Halo: High-Assurance Locate for Distributed Hash Tables

Apu Kapadia
Institute for Security Technology Studies
Dartmouth College
Hanover, NH 03755, USA
akapadia@cs.dartmouth.edu

Nikos Triandopoulos
Department of Computer Science
University of Aarhus
8200 Aarhus N, Denmark
nikos@daimi.au.dk

Abstract

We study the problem of reliably searching for resources in untrusted peer-to-peer networks, where a significant portion of the participating network nodes may act maliciously to subvert the search process. We present a new method called Halo for performing redundant searches over a distributed hash table (DHT) *structure to achieve high integrity and availability levels without affecting the storage and communication complexities of the underlying DHT. Other schemes for redundant searches have proposed new or modified DHTs with increased storage requirements at nodes, requiring modifications at all nodes in the network. In contrast, Halo aims to serve as a middleware component, making "black-box" calls of the underlying primitive search operation to eventually provide a new composite search operation of higher assurance. We apply this concept to the popular and well-studied DHT Chord, and demonstrate the efficiency and security of our approach though analytical modeling and simulation-based analysis. For example, we show that for 12% malicious nodes in the network, a regular Chord operation fails 50–60% of the time. In contrast, Halo reduces this failure rate to 1%. We show how our scheme lends itself to a recursive version that can tolerate 22% malicious nodes with the same level of success, while regular Chord fails 70–80% of the time.*

1 Introduction

Peer-to-peer (p2p) storage networks currently constitute the most developed computing architecture for implementing large-scale distributed data-management systems. These networks realize a decentralized computing infrastructure for dispersing data and computing resources among a large number of geographically-distributed machines. PAST [8], CAN [25], CFS [5], PIER [17], Kademlia [21], and OpenDHT [27] comprise a representative (and certainly not exhaustive) set of p2p applications. Conceptually, in any p2p network a resource (such as a file) is mapped to a unique participating peer, which is responsible for storing that resource. The core functionality of any p2p network amounts to efficiently *locating* resources in the p2p network. A *distributed hash table* (DHT) is a distributed data structure that implements this very functionality: given a target resource identifier, the *locate* operation returns an identifier (typically, the IP address) of the node responsible for the target resource. Locating objects usually involves a distributed search between a small subset of participating peers that share resource-allocation (or routing) information. Thus far, researchers have studied many aspects of this searching functionality—efficiency, search-structure maintainability, fault tolerance, range-search extensions, and load balancing to name a few.

Reliable searching in the face of attack. In this paper, we study secure resource location in p2p networks, trying to provide a practical solution to the fundamental security problem related to searching over p2p networks: *how can one reliably locate resources in the presence of malicious colluding network nodes?* We assume a *random Byzantine* model where a randomly selected subset of nodes can perform arbitrarily malicious behavior while participating in the distributed search process. For instance, a coalition of malicious nodes could easily attempt to redirect queries to a private p2p subnetwork that runs in parallel with an aim to degrade the performance or simply perform a denial-of-service attack. Moreover, malicious nodes are motivated to subvert a locate operation, by reporting a false malicious "owner" of a file (or other resource), rather than its true owner, thus drastically affecting the core storage functionality of the system. In particular, such behavior not only affects the integrity of data, since a malicious owner can falsify data during the reporting phase, but also the availability of data, since the false owner can also subvert the storage phase by simply discarding the received new file or disallowing future data retrievals (i.e., a file is practically "invisible" since it is erroneously stored in the wrong network node). In general, the location process in a p2p system defines an "ownership" relation between shared resources and participating nodes. By subverting this process, therefore, the adversary can perform a rich set of attacks on the integrity of the p2p system. For instance, with malicious resource locations the adversary can manage to alter the structure of the overlay network and affect the system's consistency, fairness, or load-balance. Any secure searching method should not only detect any possible attack, but more crucially, *locate* resources, effectively tolerating adversarial behavior during the search process. While existing cryptographic schemes [34] may be used to check the integrity of data, we focus on the orthogonal problem of actually locating the real data, whose integrity may then be verified cryptographically.

Solution: redundant searches to locate target. Following an algorithmic approach, we present an efficient technique for secure searching in p2p networks that exploits the power of performing a small number of carefully selected *redundant locate operations*. The main challenge with redundant searches is that due to the inherent nature of routing within DHTs, multiple redundant searches converge to visiting the same small set of nodes "close" to the target node, thereby making the redundant searches ineffective—since they may all overlap with the same malicious node(s). To circumvent this limitation, much of the existing research (such as with Cyclone [1] and Salsa [22]) has focused on "disentangling" these searches by either modifying the Chord [33] data structure or by proposing entirely new DHTs to perform multiple *disjoint redundant searches*. Unfortunately these new schemes come at the cost of increased storage at each node in the network. The problem with increased storage goes beyond actual storage constraints—maintaining up-to-date routing information to more nodes increases the complexity of maintaining the DHT. Furthermore, these schemes require either completely new DHTs or modifications at all the nodes within the network, and therefore do not provide an easily-applicable security solution for the plethora of existing and fully developed DHTs.

