23], [24], [25]. Young et al. [25] propose a quorum-based system that can tolerate a large fraction of malicious peers—strictly less than $1/3$-fraction of a quorum. We note, however, that if 10-20 percent of the nodes are attackers, then a substantial fraction of quorums will be controlled by attackers. ReDS can, thus, improve outcomes for quorum-based systems by applying reputation at the quorum level instead of the node level.

7 Conclusions

We presented ReDS, a reputation-based mechanism for improving the resilience of searches in deterministic and nondeterministic DHTs, such as Halo and Kad, against malicious nodes. We showed how information from failed searches can be used collaboratively to avoid malicious activity in the network. Our results improve significantly over Halo and Kad, showing that even exclusively local observations for reputation information can deliver large gains to the success rate when used collaboratively. A security analysis shows that strategic attackers are limited from improving their attacks. Finally, we analyzed the potential for shared reputation mechanisms and a novel attack against shared reputation, and find that using only firsthand observations is superior to sharing.

Acknowledgments

This material is based upon the work supported by the US National Science Foundation (NSF) under Grant Nos. CNS-1117866, CNS-1115693, and CAREER Award number CNS-0954133. The authors would also like to thank John McCurley for his editorial help.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” *Proc. SIGCOMM,* 2001.

[2] S. Ratnasamy, P. Francis, M. Handley, R.M. Karp, and S. Shenker, “A Scalable Content-Addressable Network,” *Proc. SIGCOMM,* 2001.

[3] A.I.T. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems,” *Proc. IFIP/ACM Int’l Conf. Distributed Systems Platforms (Middleware ’01),* 2001.

[4] P. Maymounkov and D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,” *Proc. First Int’l Workshop Peer-to-Peer Systems (IPTPS),* 2002.

[5] A. Kapadia and N. Triandopoulos, “Halo: High-Assurance Locate for Distributed Hash Tables,” *Proc. 15th Ann. Network and Distributed System Security Symp. (NDSS),* 2008.

[6] A. Nambiar and M. Wright, “Salsa: A Structured Approach to Large-Scale Anonymity,” *Proc. 13th ACM Conf. Computer and Comm. Security (CCS ’06),* 2006.

[7] M.S. Artigas, P.G. Lopez, J.P. Ahullo, and A.F.G. Skarmeta, “Cyclone: A Novel Design Schema for Hierarchical DHTs,” *Proc. IEEE Fifth Int’l Conf. Peer-to-Peer Computing (P2P),* 2005.

[8] A. Panchenko, S. Richter, and A. Rache, “NISAN: Network Information Service for Anonymization Networks,” *Proc. ACM Conf. Computer and Comm. Security (CCS),* 2009.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru, “A Survey of Attack and Defense Techniques for Reputation Systems,” *ACM Computing Surveys,* vol. 42, no. 1, pp. 1-31, 2009.

[10] M. Wright, A. Kapadia, M. Kumar, and A. Dhadphale, “ReDS: Reputation for Directory Services in P2P Systems,” *Proc. Sixth Ann. Cyber Security and Information Intelligence Research Workshop (CSIIRW),* 2010.

[11] R. Akavipat, A. Dhadphale, A. Kapadia, and M. Wright, “ReDS: Reputation for Directory Services in P2P Systems,” *Proc. ACM Workshop Insider Threats,* 2010.

[12] M. Castro, P. Druschel, A.J. Ganesh, A.I.T. Rowstron, and D.S. Wallach, “Secure Routing for Structured Peer-to-Peer Overlay Networks,” *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI),* 2002.

[13] G. Danezis and P. Mittal, “SybilInfer: Detecting Sybil Nodes Using Social Networks,” *Proc. Ann. Network and Distributed System Security Symp. (NDSS),* 2009.

[14] H. Yu, P.B. Gibbons, M. Kaminsky, and F. Xiao, “SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks,” *Proc. IEEE Symp. Security and Privacy,* 2008.

[15] C. Wilson, B. Boe, A. Sala, K.P. Puttaswamy, and B.Y. Zhao, “User Interactions in Social Networks and Their Implications,” *Proc. ACM Fourth European Conf. Computer Systems (EuroSys ’09),* 2009.

[16] N. Tran, J. Li, L. Subramanian, and S.S. Chow, “Optimal Sybil-Resilient Node Admission Control,” *Proc. INFOCOM,* 2011.

[17] P. Wang, I. Osipkov, N. Hopper, and Y. Kim, “Myrmic: Provably Secure and Efficient DHT Routing,” DTC Technical Report 2006/20, Univ. of Minnesota, 2006.

[18] D. Wagner, “Resilient Aggregation in Sensor Networks,” *Proc. Second ACM Workshop Security of Ad Hoc and Sensor Networks (SASN ’04),* 2004.

