ReDS: A Framework for Reputation-Enhanced DHTs

Ruj Akavipat, Mahdi N. Al-Ameen, Apu Kapadia, Zahid Rahman, Roman Schlegel, 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% 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, and 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 non-existent 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% of malicious nodes in the network can subvert more than 50% 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 non-deterministic 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 non-trivial failure rate while incurring substantial overhead for redundancy. For example, Halo still has a failure rate as high as 5–6% for 20% 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 non-trivial failure rate of 17–21% with only 10% malicious nodes, even with high redundancy.

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. provide an extensive survey [9]). However, this prior work mainly addresses the

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

'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:

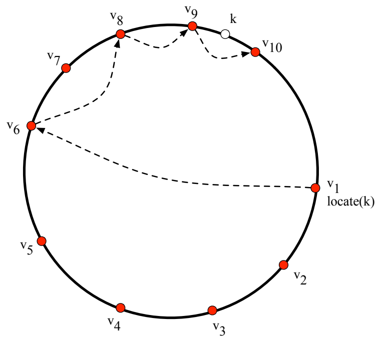
- We show that reputation can be applied to a variety of redundant DHT-based directory services to improve lookup success rates. We specifically describe *Halo-ReDS* and *Kad-ReDS*, which are implementations for Halo (a deterministic DHT) and Kad (a non-deterministic DHT).
- Building on our approach 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 first-hand observations.

2 System and Attack Models

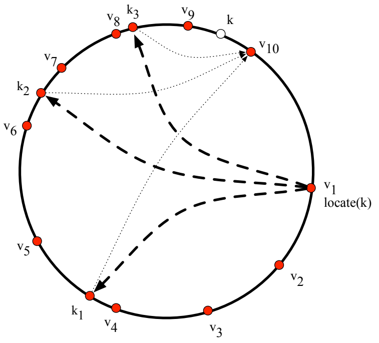
2.1 System model

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 Figure 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



(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.

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 Figure 1(a)).

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 Figure 1(b)). 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 non-deterministic routing table entries. More specifically, the j-th 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 eight bits of the key.

2.2 Attack model

Malicious nodes in the system may attempt to subvert lookup operations, e.g., by dropping or misdirecting lookups. The adversary's goal could be to cause peers to use attacker-controlled nodes, e.g., 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%) 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 $\mathsf{lookup}(t)$ 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, e.g. $ID_x = H(PU_x, N)$, 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, e.g., requiring a valid credit-card number, verifying a valid mobile-phone number by text message, or using a Sybil-resistant admission-control prototol [16]. Peers can verify the virtual addresses of nodes by checking signatures; e.g., 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 the online supplement (§6)).

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 sub-lookup 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 sub-lookups 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 sub-lookup to particular fingers. In some DHTs (e.g., Halo) this mapping is obvious because each sub-lookup proceeds independently, but in other DHTs (e.g., Kademlia) redundancy is built into the search, and greater care is needed to avoid misattribution.

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 non-deterministic 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 the online supplement (§3) — 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 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 the online supplement (§2) 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

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.

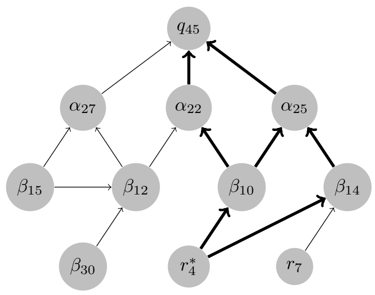


Fig. 2: Lookup graph for a lookup initiated by node q_{45} for key. Subscripts denote the XOR distance of the node from the lookup target key, i.e. $XOR(q_{45}, key) = 45$. Edges from a node are directed toward the node from which they are returned during the lookup process. Three successful paths from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

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 non-trivial.

To address this issue, we have the querying node construct a *lookup graph* (see Figure 2) that tracks the paths used to find the target owner during the lookup process. We describe the exact algorithm in the online supplement (§4). 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 in order to result in positive reputation for the malicious nodes. We explain the details of this attack and its limitations in the online supplement (§4 and §6).

3.4 Handling Churn

In our simulations (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 the online supplement (§8). 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 the online supplement (§8).

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 first-hand 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 goes into great detail on aggregation methods that are suitable for security applications, finding that median is a strong solution [18]. We note, however, that median *always* fails when the number of attackers is more than 50%. 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 the online supplement (§7.1) 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 the online supplement (§7.2).

3.6 Communication overhead

To withstand malicious behavior, various approaches have advocated the use of redundant lookups in DHTs [5]–[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% attackers but with the *square* of the typical redundancy (e.g. 169 for a redundancy of 13).

ReDS biases lookups towards 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 the online supplement (§3), 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 1000 nodes.[4] In our experiments, we use a redundancy of 10 as suggested for regular Halo with 1000 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 α=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 $n = 1000$ 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 1000 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).

Sampling. For some of our results, we used a *continuous simulation mode*, in which the network is sampled at regular

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.

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 (non-continuous) 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 $n_i = 10$.

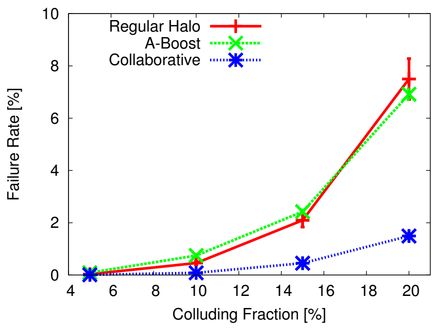
4.2 Attack success under churn

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.

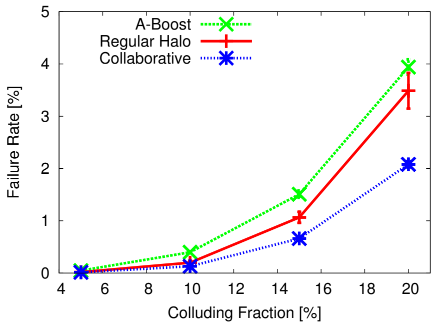
Halo-ReDS. We ran Halo-ReDS experiments in continuous simulation mode, in which all nodes were replaced on average once every 800 lookups (160 slots) — by the time a node has done 800 lookups, each node in the network has been replaced once on average. We then let this simulation continue for 20,000 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 the online supplement (§5)).

Comparison of different Halo schemes. We now compare the failure rates of regular Halo, A-Boost, and collaborative boosting for different colluding rates using non-continuous simulations. Figure 3(a) 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% for a colluding fraction of 20%, an improvement of 79% over regular Halo and 73% over A-Boost.

The results for an attack rate of $a = 0.5$ are shown in Figure 3(b). 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% 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.

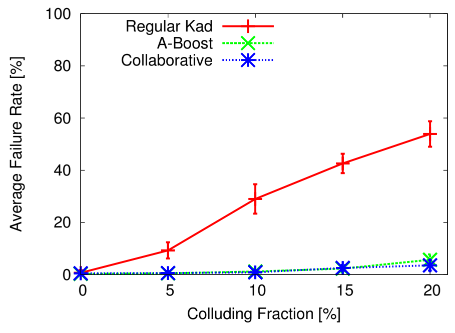
Fig. 3: **Halo.** These graphs compare the failure rates for different Halo algorithms and colluding fractions.

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.

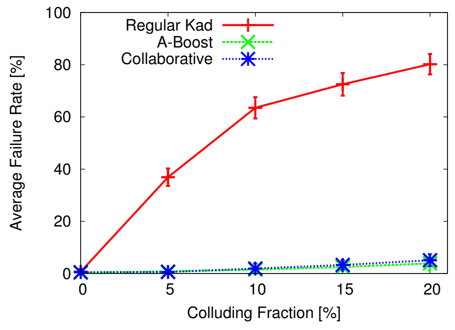
Kad-ReDS. We now compare the performance of Kad and both collaborative and A-boost versions of Kad-ReDS under churn. We present results for $r = 25\%$ churn, and $k = 10$ and $\alpha = 7$ redundancy. Figure 4(a) shows the failure rate of regular Kad, A-Boost, and collaborative boosting for an attack 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% from 54% for $c = 20\%$.[5]

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

5. Figure 4(a) can be compared with the results presented in the online supplement (§6), where we see the performance of Kad-ReDS with k=10 and α=7 degrades from 2-3% failure rate to 4-5% due to the 25% churn.