Ideally, disjoint redundant searches should be implemented in a DHT *without changing* any structural characteristics or operational modes of the DHT, or affecting its performance guarantees beyond the obvious overheads due to redundancy in the searches. We follow this approach and provide a novel scheme to create disjoint redundant searches in a DHT by modifying only the core search algorithm of the DHT (essentially by repeatedly using it) . The underlying idea is simple: *we make the observation that the target of a locate operation exists in several routing tables of nodes distributed in the DHT*. We call these nodes "knuckles."[1] Instead of searching for the actual target along several paths, we search for the knuckles to get the correct answer for a locate operation. By doing so, redundant searches are disentangled without any modification to the underlying DHT—the storage requirements remain the same, and in fact, already-deployed nodes in a existing live p2p network need no modifications. Nodes can choose to make use of our algorithm to perform redundant searches while using the rest of the network as a "black box." We call our technique *High-Assurance LOcate (Halo)*.

We apply Halo for securing resource location in Chord, which represents perhaps the most popular class of DHTs. We theoretically analyze Halo for Chord, proving its correctness, efficiency and practicality. We present a simulation-based evaluation of Halo, confirming its performance analysis. We show that Halo is able to tolerate up to 12% malicious nodes in a network of 10,000 nodes. The malicious nodes are able to subvert only 1% of searches. In contrast, a regular locate operation in Chord fails 50–60% of the time with 12% colluding nodes. We also apply Halo recursively (to find the knuckles of the knuckles), which is able to tolerate 22% colluding nodes with only 1% failed searches, whereas Chord fails 70–80% of the time. Defending against higher rates of collusion is impractical, mainly because at those rates the *true owner of an object is malicious* with a high probability, signaling more endemic problems with the network.

Replicas vs. redundant searches. DHTs provide basic put-get functionality for storing and retrieving data objects in a p2p network. It would be reasonable to speculate that certain data-integrity guarantees can be achieved by using redundancy at the put-get level of the p2p system. For instance, a data file can intentionally be stored in a small set of "randomly" selected nodes, with the hope that at least one copy can be retrieved correctly,

[1] In Chord, a node's routing table contains forward pointers to other nodes, which are called *fingers*.

even under adversarial network behavior [15]. Unfortunately, this approach is expensive in practice; in addition to downloading multiple copies of potentially large files, the operations put and get are not atomic: they are both realized by the underlying and most primitive locate operation, and are themselves subject to redirection attacks. As a result a large number of replicas are needed to guarantee integrity, further increasing the storage demands of the network. Securing the locate operation, on the other hand, aims to solve the problem at the lowest level of the hierarchy. In fact, a secure locate operation can significantly reduce the overheads of a higher-level replica-based solution. Moreover, locate operations constitute the main primitive for implementing many other important operations in a DHT, like updating routing information, joining and leaving the network, and so on.

Paper contributions. Our contributions can be summarized as follows:

1. We present a novel algorithmic approach called Halo for performing *high-assurance locate* for resources in p2p networks using redundancy.

2. By design, Halo makes use of the underlying network as a "black box," and thus does not increase the storage requirements for nodes, and also makes our solution easy to deploy or apply.

3. By providing both analytical models and simulation-based results, we demonstrate the effectiveness and practicality of Halo in Chord: a few redundant searches suffice to significantly improve the reliability of Chord's functionality.

Paper structure. After preliminaries in Section 2, we present our Halo construction for Chord and prove its correctness in Section 3. In Section 4, we present an analytical model of Halo and in Section 5 we present an experimental evaluation. We present an overview of related work in Section 6 and conclude in Section 7. The Appendix includes additional simulation graphs.

2 Preliminaries

We start by describing basic concepts and introducing some useful terminology for our solution.