[19] C. Harvesf and D.M. Blough, “Replica Placement for Route Diversity in Tree-Based Routing Distributed Hash Tables,” *IEEE Trans. Dependable and Secure Computing,* vol. 8, no. 3, pp. 419-433, May 2011.

[20] J.W. Mickens and B.D. Noble, “Concilium: Collaborative Diagnosis of Broken Overlay Routes,” *Proc. IEEE/IFIP 37th Ann. Int’l Conf. Dependable Systems and Networks (DSN ’07),* pp. 225-234, 2007.

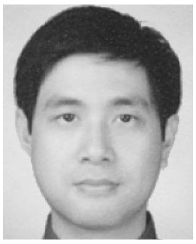
[21] M. Naor and U. Wieder, “A Simple Fault Tolerant Distributed Hash Table,” *Proc. Second Int’l Workshop Peer-to-Peer Systems (IPTPS),* 2003.

[22] A. Fiat, J. Saia, and M. Young, “Making Chord Robust to Byzantine Attacks,” *Proc. 13th Ann. European Symp. Algorithms (ESA),* 2005.

[23] J. Saia and M. Young, "Reducing Communication Costs in Robust Peer-to-Peer Networks," *Information Processing Letters*, vol. 106, no. 4, pp. 152-158, 2008.

[24] B. Awerbuch and C. Scheideler, "Towards a Scalable and Robust DHT," *Proc. ACM 18th Ann. Symp. Parallelism in Algorithms and Architectures (SPAA)*, 2006.

[25] M. Young, A. Kate, I. Goldberg, and M. Karsten, "Practical Robust Communication in DHTs Tolerating a Byzantine Adversary," *Proc. IEEE 30th Int'l Conf. Distributed Computing Systems (ICDCS)*, 2010.

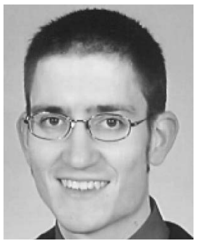


Ruj Akavipat received the PhD degree from Indiana University Bloomington. He joined Mahidol University, Thailand, as a lecturer in the Engineering Department in 2010. His research interests include distributed systems, security, mobile computing, and computer technology education.

Mahdi N. Al-Ameen received the BS degree in computer science and engineering from the Bangladesh University of Engineering and Technology in 2009. He is currently working toward the PhD degree in computer science at the University of Texas at Arlington (UTA). He is currently a graduate research assistant at the Information Security (iSec) Lab, UTA. His primary research interest includes information and cyber security. He was invited by the Springer-Verlag to publish his undergrad-thesis work on Sensor Network Topology and Fault Tolerance as a book chapter in *Advances in Wireless Sensors and Sensor Networks*.

Apu Kapadia received the PhD degree in computer science from the University of Illinois at Urbana-Champaign in October 2005. He is currently an assistant professor of computer science and informatics at the School of Informatics and Computing, Indiana University Bloomington. For his dissertation research on trustworthy communication, he received a four-year High-Performance Computer Science fellowship from the Department of Energy. Following his doctorate, he joined Dartmouth College as a post-doctoral research fellow with the Institute for Security Technology Studies, and then as a member of technical staff at MIT Lincoln Laboratory. He is interested in topics related to systems security and privacy. He is particularly interested in accountable anonymity, mobile and pervasive computing, crowdsourcing, and peer-to-peer networks. For his work on accountable anonymity, two of his papers were named as "Runners-up for PET Award 2009: Outstanding Research in Privacy Enhancing Technologies." His work on usable privacy controls was given the "Honorable Mention Award (Runner-up for Best Paper)" at the Conference on Pervasive Computing, 2007. He received the US National Science Foundation (NSF) CAREER Award in 2013. He is a member of the IEEE and the ACM.

Zahid Rahman received the BS degree in computer science and engineering from the Bangladesh University of Engineering and Technology in 2007. He is currently working toward the PhD degree in computer science at Indiana University Bloomington. He is particularly interested in privacy and security issues related to sensors in pervasive and mobile-computing systems.



Roman Schlegel received the MSc degree from EPFL in Switzerland in communication systems and the PhD degree in computer science from City University of Hong Kong. During his doctoral studies, he also spent a year as a research assistant at Indiana University Bloomington. After finishing the PhD, he joined ABB Corporate Research as a research scientist for security in industrial control systems. His research interests include privacy, network security, and applied cryptography. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Matthew Wright received the BS degree in computer science from Harvey Mudd College and the MS and PhD degrees from the Department of Computer Science, University of Massachusetts, in 2002 and May 2005, respectively. He is an associate professor at the University of Texas at Arlington. His dissertation work addresses the robustness of anonymous communications. His other interests include secure and Sybil-resistant P2P systems, security and privacy in mobile, and ubiquitous systems, and understanding the human element of security and privacy. He received the US National Science Foundation (NSF) CAREER Award and the Outstanding Paper Award at the 2002 Symposium on Network and Distributed System Security.