ReDS: Reputation for Directory Services in P2P Systems

Ruj Akavipat[†], Apurv Dhadphale[‡], Apu Kapadia[†], Matthew Wright[‡]

[†]School of Informatics and Computing
Indiana University Bloomington
Bloomington, IN, USA

[‡]Dept. of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, USA

rakavipa@indiana.edu, apurv.dhadphale@mavs.uta.edu,
kapadia@indiana.edu, mwright@cse.uta.edu

ABSTRACT

P2P systems rely on *directory services* for locating peers with the desired content and services. Directory services are themselves decentralized, such as with *distributed hash tables* (DHTs) that allow for efficient locating of objects without a centralized directory. As a system distributed over a diverse set of untrusted nodes, however, directory services must be resilient to adversarial behavior by such malicious insiders. While redundancy-based DHTs such as Salsa and Halo mitigate the effects of adversarial behavior, they incur substantial overhead due to redundant lookups. We propose *Reputation for Directory Services (ReDS)*, a framework for using reputation management to enhance the security and reduce the costs of redundancy-based DHTs in the face of insider attacks. We present ReDS designs for both Salsa and Halo, and we show that peers can significantly boost the success rates of directory lookups by considering past performance. For example, our simulations show that Salsa-ReDS can reduce lookup failure rates by up to 94%. We find that applying ReDS effectively cuts the redundancy required by both Salsa and Halo *in half* to get comparable results.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

1. INTRODUCTION

Peer-to-peer (P2P) architectures are gaining popularity and importance for applications ranging from massive-scale Internet content delivery to Internet telephony. For example, content delivery networks such as Akamai already deliver large quantities of their traffic using a distributed network, and are extending their reach to ordinary Internet clients using a P2P model [1]. The popular Skype Voice-over-IP system employs a P2P model, with a global decentralized user directory and calls routed through peers [9].

As a means to efficiently locate other peers and resources in a distributed setting, P2P systems must provide *directory services*. As a simple example, a directory service in a P2P file-sharing system lets a user know which other peers have the files she wants. These directory services are themselves decentralized, such as with *distributed hash tables* (DHTs) [10], which allow for efficient locating of objects without any centralized directory.

Since P2P architectures inherently must distribute functionality to a diverse set of peers across various network domains out of the control of any single authority, these peers cannot be fully trusted. The lack of a central authority for fundamental tasks such as routing and directory services means that the network must be resilient to malicious insiders to be usable by honest participants. Numerous reputation systems have been proposed to detect misbehaving peers and punish them or block them from using the the system. Hoffman et al. [4] provide an extensive survey of such systems. Most of these systems are focused on application-level information, such as the quality of resources (e.g., files) that a peer provides. Directory services, however, form the backbone of P2P systems. An unreliable directory service can render the entire P2P network unusable since adversaries can prevent wholesale access to files, or even selectively censor access to specific data. Furthermore, directory services have unique properties with respect to reputation that make them worth investigating separately from general-purpose reputation systems. *We propose Reputation for Directory Services (ReDS), a framework for using reputation management to enhance the security of locating information in distributed systems.*[1]

Sybil-detection approaches based on social networks [3,13] are important to ensure that the number of peers controlled by the attacker is limited. Unfortunately, social engineering can allow an attacker to connect to many points of the underlying social network and use those connections to remain undetected despite controlling a fraction of all peers. Against such an attacker, and as a way to provide defense in depth, the P2P directory service should be made resilient to attack. *Robust DHT systems*, such as Salsa[2] [8] and Halo [5],

[1]We presented the high-level ReDS idea in the context of Salsa in a 4-page work-in-progress paper [12].
[2]Although Salsa has been attacked for use as an anonymity

are an important first step towards secure directory services. They use redundancy in the lookup process to ensure that peers can find the correct information with high likelihood, even when a subset of corrupt nodes manipulate many of the lookups. These systems, however, also require a relatively high overhead — up to $O(\log n)$ *times* the cost of non-redundant lookups are needed to achieve high reliability when the attacker controls a modest fraction of the peers [5].

1.1 Contributions

Our first contribution is to show that reputation can be applied to robust DHT-based directory services to improve lookup success rates. Robust DHT systems compare the different results from redundant lookups to pick the closest match. Prior to this work, DHT systems did not use this information to improve their future results. For example, if five out of the ten lookups returned the incorrect result, this information can be used to avoid future lookups that share similar paths to the failed paths. This allows us to generate reputation based on first-hand observations.