2.1 Distributed Hash Tables (DHTs)

An overlay peer-to-peer (p2p) network is a network structure imposed on a subset of machines from an underlying larger computer network. The most elementary p2p network structures have been designed for supporting the fundamental (and necessary for any practical application) put-get functionality defined over keyed data objects. *Distributed hash tables* (DHTs) is a class of network structures and associated search protocols comprising these fundamental operations for storing in a p2p network (at some network node) an object x under key k_x and later retrieving from the network (from the same network node) object x using key k_x. Any pair of machines can communicate directly if one of them is given the network address of the other machine. Due to scalability issues, complete network representation at each participating peer is practically prohibited. Instead, network nodes typically keep minimal structural (routing) information about the p2p network by storing pointers to a small set of carefully selected network nodes. Accordingly, the overlay network is defined by a graph that specifies which machines are linked by these pointers, and it should provide algorithms for storing and locating data of interest in the overlay network.

Much of the functionality and the protocol properties of any DHT depend on two important features: (1) the *network structure*, i.e., the underlying graph representing direct network-connectivity capabilities, and (2) the *mapping* of data objects to network nodes, i.e., a systematic way with which resources are associated with network nodes for storage in the p2p system. To rigorously define these two concepts and facilitate the implementation of such a network, both concepts are defined by using a large, totally ordered *logical ID space*. Both network nodes and data resources are first assigned (using a possibly probabilistic procedure) a logical ID in this space; then, network connectivity and resource association are deterministically defined by functions operating and ranging over this logical ID space. Using these functions, one can then define a deterministic procedure mapping data resources to network nodes storing these resources. We call this operation locate: in particular, given a resource identifier x, operation $\mathsf{locate}(x)$ returns the network node identifier that x is mapped to. We say that $\mathsf{locate}(x)$ *owns* resource x.

More formally, let $q = 2^m$ be the size of the logical ID space $\mathcal{U}$, which can be considered identical to the set $\mathbb{Z}_q = \{0, 1, 2, \ldots, q-1\}$, where the successor relation is well defined using modular arithmetic. Let $\mathcal{N}$ and $\mathcal{R}$ denote the finite sets of identifiers of the possible network's nodes in the p2p system and, respectively, of the data resources to be used in the system (i.e., data objects that can be stored in the system). Typically, $\mathcal{N}$ can be the set of IP addresses and $\mathcal{R}$ any set of keys under which objects are stored in the system. Let $N = \{v_1, v_2, \ldots, v_n\} \subseteq \mathcal{N}$ be the set of existing nodes in the system. Then, any DHT structure built for N defines functions f, g, binary relations B, E and a procedure locate such that:

1. $f : \mathcal{N} \to \mathcal{U}$ is a function mapping network nodes to elements of the logical ID space.

2. $g : \mathcal{R} \to \mathcal{U}$ is a function mapping data resources to elements of the logical ID space.

3. B is a binary relation over $\mathcal{U}$ denoting resource ownership. That is, $(g(r), f(v)) \in B$ if and only if resource r belongs in node $v \in N$ (equivalently, node v owns resource r). By definition, we require that any resource belongs to a *unique* node. Note that r is not necessarily a data object identifier (e.g., file name), but any abstract resource identifier in $\mathcal{R}$ (e.g., identifying a computational resource).

4. E is a binary relation over $\mathcal{U}$ denoting direct routing capabilities in the p2p system such that directed graph $G_N = (V, E)$ defines the underlying overlay network of the p2p system. That is,

   $$V = \{u \in \mathcal{U} : u = f(v_i), v_i \in N\}$$

   and $(f(v_i), f(v_j)) \in E$ if and only if $v_i, v_j \in N$ and node v_i contains routing information about node v_j.

5. locate is a (distributed) procedure that on input a resource r and any node $v_i \in N$, using (partially only) graph G, returns the unique owner $v_r \in N$ of r, i.e., node v_r such that $(g(r), f(v_r)) \in B$.

2.2 Chord

We now instantiate the above terminology for Chord's DHT [33]. The logical space is exactly $\mathbb{Z}_q$, with $q = 2^{160}$, a large set of integers conceptually organized in a ring, i.e., in a circular fashion where successor logical IDs appear clockwise in the Chord ring. (In particular, the successor of $2^{160} - 1$ is 0.) Both functions f and g are set to be the SHA-1 hash function, that is, an efficient randomized function that is also believed to be first and second pre-image resistant: it is computationally hard, given x, to come up with y satisfying $x = f(y)$ and also, given y, to come up with $z \neq y$ satisfying $f(y) = f(z)$.[2] Thus, n network nodes identified by their IP addresses, and all possible resources identified with unique keys, are mapped into the Chord ring in an unpredictable, uniformly random, but consistent, fashion.[3]

Relation B is defined using the successor relation in the logical ring: node u that hashes to position z owns all resources r that hash to positions in the ring for which z is the *immediate successor* (in the clockwise direction). In other words, when mapped to the ring, the n network nodes partition the ring to n chords; all resources mapped to one such chord are owned by the node mapped to the clockwise-largest end point of this chord.