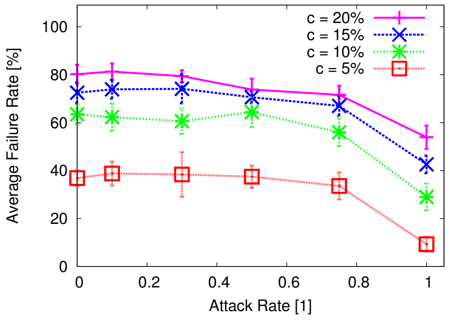


(a) Attack rate of $a = 1.0$.

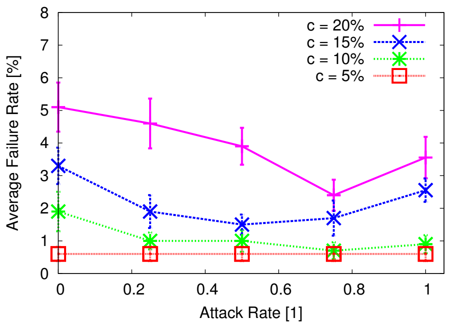


(b) Attack rate of $a = 0.0$.

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.

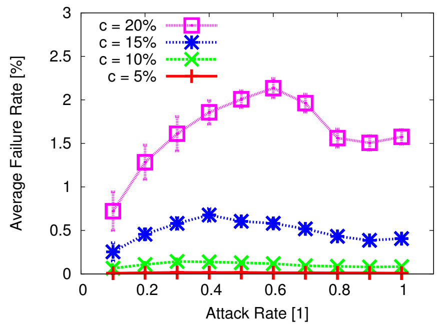


Fig. 5: **Halo-ReDS.** The failure rate for different continuous attack rates. Attacker effectiveness peaks before $a = 1.0$.

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 the online supplement (§5). 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.

Figure 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, e.g., at 60% 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% for $c = 5\%, 10\%$, and by up to 80% for $c = 15\%, 20\%$.

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% and below, no matter what rate the attackers attack with, *collaborative boosting limits their effectiveness to below 2.1%.*

Kad-ReDS. We similarly examined overall attack effectiveness in Kad. Figure 6 shows our results. Kad has very different results (Figure 6(a)) from Halo due to routing-table pollution. In particular, for Kad with colluding fraction $c = 20\%$, the failure rate is 80% when the attack rate is $a = 0.0$ and drops to 54% 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

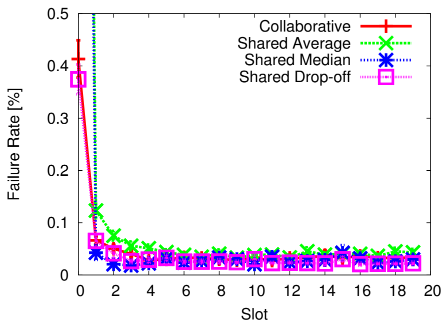


Fig. 7: Shared reputation ($c = 10\%$) at best performs about the same as normal collaborative mode.

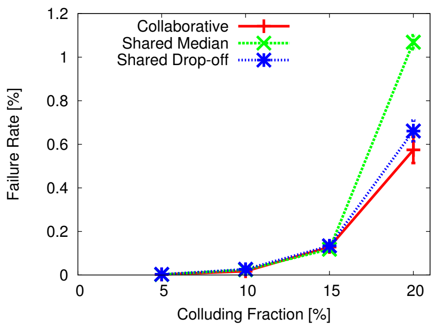


Fig. 8: Shared reputation increases failures for newly joined nodes.

lookup process. This reduces the malicious nodes' chances of being opportunistically added to k-buckets.

In comparison, Figure 6(b) 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% for all attack rates with $c \leq 20\%$, which is a 93% 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.

Figure 7 shows the results for 10% 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% 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. Figure 8 shows that using shared reputation for newly added nodes does not improve the failure rate and can even worsen the failure rate, e.g., for colluding fractions of 20%. This can be explained by noting that using collaborative boosting necessarily decreases the failure rate, as each node only operates according to its first-hand observations. In shared reputation, however, nodes become susceptible to slandering and self-promotion of malicious nodes.

4.5 Kad-ReDS-specific experiments

In the online supplement (§6) 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 the online supplement (§8). 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 moving weighted average (EWMA), 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 first-hand 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]–[25]. Young et al. propose a quorum-based system [25] 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% 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 non-deterministic 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 first-hand observations is superior to sharing.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1117866, CNS-1115693 and CAREER award number CNS-0954133. We also 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," in *SIGCOMM*, 2001.

[2] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM*, 2001.

[3] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware*, 2001.

[4] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS*, 2002.

[5] A. Kapadia and N. Triandopoulos, "Halo: High-assurance locate for distributed hash tables," in *NDSS*, 2008.

[6] A. Nambiar and M. Wright, "Salsa: A structured approach to large-scale anonymity," in *CCS*, 2006.

[7] M. S. Artigas, P. G. Lopez, J. P. Ahullo, and A. F. G. Skarmeta, "Cyclone: A novel design schema for hierarchical DHTs," in *P2P*, 2005.

[8] A. Panchenko, S. Richter, and A. Rache, "NISAN: Network information service for anonymization networks," in *CCS*, 2009.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Comput. Surv.*, 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," in *CSIIRW*, 2010.

[11] R. Akavipat, A. Dhadphale, A. Kapadia, and M. Wright, "ReDS: Reputation for directory services in P2P systems," in *WIT*, 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," in *OSDI*, 2002.

[13] G. Danezis and P. Mittal, "SybilInfer: Detecting Sybil nodes using social networks," in *NDSS*, 2009.

[14] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "SybilLimit: A near-optimal social network defense against sybil attacks," in *IEEE S&P*, 2008.

[15] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Eurosys*, 2009.

[16] N. Tran, J. Li, L. Subramanian, and S. S. Chow, "Optimal sybil-resilient node admission control," in *INFOCOM*, 2011.

[17] P. Wang, I. Osipkov, N. Hopper, and Y. Kim, "Myrmic: Provably secure and efficient DHT routing," University of Minnesota DTC Research Report, Tech. Rep. 2006/20, 2006.

[18] D. Wagner, "Resilient aggregation in sensor networks," in *ACM SASN*, 2004.

[19] C. Harvesf and D. M. Blough, "Replica placement for route diversity in tree-based routing distributed hash tables," *IEEE Trans. Dependable Sec. Comput.*, vol. 8, no. 3, pp. 419–433, 2011.

[20] J. W. Mickens and B. D. Noble, "Concilium: Collaborative diagnosis of broken overlay routes," in *DSN*, 2007, pp. 225–234.

[21] M. Naor and U. Wieder, "A simple fault tolerant distributed hash table," in *IPTPS*, 2003.

[22] A. Fiat, J. Saia, and M. Young, "Making Chord robust to byzantine attacks," in *European Symposium on Algorithms (ESA)*, 2005.

[23] J. Saia and M. Young, "Reducing communication costs in robust peer-to-peer networks," *Information Processing Letters*, vol. 106 (4), pp. 152–158, 2008.

[24] B. Awerbuch and C. Scheideler, "Towards a scalable and robust DHT," in *ACM SPAA*, 2006.

[25] M. Young, A. Kate, I. Goldberg, and M. Karsten, "Practical robust communication in DHTs tolerating a Byzantine adversary," in *ICDCS*, 2010.

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

Mahdi N. Al-Ameen is currently a PhD student in Computer Science at the University of Texas at Arlington (UTA). His primary research interest is Information and Cyber Security and is currently Graduate Research Assistant at the iSec (Information Security) Lab, UTA. He earned his B.S. in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in 2009. 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*.

Dr. Apu Kapadia is an Assistant Professor of Computer Science and Informatics at the School of Informatics and Computing, Indiana University Bloomington. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign (UIUC) in October 2005. 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 (ISTS), and then as a Member of Technical Staff at MIT Lincoln Laboratory.

Apu Kapadia 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. Apu Kapadia received the NSF CAREER award in 2013. He is a member of IEEE and ACM.

