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 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”). These helpers are currently 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: With 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 accordingly. 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

\(^1\) We note the requesting peer must verify the result by performing a handshake with the peer owning the ID, to ensure the node exists.
splitting up these ranges. For example, the second column space accurately down the chain of fingers by recursively chunks contiguous regions called chunk will go through finger \( f \) represents one chunk. For example, searches to the first finger \( (f_1) \) 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 to the first chunk will go through finger \( f_1 \), and searches to 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 \( f_3 \) as \( f_{3,1}, f_{3,2}, f_{3,3}, f_{3,4} \). 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 larger number of chunks are used, longer subpaths can be estimated.\(^2\)

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 \( \gamma \) observations. For example, using the chunk table shown in Figure 1, if \( \gamma = 5 \) and the lookup target is in chunk 12, 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 chunk 11 and 12 will be combined since they are covered by \( f_{5,3} \), which yields three records — still less than \( \gamma \). To get more observations chunk 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 \( \gamma \), 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 \( \gamma \) 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 than 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

---

\(^2\) We note the fingers may not line up exactly to chunk boundaries in the ID space, so the mapping to the reputation tree is only approximate. However, as the number of nodes increases, the number of boundary cases will be negligible.
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.

system parameter), we have that after $O(\log n)$ hops, there are at 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 $\beta$ 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., $\beta_{10}$ in Figure 2 is returned by two different queried nodes $\alpha_{22}$ and $\alpha_{25}$. 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 $\alpha$ nodes $\beta_{10}$, $\beta_{12}$, and $\beta_{14}$ 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 $\beta_{10}$ and $\alpha_{25}$ 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 to make Kad-ReDS robust against attackers who may attempt to insert a replica root close to the target key. 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 $\alpha$ nodes from the appropriate $k$-bucket. Each of the queried nodes in turn collaborates by providing the best $\beta$ 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.
Our primary concerns in 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 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. Figure 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 in-
creases. 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 $c = 5$, 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 $n_k$ 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:

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

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:

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

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 $E[s_k]$ is given by:

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

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 to 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.
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, $s_{\text{low}}$) and the benefit of whitewashing would be greatest. We should then set the initial reputation of joining nodes to $s_{\text{low}}$ to remove the incentive for whitewashing as long as the attacker behaves optimally in the oscillation attack.

**Bootstrap 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 observe 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 $\alpha$ 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:

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

where $\beta$ 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 oscillation attack. For example, one node can try to build its score while the other node attacks. Since it is not possible to
Frac. of Lookups Attacked

Fraction

0.2 0.4 0.6 0.8 1
0 100 200 300 400 500 600 700
Fraction

0.01 0.02 0.03 0.04 0.05 0.06 0.07
0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
Frac. of Lookups Attacked

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 $\tau_1$ and $\tau_2$, 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 $\tau$ in which the probability of attack on lookup $i$ is $p_i = 1$ when $Pr[A] >\tau$ 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 $\tau$, $\alpha$, and $\beta$. We examine the effect of $\tau$ for select values of $\alpha$ and $\beta$, 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 $\tau$ 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 $\tau$ 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 $\tau$, 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. The probability of being used ($Pr[A]$) similarly declines until oscillating around 0.52. The single threshold version of the oscillation attack is thus quite limited.

3. 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.
Two Thresholds. Oscillating behavior may occur over longer time scales. To examine this, we extend the threshold model to include a lower threshold $\tau_1$ and an upper threshold $\tau_2$. 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 $\tau_1$ and $\tau_2$.

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 $\rho$ 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 $\rho$ 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 $K_1$ 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 10000 nodes and 10000 lookup operations. Since node $a$ may not always 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