The underlying connectivity graph G is defined by having each node in the network being "connected with" (i.e., knowing direct routing information about) $O(\log n)$ other network nodes, called *finger* nodes. This exact set of routing information for network node v mapped to position $u \in \mathbb{Z}_q$ is defined as follows (see Figure $1(a)$). First, in a deterministic way, $m = \log q$ finger positions in the ring are associated with u: these are defined by considering successor positions of u using offsets of exponentially growing size, that is, positions $u + 2^0, u + 2^1, \ldots, u + 2^{m-1}$, $m = \log q$. Then, the network node v at position u stores in its routing table the IP addresses of the network nodes owning these m finger positions in the ring; in total, $O(\log n)$ distinct IP addresses are stored, which actually correspond to the owners of the $O(\log n)$ "most significant" positions (i.e., the positions that correspond to the longest exponential offsets).[4] This exact set of finger nodes of v is a random variable that depends on the original placement of nodes on the logical ring. Additionally, each node v knows its immediate successor node v' in the ring (and it can also know its predecessor node).

Finally, given this structure, the locate operation is implemented by iteratively, or recursively, using the local routing information for locating the best estimate of the owner of the searched key, i.e., the closest finger to the destination (see Figure $1(b)$). That is, starting from any initial node in the p2p network, when node v is queried for ID k during the location search process, node v directs the search to its finger node for which k

[2]These properties, and despite the recent attacks, make SHA-1 the best candidate for implementing the random oracle model.

[3]This means that a participating peer has practically no control in choosing its position in the ring, an important property, as we will see.

[4]In fact, routing tables store extra information that essentially maps $O(\log n)$ continuous ranges (chords) in the logical space $\mathbb{Z}_q$ to IP addresses.

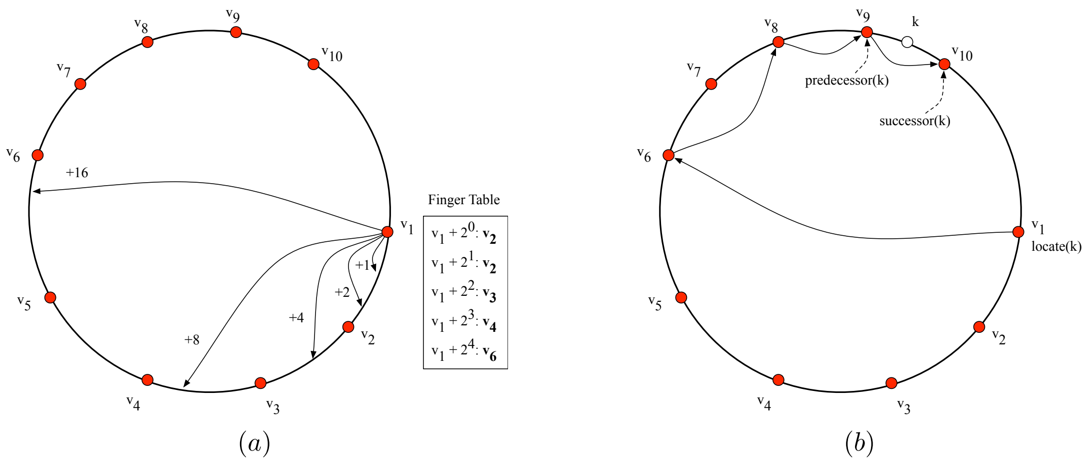

v9
v8
v10
v7
v6
+16
Finger Table
v1
$v_1 + 2^0$: $\mathbf{v_2}$
$v_1 + 2^1$: $\mathbf{v_2}$
$v_1 + 2^2$: $\mathbf{v_3}$
$v_1 + 2^3$: $\mathbf{v_4}$
$v_1 + 2^4$: $\mathbf{v_6}$
+1
+2
+4
+8
v5
v2
v4
v3
(a)
k
predecessor(k)
successor(k)
locate(k)
(b)

Figure 1. Chord's underlying graph and location search operation. (a) **Each node maintains** $O(\log n)$ **"fingers" to other nodes, where** n **is the total number of nodes in the network and the distance between a node and its fingers increases exponentially.** (b) **Iteratively, the locate operation** locate **for key** k **routes the search to the currently closest finger until the immediate predecessor network node of** k **is reached. The predecessor returns its successor as the successor network node for key** k**, that is, the owner of** k**. Because of the exponential distances between fingers, a target is located in** $O(\log n)$ **steps.**

is "clockwise-closest" from the finger node; this finger can be easily found by accessing the local routing information of v. Since fingers are organized in exponential steps, each iteration of the search would on average reduce the search space by at least a factor of 0.5. Thus, the locate operation takes on average $O(\log n)$ steps to correctly find the owner of the searched key. Note that in the real implementation of Chord, the locate operation on a search key actually returns the predecessor node v of k, and the real owner v' of k is simply found by executing the simple find_successor operation on v.