Our second contribution is to detail how we apply this information to get substantially improved lookup performance beyond what simple reputation can provide. We apply ReDS to Salsa and Halo (we briefly describe these systems in §3). In particular, we give a model for directory services and the attacker (§2) and then present our designs for ReDS in a static system (§4.2) and then in a dynamic system (§4.3). These designs leverage the specific structure of DHTs to generate more detailed reputation information. DHTs are structured specifically so that lookups are routed through the network in a predictable fashion. Thus, based on the success or failure of various lookups, the original querier can make inferences about the nodes along the lookup paths.

Our third contribution is to show, through detailed simulations (§5.1), that ReDS peers can substantially improve their resilience to attacks with the same amount of lookup overhead. In Salsa-ReDS, for example, we can cut the failure rate by up to 94% in reasonable scenarios (§5.2).

Finally, we discuss some of the key issues that we have not fully addressed in this paper (§6), such as how ReDS handles adaptive adversaries and how reputation information could be shared between peers to further enhance performance.

2. SYSTEM AND ATTACK MODELS

In this section we describe the high-level system model for distributed directory services followed by the attack model we assume in this paper.

2.1 System model

The broader category of *directory services* includes all systems that provide a means for queriers to find information that enables them to access resources in the system, including all types of data and services. The Domain Name Service (DNS) is a well-known example of a directory service that translates human-readable addresses into IP addresses. A *distributed directory service* is a P2P directory service that stores information on the various nodes and adapts dynamically to nodes joining and leaving the system. Such P2P directory services are implemented using *distributed hash tables* (DHTs), and thus DHTs are the focus of our study. In particular, we focus on structured DHTs with redundancy such as Salsa [8] and Halo [5], both of which we describe in Section 3. These systems have many desirable features for the ReDS framework and provide a rich environment to study these features in detail.

DHTs support a distributed implementation of put and get operations, in which objects are indexed by keys. For example, objects can be inserted into the directory using put(key), and retrieved using get(key). In both operations, the directory service must first map the key to a particular *owner* o in the system. Once o has been located, the resource can be inserted or retrieved from o. DHTs such as Salsa and Halo provide a distributed operation called a lookup, in which the requesting peer calls lookup(key) to find the owner o of key. Thus, lookups are the fundamental low-level operation supported by DHTs and enable higher-level directory services to be implemented efficiently in a distributed setting. Lookup operations proceed by nodes iteratively (or recursively) looking up their routing tables and advancing the lookup operation closer to the destination. We call the nodes that route a lookup operation *fingers*. A lookup in a DHT with n total nodes involves contacting $O(\log(n))$ fingers. If no nodes in the system are malicious, lookup operations succeed with high probability (transient failures may occur when nodes join or leave the system).

Malicious nodes in the system may attempt to subvert lookup operations, e.g., by dropping, or misdirecting lookup operations. In this case, DHTs like Salsa and Halo use redundant routing strategies to improve the likelihood of success. Querying nodes have a list of *helper nodes* that are used to initiate the redundant lookups. In Salsa, they are called local contacts, while Halo uses the querying nodes' fingers as helper nodes. The helper nodes may return different values to the querying node. In DHTs like Salsa and Halo, the ownership relation is defined as the clockwise closest node to the object's key in the circular ID space. Thus, even if one redundant lookup returns the correct result, by taking the closest answer to the target search key, the querying node will obtain the correct owner.

Tran et al. propose an attack on Salsa in which the attacker provides an answer that is closer to the target than the real owner [11]. This attack is easily mitigated by having the requesting peer retain the search results until the owner is verified through a simple handshake protocol; each closest result is tested until the true owner is discovered.

2.2 Attack model

We study ReDS in the context of a malicious insider who seeks to manipulate directory service lookups. The adversary's goal could be to cause peers to use attacker-controlled nodes for services and information, for example as a way to spread spam or malware or to exploit peers requesting service. It may, however, simply be a denial of service strategy in which lookup results lead to invalid or incorrect nodes. To achieve these ends, the attacker's most effective strategy against a highly distributed network is to control a large number of peers (or virtual peers by controlling its location in the address space) in the system. Social-network-based anti-sybil techniques such as SybilInfer [3] and SybilLimit [13] may be employed to prevent the number of malicious peers from growing without bound. Nevertheless, we expect that through social engineering, the attacker may be able to inject a constant fraction of the total number of peers

system [2,7], it remains a viable robust DHT. The attack on lookups presented in [11] is easily mitigated via a handshake with the resulting peer.