Zahid Rahman is currently a PhD student in Computer Science at Indiana University Bloomington. He earned his BS degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET) in 2007. He is particularly interested in privacy and security issues related to sensors in pervasive and mobile-computing systems.

Dr. Roman Schlegel has an MSc from EPFL in Switzerland in Communication Systems and a PhD in Computer Science from City University in Hong Kong. During his doctoral studies he also spent a year as a research assistant at Indiana University Bloomington in the US. After finishing his 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 also a member of the IEEE, the IEEE Computer Society and the ACM.

Dr. Matthew Wright is an associate professor at the University of Texas at Arlington. He graduated with his Ph.D from the Department of Computer Science at the University of Massachusetts in May, 2005, where he earned his M.S. in 2002. 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. Previously, he earned his B.S. degree in Computer Science at Harvey Mudd College. He is a recipient of the NSF CAREER Award and the Outstanding Paper Award at the 2002 Symposium on Network and Distributed System Security.

ReDS: A Framework for Reputation-Enhanced DHTs (Supplementary Material)

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

1 Supplemental Contents

In this supplemental document we provide detailed analysis and investigation of ReDS. In particular, we include the following contributions:

- Section 2 provides a detailed description of A-Boost, the local mode of Halo-ReDS that we compare the collaborative mode against in our simulation experiments.
- Section 3 briefly analyzes the effect of the collaborative mode on path length in Halo-ReDS.
- Section 4 provides a detailed description of the Kad-ReDS algorithm.
- Section 5 shows the evolution of failure rate for Halo over time, illustrating how the failure rate achieves a steady-state value.
- Section 6 shows results specific to Kad-ReDS, including routing table pollution and parameter selection.
- Section 7 describes a novel mechanism for computing shared reputation scores in the ReDS setting and our analysis of this scheme.
- Section 8 is our complete security analysis, including a detailed investigation of oscillation attacks and analysis of an attack on shared reputation in the ReDS setting.

2 Algorithm for Halo-ReDS with A-Boost (local boosting)

In this section we provide an overview of A-Boost. This material was also described in an earlier workshop paper [1].

2.1 Local vs. adaptive boosting

In this section we discuss how a node initiating a lookup process can 'boost' its chances of success by picking the best first hop in the route based solely on local observations.

The first intuition behind ReDS is that robust DHT systems use redundant lookups that can be used to distinguish good lookups from bad ones. In particular, such systems rely on the fact that, among the IDs returned from a redundant lookup, the ID closest to the target is a more accurate response than any other returned ID.[1] We present three possible ways that a requesting peer can use this information (the first two are instructive to help understand the third technique, which is what we use for ReDS):

1-Boost: The requesting peer can mark all the helper peers (we call the first-hop nodes selected for a redundant search "helpers", which are selected from the fingers of the originating node) who provide the closest ID as slightly more reliable than the helper peers who provide inaccurate responses. After a number of lookups, the most reliable helper peers will be found and used. In other words, the requesting peer gains confidence that the *first* hop in a lookup is honest. We call this approach *1-Boost* because the success rate of a lookup is boosted with one honest node. The major drawback of 1-Boost is that the reliability of a helper peer depends not only on the helper peer itself but on all of the fingers it uses along the various lookup paths. *Thus, a perfectly reliable and honest helper peer may be marked as unreliable, even when it can provide useful lookups along some paths.*

2-Boost: For each helper peer the requesting peer maintains a score for each corresponding entry in the helper peer's routing table. Depending on the lookup key, the requesting peer can estimate which finger was used by the helping peer and score that finger. On subsequent lookups, the requesting peer can pick the helpers that maximize the chances of a successful lookup through reputable fingers. We call this approach *2-Boost* because it aims to choose paths where the first two nodes in the path are good.

Adaptive-Boost: The final step is to generalize 1-Boost and 2-Boost to the entire lookup path. The requesting peer can estimate the path taken to any possible target region and maintains reputation scores for all nodes in the DHT based on whether or not lookups through those nodes succeeded. But this would result in a large data structure and would take a huge number of lookups to obtain enough information about all possible paths. Adaptive-Boost (*A-Boost*) therefore estimates the reputation of nodes as far along the path as possible, as long as there are enough

1. We note the requesting peer must verify the result by performing a handshake with the peer owning the ID, to ensure the node exists.

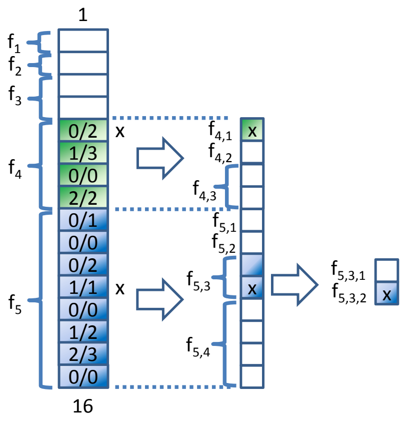


Fig. 1: Example reputation tree. $f_{i,...}$ denote fingers covering address chunks (shown as boxes). "x" marks an example of the lookup targets and highlighted boxes are chunks that would be combined if the number of observations in the smaller combination (shown in projections to the right) were insufficient. The number (v/y) in each box is the ratio of total recorded successes (v) to total recorded lookups (y) for that chunk.

observations at that depth. We define a parameter γ as the minimum number of observations required for the given depth to be used in A-Boost.

A-Boost can effectively 'shorten' the lookup path with respect to lookup reliability. Intuitively, as more lookups are done it is increasingly likely that a future lookup will share more of the path with prior lookups. If prior lookups are a good predictor of future performance, then this allows for the identification of reliable sub-paths or prefixes (the latter part of the path remains unpredictable). The requesting peer can then select helper nodes so as to use those reliable sub-paths (A-Boost) more often than unknown or poorly performing sub-paths.

2.2 A-Boost reputation tree

To model how ReDS with A-Boost operates, we use a *reputation tree*. For each helper node, the requesting node stores its scores in a tree that approximates the paths used in lookups by that helper node. The ID space is divided into contiguous regions called *chunks* according to the nodes that will be on the path for any lookup into that chunk.

An example of how the chunks are aligned with the fingers (f_i) of a finger f is shown in Figure 1—each cell represents one chunk. For example, searches into the first chunk will go through finger f_1, and searches into Chunks 5–8 will go through finger f_4. Because we are using 2^m chunks (for some integer m), we can further align the ID space accurately down the chain of fingers by recursively splitting up these ranges. For example, the second column shows the fingers and chunks for f_5 as $f_{5,1}, f_{5,2}, f_{5,3}, f_{5,4}$. The figure also shows the fingers for $f_{5,3}$ as $f_{5,3,1}, f_{5,3,2}$. A lookup from f to the chunk marked with the second 'x' is expected to traverse the subpath $f, f_5, f_{5,3}, f_{5,3,2}$. If a large number of chunks is used, longer subpaths can be estimated. We note that since fingers may not line up exactly to chunk boundaries in the ID space, the mapping to the reputation tree is approximate. As the number of nodes increases, however, the number of boundary cases will be negligible.

The reputation score for helper node f for a particular target chunk is simply the number of successful lookups divided by the number of attempted lookups at the lowest level in the reputation tree with enough data, i.e. with at least γ observations. For example, using the chunk table shown in Figure 1, if $\gamma = 5$ and the lookup target is in Chunk 12, then the total number of observations ($= 1 < \gamma$) is not enough. The algorithm then steps back one level from $f_{5,3,2}$ to $f_{5,3}$. The lookup records in Chunks 11 and 12 will be combined, since they are covered by $f_{5,3}$, which yields three records — still less than γ. To get more observations Chunks 9 to 16 will be combined as they are covered by f_5 which is a parent of $f_{5,3}$. At this point enough observations (= 9) are obtained. The algorithm then produces $4/9 = 0.44$ as a reputation value for this finger. In the case that the total observations from all chunks is still less than γ, the reputation value of the finger is set to 0.5.