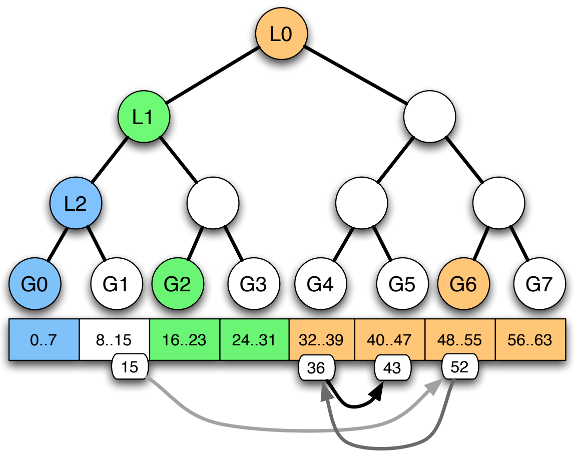

Figure 1: Salsa Virtual Tree: Node 15 in group G1 has global contacts in groups G0, G2, and G6. Arrows on the bottom show a recursive lookup path; the lines get darker as they get closer to the target.

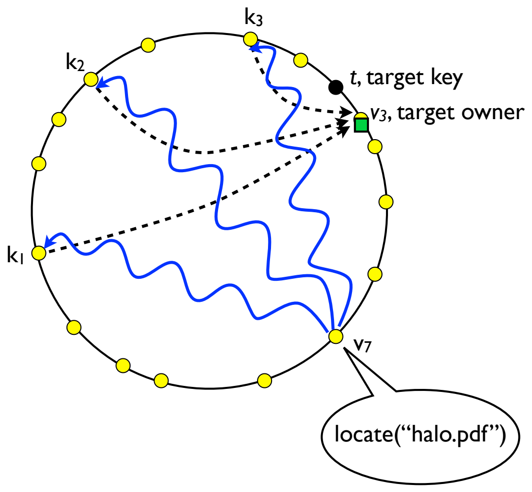

Figure 2: Halo Knuckles: Using the Halo technique, node v_7 performs three redundant "knuckle searches" (each of these is a regular Chord lookup, conceptually indicated with wavy arrows) yielding k_1, k_2, k_3, which in turn provide the location of v_3.

into the network without detection. We expect such an attacker to both directly manipulate lookup results as well as try to deceive any attempt at using reputation or malicious node detection. Thus, we must devise system designs that are robust to both types of attack.

In particular, we assume that once a lookup operation traverses through a malicious node, it will manipulate the subsequent lookup operation to map the lookup key to the closest malicious node rather than the true owner. For now, we also assume that the attacker will *always* subvert lookup operations, and we leave more sophisticated attack models to future work. As we will see, our ReDS approach keeps track of which helper nodes perform well for lookups to what regions, thus greatly mitigating such attacks. As a result, ReDS forces attackers to focus on more targeted attacks. Although we expect that adversaries who attack only part of the time will be reflected in reputation scores, we leave the study of such attack models to future work (see Section 6).

3. SALSA AND HALO

In this section, we briefly describe the Salsa and Halo systems, so as to better describe how we apply ReDS to each of them (see Section 5). We give a concrete example for Salsa, and provide a high-level description of Halo assuming familiarity with Chord [10].

3.1 Salsa

Salsa is a fully distributed directory service for node discovery in anonymity systems [8]. Salsa is a structured P2P system, similar to Chord [10], in which there is an ID space that is mapped to by a consistent hash function (e.g. SHA-1 mapping to a 160-bit ID space). The Salsa architecture is based on a virtual balanced binary tree, as depicted in Figure 1. Nodes are placed into groups created by dividing the ID space into contiguous regions — group G1 in the figure contains nodes with IDs in the range 8 to 15. All nodes know the peers in their group (their *local contacts*), as well as a small set of *global contacts* that are outside of their group. In particular, global contacts are selected randomly, but based on the virtual tree structure, as indicated by the colors in Figure 1. By using the global contacts in a recursive lookup process, as depicted by the arrows in the figure, a lookup will reach a member of the target's group in $O(\log_2 G)$ steps, where G is the number of groups in the system. To protect against malicious nodes manipulating lookups, Salsa uses *redundant lookups*, in which the requesting node asks a subset of its local contacts to perform lookups for the same target. Randomness in global contact selection provides *path diversity*, which prevents the redundant lookups from using the same node in the lookup path, since using the same node would negate the benefits of redundancy. A useful benefit of the structured ID space is that, when a requesting node receives conflicting results, the actual target owner is the closest to the target by definition. These benefits make Salsa a potentially useful directory service for many applications in which reliable service is needed, not just for anonymity.

3.2 Halo

Chord [10] is a well-known structured P2P system with good stability properties under independent node failures. Unfortunately, locate operations are easily subverted with adversarial behavior in Chord. For example, in a network with 10,000 nodes, just 10% malicious nodes can result in 50% failed lookups [5] (where the target key belongs to a non-malicious node). A naive redundant search in which, for example, a node v initiates five different searches for target t, is not particularly effective either, because the structure of Chord forces these paths to converge closer to the target. Even as high as 13 redundant searches in this case would still result in 20% failures in Chord. Halo [5] makes the simple observation that each node v occurs in $O(\log n)$ other nodes' finger tables. We call those nodes the *knuckles* of v. Thus searching for those knuckles instead of the actual target effectively "disentangles" the redundant searches. Kapadia and Triandopoulos [5] show that the knuckle locations can be accurately predicted, and lookups for these knuckles need at most one additional hop than a regular Chord lookup. They also show that knuckles exist 75% of the time at the predicted locations (this is an inherent consequence of the structure of Chord). While 25% of the redundant lookups directed at knuckles are destined to fail, the performance of Halo is still significantly better than Chord. In this example, less than 2% lookups fail with 13 redundant lookups.

3.3 Limitations

Salsa and Halo both use redundancy and path diversity to reduce lookup failures. A problem with these systems, however, is that lookup paths grow with the size of the system. Although this growth is merely logarithmic, it means that very large systems have paths with non-trivial lengths (e.g., for a system with 100,000 nodes, paths will be approximately 5.5 and 8.3 peers long for Salsa and Halo, respectively). If *any* of the peers on such a path is controlled by the attacker, the result from that lookup path will be corrupted, so longer paths are inherently less secure. Robust DHTs compensate for this by increasing redundancy; while sufficient to provide robust lookups, this adds greatly to the system overheads. For example, achieving high robustness requires $O(\log n)$ times as much communication in Halo for lookups than in Chord.

4. REPUTATION FOR DIRECTORY SERVICES

We now present the design of Reputation for Directory Services (ReDS). We first give a high-level overview of the design. We then describe the ReDS design in detail for a static system. Finally, we present a modified system for a dynamic environment in which the peers join and leave.

4.1 ReDS Overview

The first intuition behind ReDS is that robust DHT systems use redundant lookups (initiated through *helper peers*) 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.[3] We present three possible ways that a requesting peer can use this information:

1-Boost: The requesting peer can mark all the helper peers 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, 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 right helpers that maximize the chances of a successful lookup through reputable fingers. We call this approach *2-Boost* because it aims to choose lookup paths where the first two nodes in the path are honest.

Adaptive-Boost: The final step is to generalize 1-Boost and 2-Boost to the entire lookup path. In this approach, 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. Of course, this would result in a large data structure, and it 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 in the path as possible, as long as there are enough 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, the more lookups that are done, the more likely it is that a future lookup will share more of the path with prior lookups. If the prior lookups are a good predictor of future performance, this will allow the requesting peer to identify 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.

4.2 Static ReDS

We now describe more specifically how to implement ReDS with A-Boost in a *static environment*, in which peers remain in the system nearly all the time. While the static environment is not very realistic in most large P2P systems, it allows us to analyze and evaluate ReDS in more detail. We describe how ReDS can be implemented in a *dynamic environment*, i.e., one in which peers do leave and join the system, in Section 4.3. 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. For simplicity, we assume that each lookup follows a path in a balanced binary tree from the root to one leaf, and the leaves represent 2^m chunks, for some integer m.

An example of how the chunks are aligned with the fingers (f_i) of a finger f is shown in Figure 3—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_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 larger number of chunks are used, longer subpaths can be estimated.

The reputation score for helper node f for a particular tar-

[3]Again, we note the requesting peer must verify the result by performing a handshake with the peer owning the ID.

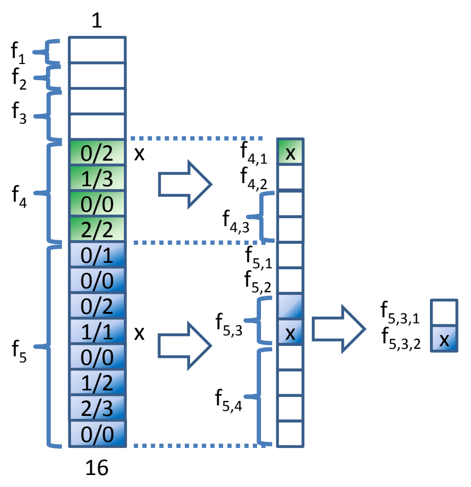

Figure 3: Example reputation tree. $f_{i,..}$ denote finger-covering address chunks (shown as boxes). 'x's mark examples of the lookup targets and highlighted boxes are chunks that would be combined if the number of observations in the smaller combination (shown in projections to the right) is insufficient. The number (v/y) in each box is the ratio of total recorded successes (v) to total recorded lookups (y) for that chunk.

get 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 3, 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 which is still less than γ. 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 γ, the reputation value of the finger is set to 0.5.