Thus the lookup success rate for the lowest relevant subtree for which the helper peer satisfies the γ threshold can be used as the total reputation score for the helper node for that lookup. Then the R helper nodes with the highest scores are selected for the redundant lookup. Note that Halo's redundant lookups go to the various knuckles of t, and thus each lookup has a different target. Therefore, during scoring the reputation tree for a helper (finger) node is updated based on the specific target for each lookup. For each lookup in a redundant search for target t, a finger with the highest reputation value for the target of the lookup, which can be t or one of t's knuckles, is picked. Also, while selecting fingers for the redundant lookups, once a finger is selected (based on the A-boost score) it is removed from the pool of that redundant search. The process is repeated until R fingers are selected for the R lookups.

3 Halo-ReDS Model and Analysis

In a Chord lookup the lookup locates the predecessor first and asks it to return the successor (the target node). Thus, a malicious predecessor can still subvert a lookup for its successor, because it controls *all* lookups for that successor. To alleviate this problem we assume that each node knows k' additional successors ($k' + 1$ in total), so that the last hop is short-circuited as long as the lookup reaches the k'-vicinity of the target. This adds a modest overhead of storing and updating k' nodes for each peer's routing table. As shown below, k' is a constant that can be kept low.

A potential issue with this approach to building k-buckets is that the predecessors of a finger are not as close to the target at each hop, thereby increasing the lookup cost. Since each hop may regress by at most k nodes at each hop, the average number of nodes between the current hop and the target node after hop i is at most $n/2^i - k/2^{i-1} + 2k$. Therefore, assuming $k < \log n$ (which we can ensure as a system parameter), after $O(\log n)$ hops there are at

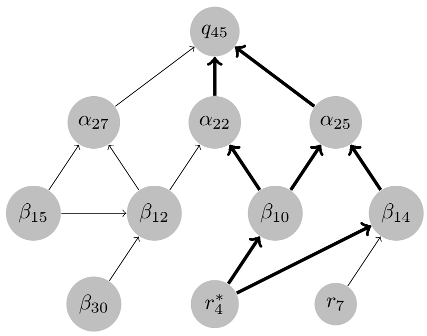


Fig. 2: Lookup graph for a lookup initiated by node q_{45} for key. Subscripts denote the XOR distance of the node from the lookup target key, i.e. $XOR(q_{45}, key) = 45$. Edges from a node are directed toward the node from which they are returned during the lookup process. Three successful paths from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

most $2k + 1$ nodes between the target and the current hop. Note that we short-circuit lookups to the k'-vicinity of the target. Thus, by setting $k' \geq 2k + 1$, the system administrator can ensure that the use of predecessors will cost at most one additional hop compared to regular Chord. In our simulations we used $k = 2$ and $k' = 8$ for a 1,000-node network and found that path lengths for Halo-ReDS increased by 0.15 hops on average compared to regular Halo.

4 KAD-REDS ALGORITHM

Kad-ReDS maintains a *lookup graph* that tracks the paths used to find the target owner during the lookup process. The querying node starts with itself as a vertex, as shown in Figure 2. For each of the β results, a directed edge is constructed to the intermediate queried node returning that result. We note that duplicate nodes can be returned by different queried nodes during a lookup step; e.g., β_{10} in Figure 2 is returned by two different queried nodes α_{22} and α_{25}. Cycles are also possible, as a node may return an ancestor. The querying node incrementally builds such a graph by using Algorithm 1 at each step of the lookup. For example, in the next iteration the querying node q_{45} selects the closest α nodes β_{10}, β_{12}, and β_{14} to expand next.

After the search terminates with one or more replica roots being identified, we mark each lookup path as being successful if it terminated in finding the *closest* replica root to the key. This is done by traversing the lookup graph in depth-first search order as described in Algorithm 2, with the closest replica root as the starting node u. Algorithm 2 avoids cycles by keeping track of nodes visited during the traversal. Each finger appearing on a successful path is then credited +1 to its reputation score, since it was involved in locating the closest correct replica root. Other nodes in the graph are not credited. For example, in Figure 2 reputation scores of the contacts β_{10} and α_{25} in the k-bucket of q_{45} are increased by 2, as two successful paths go through these nodes. Our rationale for locating the closest replica root is that it makes Kad-ReDS robust against attackers who may attempt to insert a replica root close to the target key.

Algorithm 1: Building the lookup graph for Kad-ReDS

```
input : $q$ : Querying node
        $i$ : Current lookup step
        $\alpha_i$ : a queried node at step $i$
        $N_\alpha(1:\beta)$ : $\beta$ result nodes returned by $\alpha_i$
        $G_{i-1}\langle V, E\rangle$ : Lookup graph at step $(i-1)$
output: $G_i\langle V, E\rangle$ at step $i$
if $i$ = 0 then
    Add vertices $q$ and $\alpha_i$ in $V(G_{i-1})$;
    Add a directed edge $e = (\alpha_i, q)$, from $\alpha_i$ to $q$ in $E(G_{i-1})$;
end
for each node $\beta_j$ in $N_\alpha$ do
    if $\beta_j$ is not a vertex in $G_{i-1}$ then
        Add vertex $\beta_j$ in $V(G_{i-1})$;
    end
    Add a directed edge $e = (\beta_i, \alpha_i)$ from $\beta_i$ to $\alpha_i$ in $E(G_{i-1})$;
end
```

Algorithm 2: UPDATING REPUTATION SCORE FOR KAD-REDS

```
UpdateReputation() input : $q$ : Querying node
        $G\langle V, E\rangle$ : Final lookup graph
        $u$ : A vertex in the graph $G\langle V, E\rangle$
        visited : List of visited nodes
output: Updated reputation scores of $q$'s $k$-bucket contacts
if $u$ is in visited list then
    return;
end
if $u$ = $q$ then
    return;
end
increment by 1 the reputation score of $k$-bucket contact $u$ of querying node $q$;
mark node $u$ as visited;
for each node $v$ with an edge from $u$ to $v$ in $G$ do
    UpdateReputation($q$, $G$, $v$);
end
```

By crediting nodes for locating the closest replica root, the attacker is forced to insert itself at a location closer than any other replica root, which is significantly harder.

During the lookup process, the querying node uses its local reputation information to pick the best α nodes from the appropriate k-bucket. Each of the queried nodes in turn collaborates by providing the best β contacts from the appropriate k-bucket using their reputation scores from first-hand observations. Thus, at each step the basic Kad algorithm of cutting the remaining distance in half is honored, except that a better choice is made while picking nodes from within the k-buckets. We also modified the bucket eviction policy, in which regular Kad replaces the least-seen node when a new node is being added to an already-full k-bucket. Kad-ReDS instead replaces the node with the lowest reputation score, breaking ties by replacing the least seen of the least reputed nodes.

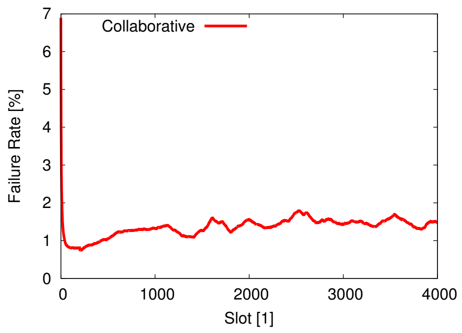


Fig. 3: **Halo (Continuous).** The evolution of the failure rate of Halo-ReDS with $c = 20\%$. The failure rate appears to stabilize around a steady-state value.

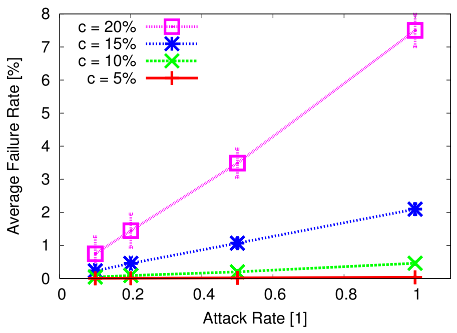


Fig. 4: The overall attack effectiveness in Halo increases linearly with attack rate a.

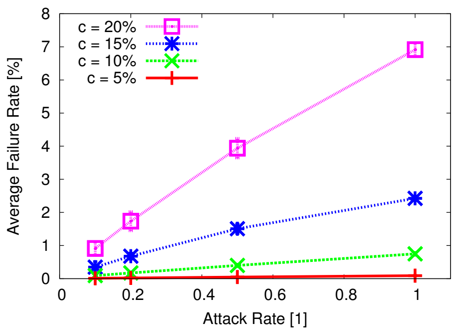


Fig. 5: The overall attack effectiveness for A-Boost also grows linearly with a.

Attack model under Kad-ReDS. The k-bucket population can change dynamically in Kad as nodes are encountered during the lookup process. To model the most effective attack possible, we make the following change to the attack model for Kad. When the attacker chooses not to attack a query, he attempts to provide the correct closest replica root at the end of the lookup to maximize his reputation scores. Additionally, he attempts to pollute routing tables with malicious nodes during the lookup process. In particular, a malicious node knows about all of the other malicious nodes and can provide them as answers to a query. He does so only as long as they are at least one bit closer in the XOR distance to the target, since more distant malicious nodes will be ignored. This approach allows the attacker nodes to be seen and possibly selected for being added to k-buckets more than would be expected. At the same time, malicious nodes can still provide a correct lookup result upon termination of the lookup process, ensuring a positive effect on reputation.

5 Halo-Specific Experiments

Halo Continuous Simulation. Figure 3 shows the results for $c = 20\%$ and attack rate $a = 1.0$. The failure rate quickly drops from a relatively high rate of 7% early on, when no reputation information is available, to less than 1%. As churn sets in the failure rate increases until it reaches a steady state of around 1.5%, indicating that the effect of node churn on ReDS is limited.

Overall attack effectiveness of Halo. As expected, if only a fraction a of lookups is attacked, then the average failure rate is a times the failure rate when the attack rate is 1 (Figure 4).

Overall attack effectiveness of A-Boost. In A-Boost, which is generally about as effective as regular Halo under churn, the results were quite similar (Figure 5). For $a = 0.1$ attack rate both systems had less than a 1% failure rate for c =5% to 20%. For $a = 0.5$ and c =20%, the failure rate was between 3.5% to 4.0%, and for $a = 1.0$ and c =20%, the failure rate was between 6.9% to 7.5%.

6 Kad-ReDS-specific experiments

We now turn towards evaluating our design for Kad-ReDS. Our primary concerns for Kad-ReDS are whether routing table pollution can be overcome, the general effectiveness of the design under different redundancy parameters, and the performance of A-Boost and collaborative boosting under churn.

Routing Table Pollution. We find that routing table pollution is the most critical factor in Kad and Kad-ReDS performance. Thus, we first seek to understand the extent of routing table pollution in these systems. To this end, we perform a continuous time simulation with 10,000 nodes under $r = 25\%$ churn. Each of the nodes performs 100 lookups in order to populate the k-buckets and build the reputation system. We divide the simulation time into 400 slots of 2500 lookups each.

As discussed in Section 4, attacker nodes attempt to get their own nodes into as many routing tables as possible. In Kad-ReDS, however, attacker nodes with low reputation scores will have little chance of being selected as the next-hop contacts in future lookups and will eventually be kicked out of many k-buckets. Figure 6 shows the pollution of routing tables over the training period for both regular Kad and Kad-ReDS. We see that the reputation system is very effective, leading to a decreasing rate of pollution of routing tables with Kad-ReDS, compared with increasing pollution rates in Kad.

Redundancy. We also explore the performance of Kad-ReDS in extensive, non-continuous simulations. First, we break down the performance in detail without churn. Fig-

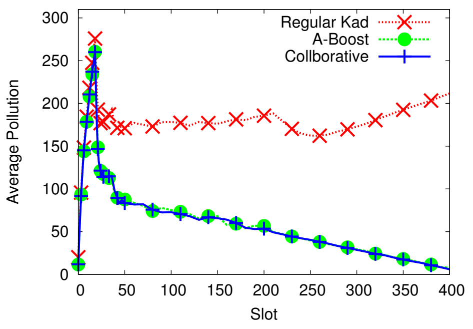
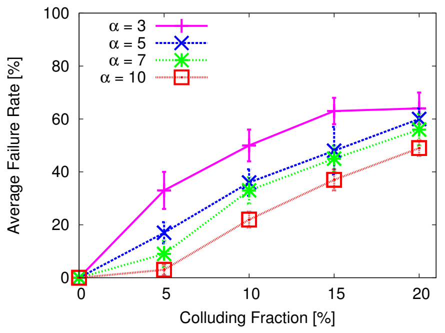
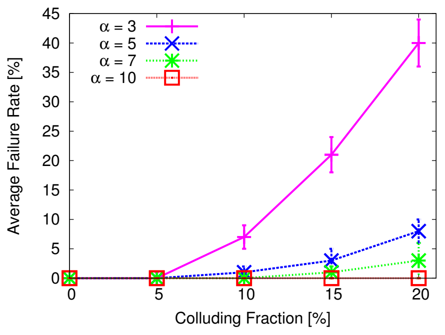


Fig. 6: **Kad (Continuous).** Evolution of routing table pollution over the lifetime of an experiment with $c = 20\%$ and $a = 1.0$. Regular Kad faces increasing pollution over time, while Kad-ReDS identifies and removes malicious fingers from k-buckets.



(a) Kad.



(b) Kad-ReDS, collaborative boosting. Note that the y-axis is from 0-45%

Fig. 7: **Kad (No Churn).** Failure rates with $k = 10$, $a = 1.0$, and varying α and c. Higher redundancy (α) improves both systems, but Kad-ReDS is much better than Kad for $\alpha > 3$.

ure 7 shows the effect of redundancy on regular Kad without churn. We use an attack rate of $a = 1.0$. Figure 7(a) shows how the failure rate decreases as the redundancy increases. However, even with a high redundancy of $\alpha = 10$, the failure rate when $c = 10\%$ is over 21%. Collaborative boosting dramatically improves the system. Figure 7(b) shows failure rates for Kad-ReDS. With a lower redundancy of $k = 10$ and $\alpha = 5$, the failure rate when $c = 10\%$ is less than 1%. When $c = 20\%$, $k = 10$, and $\alpha = 7$, Kad has a 56% failure rate, while Kad-ReDS has a 3% failure rate, a 95% decrease.

7 Shared Reputation

In this section we investigate shared reputation in ReDS in detail. First, we propose the Drop-Off scheme for calculating reputation scores. We then show that this scheme is more effective than median in our setting.

7.1 Drop-off

Prior work has investigated robust statistics for aggregating values from possibly untrusted peers [2], [3]. For example, the median of the scores is known to be more robust than the average, since one extreme value that would greatly impact the average merely shifts the median up to the next value. In Drop-off we aim to provide a more robust aggregator for reputation that is effective against both slandering and self-promotion.

We now 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 key assumption in this approach is that the score from first-hand observations is a better approximation of the correct score than scores from slandering or self-promoting attackers. We aim to balance between accepting (and hopefully gaining from others' reputation information, which may be different from our own) while trying to limit vulnerability to slandering and self-promotion attacks.

Let $r_k(f)$ be the first-hand reputation score of finger f, as measured by knuckle k. k receives a reputation score for f from joint knuckle j, which is $r_j(f)$. k then calculates $w = 1 - |r_j(f) - r_k(f)|$ and places $r_j(f)$ into a *scoring bin* for f with probability w. Intuitively, the further j's score is from k's the less likely it is to be included in the scoring bin. k's shared reputation score for j for the current epoch is the median of the scores in the scoring bin.

We note that Drop-off does not consider historical information, such as consistency with other peers. The reasons for this are: (1) the attacker can manipulate such an approach by acting differently for different peers and thereby create confusion; and (2) attackers otherwise have no incentive to oscillate in their shared reputation scores. To the latter point, note that attackers attempting to slander (or self-promote) will gain nothing by suddenly switching to sharing a high (or low) score instead.

7.2 Attacks on shared reputation

We now evaluate Drop-off against slandering and self-promotion attacks, the most prominent attacks against a shared reputation system. The goal of these attacks in our setting is to make a targeted peer select malicious fingers for lookup operations.

The key assumption in this approach is that the score from first-hand observations eventually converges to the correct score for that node and is typically a reasonable approximation of the correct score, at least relative to scores from slandering or self-promoting attackers.