We observe that, in general, the reputation tree may not be a balanced binary tree. In this case, ReDS requires the tree of possible lookup paths have a maximum depth (i.e. maximum path length) of $O(\log(n))$ hops with no greater than $O(\log(n))$ fanout at each level. ReDS also requires the ID space can be divided into progressively smaller chunks that map to the possible lookup paths. Any robust DHT that fulfills these requirements in a static environment will provide enough lookups for creating a useful reputation tree. Given such a tree, there are myriad ways to calculate reputation scores and use them for selecting helper peers. We propose 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.

Salsa-ReDS. Applying ReDS to Salsa can be done by mapping the reputation tree to the Salsa virtual tree. Since the Salsa tree is also a balanced binary tree with groups as the leaves, this mapping is straightforward. However, a single real node may be represented by several virtual nodes in the tree. For example, in Figure 1, a lookup through helper peer 15 for an ID in group G6 will have three virtual nodes, but only one real peer (52) in the path. The requesting node will still get accurate reputation information about the path over time, but malicious nodes may not be detected as quickly as they would if the true path was known.

Halo-ReDS. Applying ReDS to Halo is similar. One difference with Salsa is that 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. Additionally, unlike Salsa, in which all redundant lookups have the same target t, 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 of target t, a finger with the highest reputation value based on the t's knuckle as a target is selected for the knuckle. Once a finger is selected it is removed from the selection pool of that redundant search. The process is repeated until R fingers are selected for the R knuckles.

4.3 Dynamic ReDS

In large P2P systems, peers generally 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. In ReDS, we mitigate these risks in two ways. First, we make the reputation scores follow an exponentially weighted moving average, so as to balance between the most recently observed behavior and longer-term behavior. Second, we increase the amount of exploration to ensure that the requesting peer has a diverse set of information about potential helper peers.

Moving Average. We first describe our use of an exponentially weighted moving average. Let $r(h_i, t)$ be the reputation score for helper peer h_i after t lookups. At time $t+1$, the requesting peer performs a lookup, for which helper node h_i provides a result that we describe with random variable X. $X = 0$ if the lookup through h_i fails and $X = 1$ if the lookup succeeds. The requesting node then updates the reputation score as:

$$r(h_i, t+1) = \alpha X + (1-\alpha) r(h_i, t).$$

Note that in a static system, this calculation generates reputation scores that oscillate near the score that would be given by taking the success rate. An initial reputation score has to be set for each node, e.g. $r_0 = r(h_i, t = 0)$. r_0 could be set according to the expected level of misbehavior or optimistically set to $r_0 = 1.0$. ReDS should not be used with a pessimistic setting such as $r_0 = 0.0$, as the attacker peer who provides a single good lookup result will have a higher reputation than any unused nodes.

This simple approach allows the reputation score to adapt to changes in the lookup path or in the behavior of any given peer. Further, the system can be tuned to the system's relative dynamism by changing the value of α. We can set α higher for more dynamic systems to emphasize more recent

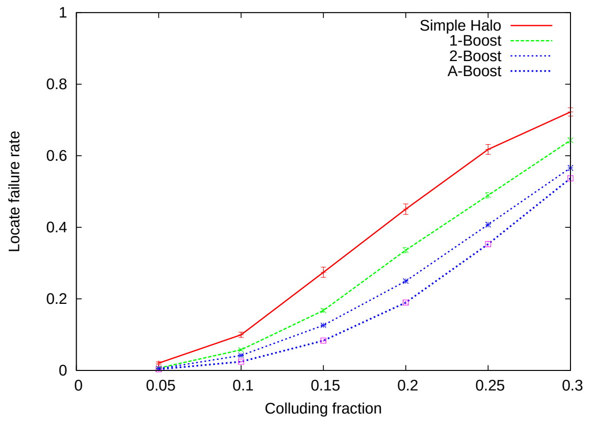

Figure 4: Locate failure rates for Simple Halo and Halo-ReDS under different percentages of colluding peers.

behavior, and lower in less dynamic systems to make better use of long-term information on peer reliability. α should never be set too low, however, as the attacker could take advantage of this to manipulate the system, without the system being responsive enough.

Enhanced Exploration. We now turn to the balance of exploration and exploitation. In Static ReDS, we emphasize exploitation of reputation information for the best lookup results. In dynamic systems, however, we must ensure that there is sufficient reputation information for a larger set of helper peers. When exploitation is high, relatively few peers are used for any given subtree, maximizing the chances of getting a good path. This means, though, that the loss of a small number of highly reputable helper peers or changes in their paths could cost the requesting peer much of reputation information that has been built up about those paths. The relative lack of information about other helper peers leaves the requesting peer without enough information to make good selections. To provide a balance between exploration and exploitation, we propose to select from all available helper peers, while probabilistically biasing the selection to more reliable peers.

In particular, we use the following mechanism to create a probability distribution for selecting helper peers. Consider helper peers h_i, for $i = 0, 1, \ldots m-1$, each with reputation score $r(h_i, t)$. We take the sum $S = \sum_{i=0}^{m-1} r(h_i, t)$. Then we define a probability value $p(h_i, t) = r(h_i, t)/S$ as the probability of selecting helper peer h_i. Note that $\sum_{i=0}^{m-1} p(h_i, t) = 1$.

For more flexibility in the balance between exploration and exploitation, we can calculate a modified reputation score $r'(h_i, t) = f(r(h_i, t))$ used to calculate the $p(h_i, t)$ values. We propose the function $f(x) = x^{\beta}$, where β is the *exploitation parameter*, a value that dictates the amount of exploitation by increasing or decreasing the relative differences between scores before they are used to created the probability distribution.

5. EVALUATION

We now present results from extensive simulations of Salsa-ReDS and Halo-ReDS. We first describe our experimental setup, then present the simulation results.

5.1 Experimental setup

We built simulators for Salsa-ReDS and Halo-ReDS in Java. In static mode, both simulators operate with a fixed set of attackers and nodes, with static routing tables. Our implementation of Salsa-ReDS supports continuous time simulation, enabling us to study a dynamic system in which nodes join and leave the system over time.

All our simulations were run for networks with 10,000 nodes. Each data point in our graphs corresponds to an average value over 100 different instantiations of the DHT. In each instantiation, a node was selected as the querying node. If a reputation system is present, the node first performed r training lookups to build its reputation system. Then 1,000 lookups were performed *without updating* the reputation system, and only the results from the lookup without updating the reputation system are used to measure the system performance. This approach allows us to take a *snapshot* of the system after the r training lookups, rather than collecting lookup outcomes while reputation systems are still adapting.

5.2 Simulation Results

Boosting comparison. Figure 4 compares regular Halo's performance with 1-, 2- and A-Boost algorithms. The simulations were run using a redundancy of four and 1,000 training lookups. The plot shows that A-Boost significantly improves Halo's lookup failure rate and consistently performs better than 1-Boost and 2-Boost, as expected. For example, with 15% malicious nodes in the network, Halo fails 27.4%, 1-Boost 16.8%, 2-Boost 12.6%, and A-Boost 8.3% of the time. With 30% malicious nodes in the network, the failure rate increses to 82.6% for Halo, 64.3% for 1-Boost, 56.6% for 2-Boost, and 53.7% for A-Boost.

The effect of training time on performance. Figures 5 and 6 compare the performance of "Simple" Halo and Salsa, respectively, with A-Boost applied to both systems. We show results for a redundancy of four after 100, 200, 500 and 1,000 training lookups. The graphs shows that the performance of A-Boost depends on the amount of information collected. For example, in Salsa, when the fraction of malicious nodes is 0.3 (30% of nodes are bad), A-Boost with 100 training lookups reduces the failure rate by 75%, while A-Boost with 1000 training lookups reduces the failure rate by 94%. Note that the latter is actually a 77% decrease over the former, from 7.4% failures with 100 training lookups to just 1.7% failures with 1000 training lookups. A-Boost does especially well at providing high assurance. For example, in Salsa with 15% malicious nodes, Salsa fails in 6.6% of lookups but fails in only 0.6% of lookups using A-Boost after 100 training lookups. We did not observe any failures after 1000 training lookups.

In Figure 6, we show results for Halo with A-Boost at the same redundancy, even though the locate failure rates are relatively high, to illustrate a lower assurance scenario. A-Boost continues to provide substantial benefits.

The effect of redundancy level on performance. Figures 7 and 8 compare the performance of Simple Salsa and Halo, respectively, with performance of both systems using A-Boost under different levels of redundancy. We simulated both systems with the following parameters: redundancy of 3, 5, 7 and 10; 100, 200, 500, and 1,000 training lookups; 25%

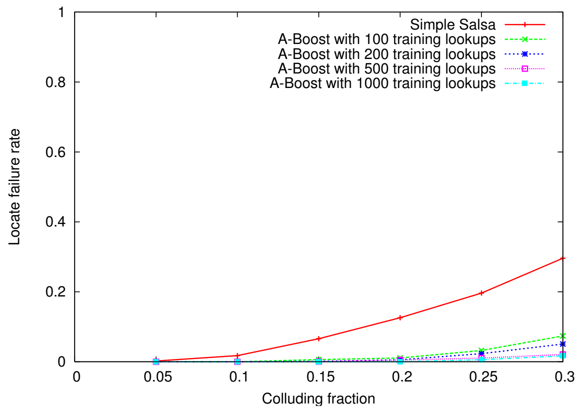

Figure 5: Locate failure rates for Simple Salsa and Salsa-ReDS with increasing training lookups.