We now derive an equation to estimate the expected score that the Drop-off method would provide. Let k be the knuckle of a finger f and let $r_k(f)$ be k's reputation score for finger f based only on first-hand observations. For both slandering and self promotion attacks, let us assume that $r_m(f)$ is the reputation score of f as received from the malicious knuckles and $r_h(f)$ is the reputation scores of f as received from honest knuckles. Let n_h be the number of k's joint knuckles of f that are honest and n_m be the number that are malicious. Finally, let $d_h = |r_h(f) - r_k(f)|$ and $d_m = |r_m(f) - r_k(f)|$ be the differences between k's score and the scores from its honest and malicious joint knuckles, respectively. Based on the Drop-off algorithm, $(1 - d_h)$ is the probability of selecting the score from an honest knuckle to calculate median. Letting p be the probability that the number of honest nodes selected is more than the number malicious nodes selected, we have:

$$p = \sum_{i=1}^{n_h} \sum_{j=0}^{i-1} \left(\binom{n_h}{i} (1 - d_h)^i d_h^{n_h - i} \binom{n_m}{j} (1 - d_m)^j d_m^{n_m - j} \right)$$

Let q be the probability that the number of malicious nodes and honest nodes selected are the same, meaning that the median will be calculated as $\frac{r_h(f)+r_m(f)}{2}$. Assuming that at least one honest node and malicious node are selected, we get:

$$q = \sum_{i=1}^{n'} \left(\binom{n_m}{i} (1 - d_m)^i d_m^{n_m - i} \binom{n_h}{i} (1 - d_h)^i d_h^{n_h - i} \right)$$

Here, $n' = n_m$ when $n_m \leq n_h$ and $n' = n_h$ when $n_m > n_h$.

In total, the expected Drop-off score $E[s_\delta]$ is given by:

$$E[s_\delta] = p \times r_h(f) + q \times \frac{r_h(f) + r_m(f)}{2} + (1 - p - q) \times r_m(f)$$

To illustrate the effect of the Drop-off approach, let us consider the following simple numerical example of a self-promotion attack against a knuckle K. Suppose that for calculating the score of a malicious finger F, K finds 11 joint knuckles, of which six are attackers. Let us say that the 'true' reputation score for F is 0.1 (only 10% of the searches through it will succeed), while K's estimated score from first-hand observations is currently 0.3.

Assume for simplicity that all six attackers claim that their reputation score for F is 1.0, while all five honest nodes report a score of 0.1 for F. The average score is 0.59, which is much higher than the true score. The median has reached the *breakdown point*, since more than half of the nodes are malicious; the median score becomes 1.0. For Drop-off we first must examine the population of the bucket. The expected number of honest nodes in the bucket is four, while the expected number of malicious nodes is 1.8. The actual population may vary, but for any combination of nodes in the bucket such that the number of honest nodes is more than the number of malicious nodes, the Drop-off score will be 0.1. For this example, that represents approximately 88% of the cases. In another 8.4% of the cases, there are equal numbers of honest and malicious nodes, in which case the score is 0.55. In total, the expected Drop-off score is 0.17.

The system works similarly against slandering attacks. With this approach the Drop-off system provides much better scores than those provided by taking the average. It also provides a way to avoid the breakdown point that the median faces against a majority of attackers as joint knuckles.

8 Security Analysis

In this section we present an analysis of the security of ReDS against various attacks. Since exploring these possibilities by fixing one or a few parameters in simulation would be tedious and time consuming, we analyze these situations theoretically. We begin by discussing a range of attacks and conclude that only oscillation attacks and targeted attacks on keys are serious threats to ReDS. Thus, we carefully analyze oscillation attacks and show how to limit their effectiveness. Targeted attacks are a further challenge and we plan to address them in future work. We also examine a novel attack against shared reputation in ReDS.

8.1 Attacks on first-hand observations

In ReDS a node maintains its own reputation tree for each node in its k-buckets. Other than an oscillation attack, there are several ways an attacker might try to manipulate first-hand observations: whitewashing, bootstrapping, and targeted attacks. We now discuss each in turn.

Whitewashing Attacks. In a whitewashing attack, a node leaves and rejoins the system to get a better reputation score. This attack can be partially mitigated by having nodes cache reputation values for nodes that have left, up to a memory limit. In DHTs with a k-bucket, we expect each peer to keep $O(k \log n)$ nodes in memory for routing. When a node leaves it should be removed from the k-bucket, but a tuple containing its certificate and reputation score (just a single overall value, not the full reputation tree) can be kept in a least-recently-seen queue. If the node returns using the same public key and ID, then it can be added back into a k-bucket with its old reputation score as its starting top-level score. A certificate, such as in X.509 format, can fit in 2 KB (less with a more compact format and ECC). Thus, we can easily store the 1000 most recently seen nodes in just a few MB of memory. Although an attacker could attempt to cycle that many identities through a given node's cache, each one would have to be accepted into the node's k-buckets, making the attack slow at best. In deterministic DHTs like Halo, the attacker nodes would have be the correct finger or finger's predecessor positions to be accepted at all. Thus, the attacker could not cycle many of its nodes through the cache, and its poor reputation scores would remain in memory for a long time, depending on the exact rate of churn.

In non-deterministic DHTs like Kad, the attacker can attempt to join the k-buckets of a new set of peers.

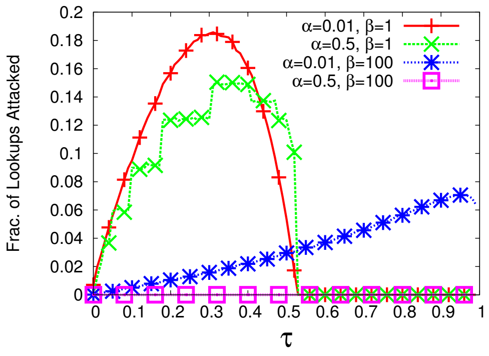


Fig. 8: **Analysis: One Threshold:** For varying τ and four combinations of α and β, the fraction of lookups attacked.

This is simply routing table pollution, however. As shown in Section 6, Kad-ReDS prevents this attack from being effective.

Another important mechanism for preventing whitewashing is to use a sufficiently low initial reputation score. whitewashing can be seen as an advance on the oscillation attack, in which the attacker attempts to gain a higher reputation score in the joining round than it would by staying in the system and behaving honestly in a standard oscillation attack. From the analysis of the oscillation attack in Section 8.2, we could identify the expected score at a time when the attacker's reputation reaches its nadir (say, s_{low}) and the benefit of whitewashing would be greatest. We should then set the initial reputation of joining nodes to s_{low} to remove the incentive for whitewashing as long as the attacker behaves optimally in the oscillation attack.

Bootstrapping Attacks. In the beginning phase of the system we do not have enough observations for nodes to build their own reputation scores. In this phase we give each node an initial reputation score, and the probability of a node being selected for the first lookup from a given k-bucket is $1/k$. If the node returns a bad result, then the requesting node immediately switches to another node in the k-bucket, limiting the effect of an all-out attack in the early phases of the system. With time, we get the required observations, and peers can distinguish between the honest and malicious nodes in their k-buckets.

Targeted Attacks on Keys. Attackers in ReDS may also try blocking access to a specific resource, or provide a malicious version of the resource, without attacking other lookups in the system, i.e., the attacker only manipulates lookups for a specific target key t. This is more challenging for ReDS than generic attacks because it can only be observed when the desired resource is being requested. We believe that limited tracking of attacked keys may be possible, but we leave further exploration to future work.

Targeted Attacks on Users. Similarly, an attacker may be interested in preventing a specific peer from accessing resources in the system. Since ReDS is most effective with the collaborative help of other nodes, the benefits of ReDS are limited against this attack.

8.2 Oscillation Attack

In an oscillation attack the attacker follows a strategic approach, alternately acting as a benign node and then a malicious node. By behaving as an honest node the attacker increases its reputation scores in order to increase the probability of being selected in future lookups, while performing malicious activities in later periods. This attack can be especially dangerous for ReDS when lookups are made in recursive mode, since this adaptive behavior is hard to observe when making indirect observations about the performance of lookup paths beyond the first hop.

We now analyze the effectiveness of the oscillation attack and show that it has limited ability to undermine the system, especially over time. The intuition of our finding is that the attacker must either lose opportunities to attack while rebuilding his reputation score or maintain a low reputation score and continually lose opportunities to attack.

Although in the rest of the paper we study a simpler version of ReDS, we explore a more general version of ReDS in this analysis. In particular, we leverage an exponentially weighted moving average (EWMA) to track nodes' scores with more emphasis on recent activity. The reputation score (s_{i+1}) of a given node just before lookup $i+1$ is given by: $s_{i+1} = \alpha r_i + (1-\alpha)s_i$, where r_i is the result of the lookup and α is the weight given to the most recent results. We also allow the node to select a peer from the k-bucket in a way that balances exploration (trying other nodes) and exploitation (making use of the known scores). To do this we set the probability of selecting the attacker node a (let us call this event A), who has score $s(a)$, as:

$$Pr[A] = \frac{s(a)^\beta}{\sum_{j \in k-bucket} s(j)^\beta},$$

where β is a weighting parameter. These two generalizations allow us to explore and understand the impact of these design choices in analysis. Further, they actually make our analysis easier, since this version of ReDS is probabilistic. A deterministic ReDS would be harder to study. Finally, since the main benefit of EWMA and exploration is to limit oscillation attacks, and oscillation is not part of our simulation attacker model, we explore these parameters here instead of adding variables to our already extensive simulations.

For the analysis we focus on a single attacker node and make the following simplifications:

- We examine the case when there is exactly one attacker and one honest node in a k-bucket.
- The honest node's reputation score is fixed at s_h.
- We do not consider churn.
- When the attacker acts as a benign node, its reputation score is not affected by the malicious activities of any other nodes in the lookup path.

The first assumption keeps our analysis tractable. We note that if there are multiple attackers in a k-bucket, then there are more interesting ways to do an oscillation attack. For example, one node can try to build its score while the other node attacks. Since it is not possible to

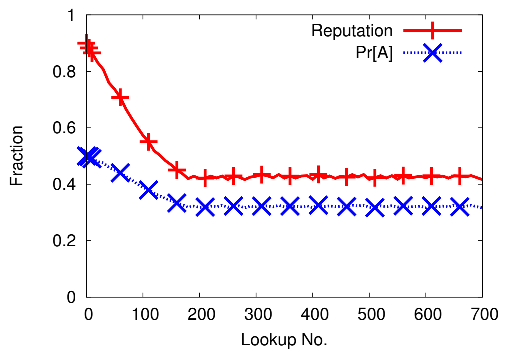


Fig. 9: **Analysis: One Threshold:** The reputation scores and $Pr[A]$ as the lookups proceed for $\alpha = 0.01$ and $\beta = 1$.

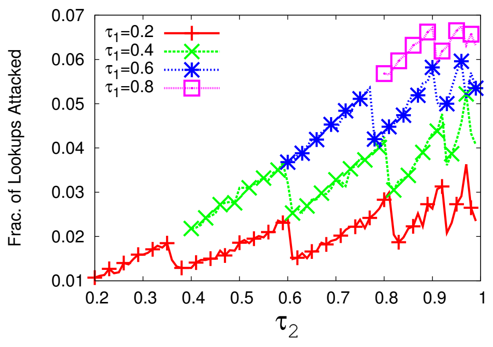


Fig. 10: **Analysis: Two Thresholds:** For varying τ_1 and τ_2, the fraction of lookups attacked.

'out-honest' the honest node, however, all attacker nodes will typically have a lower score than any honest node. At worst, the attacker could increase his attack opportunities linearly with the number of attackers in a k-bucket, with each attacker attacking in turn. It is important to note that for reasonable fractions of attackers in the system, having a large proportion of attackers in a k-bucket should be rare.

The second assumption is also for simplicity. If the honest node's score fluctuates moderately, it can open up small opportunities for the attacker when the honest node's score is low. While a full examination of these possibilities is beyond the scope of this analysis, intuitively there is little for the attacker to gain. As we show below, we can tune ReDS to pick the honest node after just one or two attacks from an attacker. An attacker could also attempt to attack only the honest node's reputation from the perspective of the querying node, e.g. by only attacking lookups that go through the honest node. Doing so, however, is a targeted attack that requires the attacker to sacrifice the reputation of his other nodes without successfully undermining many lookups. Such a targeted attack is beyond the scope of our analysis, as mentioned in Section 8.1.

By not considering churn we lose out on the attacker's remaining opportunity to get lookups to attack. We evaluate with churn in our extensive simulations.

The fourth assumption is the best strategy for our attacker. By coordinating his malicious nodes to attack all at the same time, he only risks losing reputation in an attack when he is also maximizing his chance to modify a lookup result. Thus, the oscillation attack is a global strategy.

To make the analysis tractable, we examine a limited set of possible functions for the attacker to select the probability of attacking a lookup: *one threshold*, *two thresholds*, and *probabilistic*. We examine each of these in turn.

One Threshold. We first consider a threshold τ in which the probability of attack on lookup i is $p_i = 1$ when $Pr[A]_i >= \tau$ and otherwise $p_i = 0$. This captures the intuition that the attacker should attack when it is being selected often enough to have an impact and otherwise it should rebuild his score.

The main metric we employ, and that our attacker seeks to maximize, is the expected number of lookups that the attacker can attack ($E[attacks]$) given the total number of lookups L that the user performs through the k-bucket of interest. This can be written as:

$$E[attacks|L] = \sum_{i=1}^{L} Pr[A]_i \times p_i.$$

Since p_i depends on $Pr[A]_i$, and each round's behavior depends on the results of the prior rounds, we did not seek a closed-form solution. Instead, we developed a simple numerical simulation of the above formula for a range of values of τ, α, and β. We examine the effect of τ for select values of α and β, as shown in Figure 8. We use $\alpha = 0.01$ as a *slow-learning* model, emphasizing longer histories, and $\alpha = 0.5$ as a *fast-learning* model, emphasizing recent behavior. Similarly, we use $\beta = 1$ as a *lightly biased* model, emphasizing exploration among k-bucket members, and $\beta = 100$ as a *heavily biased* model, emphasizing exploitation of knowledge. Figure 8 shows that the attacker can choose τ to attack a substantial fraction of lookups in both lightly biased models and in the slow-learning, heavily biased model. In these models the attacker can identify a peak at which τ is optimal for the model. However, we also see that the fast-learning, heavily biased model is very effective against this attacker, with $E[attacks] = 1$ at all values of τ, i.e. the attacker effectively never gets selected after the first attack. This is similar to the model that we use in our simulations.

To further break down how the attacker modulates its behavior, we examine the first few hundred lookups in the slow-learning, lightly biased model in Figure 9. We chose this model with $\tau = 0.32$ to show the best case for the attacker. We see that the attacker's reputation score steadily declines until oscillating around 0.42. For comparison, in our Halo-ReDS collaborative mode simulations with $c = 0.2$ fraction of attackers, we found that approximately 80% of honest nodes have reputation scores between 0.6 and 0.8.[2] The probability of being used ($Pr[A]$) similarly declines until oscillating around 0.32. The single threshold version of the oscillation attack is thus quite limited.

2. We expect that some honest nodes have lower scores from the perspective of a given node because they are tried a small number of times, have low success rates, and are no longer used due to other honest nodes being available in the k-bucket.

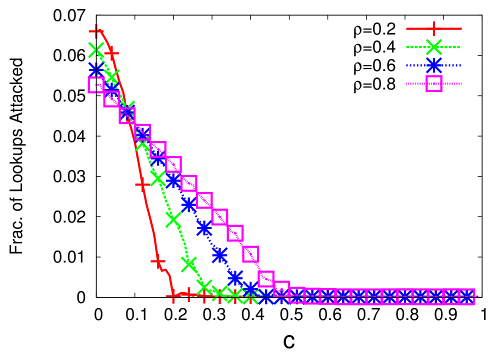


Fig. 11: **Analysis: Probabilistic:** For varying ρ and c, the fraction of lookups attacked.