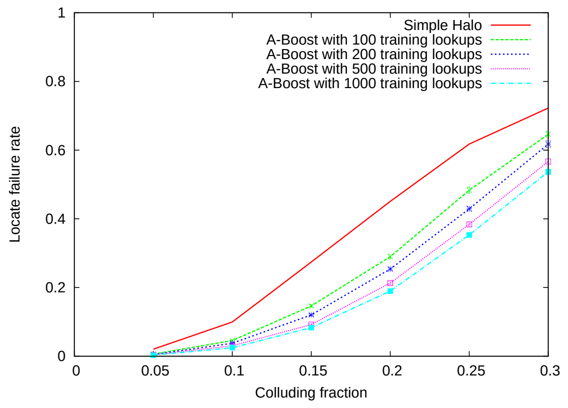

Figure 6: Locate failure rates for Simple Halo and Halo-ReDS with increasing training lookups.

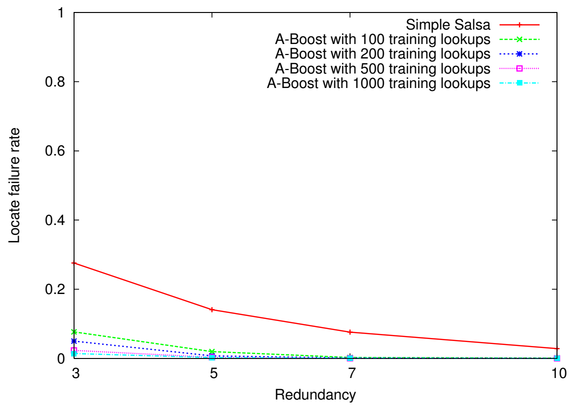

Figure 7: Locate failure rates for Simple Salsa and Salsa-ReDS with increasing redundancy for 25% colluding nodes.

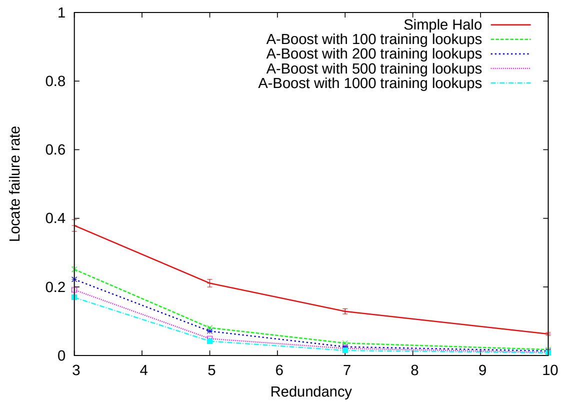

Figure 8: Locate failure rates for Simple Halo and Halo-ReDS with increasing redundancy for 15% colluding nodes.

colluding nodes for Salsa and 15% colluding nodes for Halo. Figure 7 shows that for Salsa with A-Boost after just 100 training lookups, we get almost the same performance for a redundancy of three as Simple Salsa with a redundancy of seven. After 500 training lookups, Salsa with A-Boost performs the same at redundancy three as Simple Salsa at redundancy ten. Halo shows similar results.

Dynamic environment. Figure 9 shows how A-Boost with moving average scoring and more exploration performs in a dynamic environment. The simulation has nodes leaving and joining the system at approximately the same rate as the requesting node makes lookups, i.e. one node leaves and one node joins on average for every lookup. We set the moving average weight $\alpha = 0.5$ and set $\beta = 1.0$ in the exploratory request scheme. The other parameters are the same as in Figure 5. We find that A-Boost also performs well despite the volatility in the system. As should be expected, since A-Boost must balance exploitation and exploration, the results are not as good. For example, with 30% attackers, the failure rate after 100 lookups is 11.6%, as opposed to 7.4% in the static system. Nevertheless, the results still show a substantial improvement over Simple Salsa.

6. DISCUSSION

Adaptive adversaries. Currently we assume that adversaries try to subvert *all* lookups. While we believe that our scheme will also effectively point out adversaries that are malicious for some fraction of the time (e.g., the adversaries may try to degrade the service of the DHT by attacking 20% of the time instead of 100% of the time as we currently assume), this hypothesis remains to be validated. We note that the attacker can only evade detection by substantially reducing his attacks. For example, if the attacker attempts to build up reputation, he must provide accurate results. Then his peers' reputation will quickly degrade relative to honest peers once his attack begins, as long as we apply Dynamic ReDS with appropriate parameters. We also note some adversaries might target certain types of content, and thus behave normally except for when queries are made for certain types of content. We believe it is possible to extend our technique to build reputation for types of content and other possible targets.