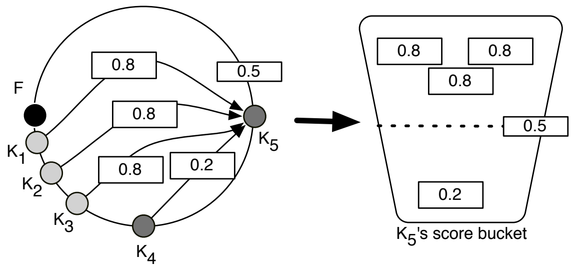


Fig. 12: **A use-based attack.** F, a malicious finger, attacks lookups from knuckles K_4 and K_5 but not those from K_1, K_2, and $K3$. Scores are reported to K_5, whose score bucket is shown on the right.

Two Thresholds. Oscillating behavior may occur over longer time scales. To examine this we extend the threshold model to include a lower threshold τ_1 and an upper threshold τ_2. The attacker will set $p = 0$ whenever $Pr[A] \leq \tau_1$, i.e. the attacker's reputation score has dropped too much to be selected very often. He will set $p = 1$ whenever $Pr[A] \geq \tau_2$, i.e. the attacker has built up sufficient reputation to attack again. The key question is how the attacker will set the thresholds τ_1 and τ_2.

In Figure 10 we see the attack rates for lookups in the slow-learning, heavily biased model. We have similar results for each model as with the one threshold attacker. First we note that in this model the attacker is never able to attack more than 7.1% of lookups. Further, the attacker's best strategy is to keep his range of scores quite high, requiring him to behave honestly for most lookups. In the fast-learning, heavily biased model, the attacker can never attack more than one lookup.

Probabilistic. Since the attacker can also employ a probabilistic attacking strategy, we also examine a probabilistic version of the oscillation attack. We let the attacker's probability p of attack for a given lookup i be:

$$p_i = \rho(Pr[A]_i - 0.5) + c$$

for attacker-chosen constants ρ and c. Although this function is linear, it covers a wide range of possible attacker policies. Figure 11 shows the change in number of lookups attacked in the slow-learning, heavily biased model for varying ρ and c. As with the other attacker models, the attacker has very limited success (again, he can only attack at most 7.1% of lookups). Additionally, the fast-learning, heavily biased model still only allows for one attack.

In sum, in all three attacker models the oscillation attack provides little to no advantage to the attacker.

8.3 A use-based attack on shared reputation

We now consider attacks on shared reputation. Despite the relative resilience of the Drop-Off scheme, it is vulnerable to a novel attack that greatly affects the possibility of shared reputation in ReDS. This *use-based attack* works against any ReDS system in which a given finger is used more by some knuckles than others. The attacker seeks to limit the loss of reputation from attacks while attacking as many lookups as possible. The attacker can achieve this by attacking the lookups from its knuckles who use his node as a finger more while not attacking lookups from other nodes. When the joint knuckles share reputation information about this malicious finger, they will have conflicting scores. The attacker's goal is to arrange its attacks so that the low scores are mostly ignored by other nodes.

We now describe a use-based attack in detail as applied to Halo-ReDS with shared reputation. A version of this attack should also work against Kad-ReDS with shared reputation, due to the XOR metric, or against any ReDS system in which a large fraction of lookups go through just a few fingers. For simplicity, we assume that each node in the Halo DHT performs the same number of lookups. The assumption is valid when peers perform a large number of lookups, and the probability of a given peer to initiate a lookup follows uniform distribution. With non-uniform distributions of lookup rates, the attack should have the same results on average.

In the use-based attack the attacker node acts as a malicious finger for m of its k knuckles and as an honest node for the remaining $k - m$ knuckles. An attacker node with ID a attacks the m most distant knuckles, as these knuckles use node a to cover a larger fraction of the ID space. In particular, a performs maliciously for the knuckles having ID $a - 2^{log(n)-i}$, where $i = 1, 2, \ldots, m$. Given l lookups using node a, we estimate that the number attacked on average will be $\sum_{i=1}^{m} \frac{l}{2^i} = l\left(1 - \left(\frac{1}{2}\right)^m\right)$.

We show an example of the attack in Figure 12. F is a malicious finger with knuckles K_1 to K_5. Here $m = 2$, meaning that lookups from K_4 and K_5 are being attacked, accounting for 75% of the lookups through F. K_5 is a new node with reputation score 0.5 for F, whereas K_4's score of 0.2 for F reflects F's attacks on its lookups. We show the reputation scores sent to K_5, which include three scores of 0.8 from knuckles K_1 to K_3 and 0.2 from K_4. Using the median the score will be 0.8, while using Drop-off the expected score is 0.75. In either case, the finger can thus attack many lookups while retaining a high reputation.

We further study the use-based attack in a simple simulator of the Drop-Off scheme, using $10,000$ nodes and $10,000$ lookup operations. Since node a may not always

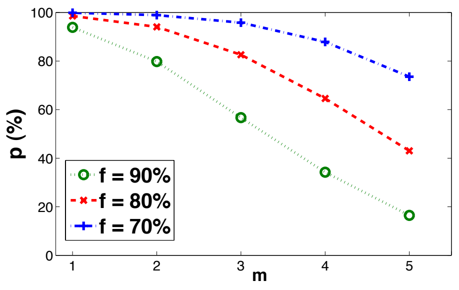


Fig. 13: Percentage of cases ($p\%$) in which a given attacked knuckle computes the reputation score of the attacker finger as 0.8 ($s = 80\%$)

behave the same to a given knuckle, we define two parameters. For the m knuckles for which a acts maliciously, let f represent the percentage of lookups through a that fail. Let s as the percentage of successful lookups through a for the $k - m$ of knuckles for which a acts honestly.

For example, if $s = 80\%$ and $f = 80\%$, the victim knuckles give a a score of 0.2 and the other $k-m$ knuckles, 0.8. In Figure 13 we consider the shared reputation scoring of the m knuckles when using all k scores. When $m = 1$, $s = 80\%$, and $f = 80\%$ the lone victim knuckle uses 0.8 as the shared reputation score of a in $p = 98.6\%$ of cases. At the same time, he can attack 50% of all lookups going through it. If an attacker acts maliciously for more knuckles, it causes more lookups to fail, but its credibility is decreased to those knuckles. Thus, the value of p decreases as we increase m. Figure 13 shows that for $m = 5$, we get $p = 43.1\%$, while the attacker can attack 78% of lookups.

In sum, the use-based attack enables the attacker to attack a majority or large fraction of lookups while still getting a good reputation score most or nearly all of the time.

Countermeasures. We first note that the Drop-Off scoring scheme may not be the best suited to stop the attack, as it is designed mainly to resist slandering and self-promotion attacks. Basic schemes, however, fare even worse. Using the median, the attacker would be able to attack half of its knuckles and still attain an excellent reputation score. For 10,000 nodes, this means the attacker could attack six out of 13 knuckles, covering 98.4% of lookups and have a perfect reputation score. Average is better, but is much more vulnerable to slandering and self-promotion. One could note that in Drop-Off, the node is ignoring its own score to its detriment. Making the score more centered on the node's own local score, however, means not obtaining any significant benefit from sharing reputation over only using first-hand observations.

One could attempt to design a scheme specifically to counter this attack, but it must also resist slandering and self-promotion attacks. For example, one could weight the scores of distant knuckles more heavily than nearby knuckles to reflect greater use by distant knuckles. Unfortunately, weighting the scores of any knuckles more heavily gives them greater power to perform slandering or self-promotion. Another countermeasure is to use a DHT in which all fingers are used equally. This suggests that Salsa, in which all local contacts are used equally [4], is more suitable for shared reputation. Considering the combined effect of the use-based attack, the limited benefits shown in our simulations, and the overhead of shared reputation, we recommend against shared reputation in ReDS.

References

[1] R. Akavipat, A. Dhadphale, A. Kapadia, and M. Wright, "ReDS: Reputation for directory services in P2P systems," in *Proceedings of The ACM Workshop on Insider Threats*, Oct. 2010, pp. 47–54.

[2] D. Wagner, "Resilient aggregation in sensor networks," in *SASN*, Oct. 2004.

[3] H.-C. Hsiao, C.-Y. Wang, J. M. Hellerstein, W.-C. Teng, and C.-L. Lei, "Veriable order statistics for secure aggregation," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-48, Apr 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-48.html

[4] A. Nambiar and M. Wright, "Salsa: A structured approach to large-scale anonymity," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, Oct. 2006.