Coordinated Attacks. An attacker may attempt to determine when he has a good chance to control *all* redundant copies of the same lookup and only modify the lookup results when he will be successful. While this may require real-time

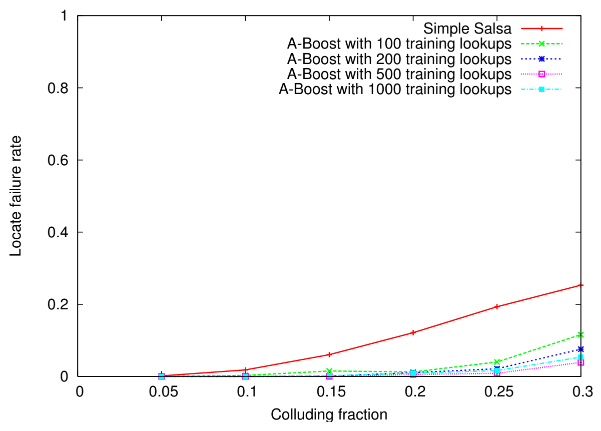

Figure 9: Comparing Simple Salsa and Salsa with A-Boost in the dynamic environment for redundancy 4 under different percentages of colluding peers.

coordination, it is a threat to undermine ReDS. This threat can be mitigated by conducting the redundant lookups in sequence, rather than in parallel. This requires the attacker to decide at the first lookup whether to attack or not. The additional delay due to this technique can be partially offset in some applications by performing lookups in advance. It may also be possible to tradeoff between security and performance by conducting lookups partially in parallel. Since the attacker can attempt to delay the early lookups to gain an advantage, unusually slow lookups can be used as part of the reputation score.

Shared reputation. In this paper we study how peers can boost the success rates of their lookups by maintaining only first-hand reputation information. In particular, nodes do not share second-hand reputation information with other nodes and, as demonstrated, must perform several training lookups each to build up reputation effectively. As Hoffman et al. note [4], there may often be insufficient data for reputation when only first-hand information is available. Lagesse et al. combat this problem with *exploratory requests*, in which the requesting peers asks for irrelevant and verifiable information purely to build reputation scores [6]. While exploratory requests may be necessary in some systems, we note that redundant requests in robust P2P systems provide a large pool of verifiable requests with no additional overhead just for building reputation scores.

In the future, we plan to explore strategies to share such reputation information to potentially lead to faster convergence of reputation and possibly drastically reduce the number of training lookups. Since we assume adversarial settings, a reputation sharing scheme would have to be robust against adversaries who can falsify reputation information.

7. CONCLUSIONS

In this paper, we propose *Reputation for Directory Services (ReDS)*, a general framework for reputation management to enhance the security and performance of robust distributed hash tables (DHTs) in the face of malicious insiders. We describe how to apply ReDS to two DHTs, Salsa and Halo, and show through simulation and analytical analysis that Salsa-ReDS and Halo-ReDS both perform significantly better than without ReDS after as few as 100 training lookups. We believe that the ReDS framework represents a significant step in improving the robustness of P2P systems and that more can be to build on this design.

8. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under CAREER award number CNS-0954133. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

9. REFERENCES

[1] Akamai netsession interface. `http://www.akamai.com/html/misc/akamai_client/netsession_interface_overview.html`.

[2] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *CCS*, Oct. 2007.

[3] George Danezis and Prateek Mittal. SybilInfer: Detecting sybil nodes using social networks. In *NDSS*, Feb. 2009.

[4] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Comput. Surv.*, 42(1):1–31, 2009.

[5] Apu Kapadia and Nikos Triandopoulos. Halo: High-assurance locate for distributed hash tables. In *NDSS*, Feb. 2008.

[6] B. Lagesse, M. Kumar, and M. Wright. Arex: An adaptive system for secure resource access. In *P2P*, Sep. 2008.

[7] P. Mittal and N. Borisov. Information leaks in structured peer-to-peer anonymous communications. In *CCS*, Oct. 2008.

[8] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *CCS*, Oct. 2006.

[9] P2P telephony explained—for geeks only. `http://www.skype.com/help/guides/p2pexplained/`.

[10] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM Conference*, Aug. 2001.

[11] A. Tran, N. Hopper, and Y. Kim. Hashing it out in public: common failure modes of DHT-based anonymity schemes. In *WPES*, Nov. 2009.

[12] Matthew Wright, Apu Kapadia, Mohan Kumar, and Apurv Dhadphale. ReDS: Reputation for directory services in P2P systems (extended abstract). In *CSIIRW*, Apr. 2010.

[13] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit: A near-optimal social network defense against sybil attacks. In *IEEE Symp. on Security and Privacy*, May 2008.