For simplicity we often refer to node v mapped to position u in the ring as simply node u; we use this convention also for target, predecessor and successor nodes.

2.3 Threat model

Secure searching in p2p networks is a fundamental concern, exactly because by definition no central administration procedure or control mechanism is implemented to govern the system functionality and verify its integrity. Ensuring trustworthy system functionality, therefore, heavily depends on reliable resource searching. Because of their highly distributive nature, p2p networks correspond to an inherently powerful threat model: any participating network node or any coalition of nodes can easily exhibit a malicious behavior and thus not conform with the distributed protocol designed for implementing the locate operation. Trivially, a node that is accessed during the location search process can effectively fail the search, by either stopping or maliciously redirecting the search to an arbitrary or also malicious node.

To capture the above threat, we consider that a constant fraction c of the n network nodes in the p2p system are controlled by an adversary and can thus act maliciously. We adopt the random Byzantine adversarial model. Each network node participates in the coalition of malicious nodes with independent probability c, $0 < c < 1$. This independent-probability assumption is justified because in most p2p networks participating nodes cannot control their locations in the logical ID space, since these are usually determined by a cryptographic hash function, and also because there is very little choice in maliciously selecting the network-node IDs, since IP addresses are generally difficult to be set to a special target value. We note that in theory it is possible for a malicious node with access to a large range of IP addresses to gain control of a resource. In a network of a million nodes, for example, the adversary would need to control approximately one million IP addresses to become the owner of a particular ID space. We assume that a set of colluders does not have access to a large number of IP addresses and, therefore, can at best be uniformly distributed over the logical ID space. We impose no other assumption about the malicious coalition. In particular, members of the adversarial coalition are considered to have complete information; that is, they share their individual routing information or any other information they wish: specifically, they can have complete knowledge of the underlying overlay network structure.

Our threat model adopts the following worst-case attack scenario, where the adversary's goal is not simply to redirect the search once to some arbitrary new location, but instead to unsuccessfully terminate the search at a node in the coalition. Since function f is public, any querier searching for resource r knows the search key $k = f(r)$, i.e., the position in the ring that r maps to, the adversary should falsify the search in the best possible way that is unlikely to be detected by the victim querier. Accordingly, any node v in the network that is contacted during a location operation that searches for key k, reacts as follows: if v is not in the set of malicious nodes, it runs the correct algorithm for redirecting the search; otherwise, v immediately maliciously terminates the search by redirecting to the clockwise-first (closest) malicious node that succeeds the actual owner of target k. We consider that the coalition of malicious nodes chooses the clockwise left-most adversarial node in the ring that succeeds the target to best foil detection. Notice that our convention that this redirection occurs in an atomic step is (i) feasible, because the adversary has complete knowledge of the overlay network structure, and (ii) not restrictive, because the adversary could simply perform the same result by redirecting the search through nodes in the coalition, i.e., pretending that a "normal" search is executed.

2.4 Redundant searches in Chord

As we have described, our goal it to find solutions for secure p2p searching that are practical and easy to implement and at the same time make only black-box use of the primitive locate operation. In particular, to achieve these goals we wish to employ *redundancy* for augmenting the search process in a way that provides *high-assurance* results. That is, at the necessary cost of increasing the searching complexity due to the redundant searches, we aim at designing a redundant search method such that the random positions of the coalition are avoided and the target key is successfully located. These searches would ideally "go through" different network nodes, thus increasing the chances that at least one, *easily identifiable* correct search exists for any random data resource and any random subset of malicious nodes. Using ℓ-redundancy, that is, performing ℓ redundant, different and independent searches, where ℓ is a small integer, we can trade-off communication to better tolerance against misdirection attacks. This trade-off seems to have a good pay off: more and more bandwidth is no longer a concern in high-speed networking architectures; in particular, each search-related communication between nodes in a p2p network (e.g., Chord) corresponds to exchanging only a constant number of information (in practice, only a few bytes payload).

A naive approach for implementing ℓ-redundancy would be to simply perform ℓ different locate operations for the given target starting from different (e.g., random) initial nodes. Most classes of DHTs such as Chord, however, share the following problem: the straightforward implementation of this approach fails due to the fact that even randomly selected searches overlap in a large set of neighboring network nodes (see, e.g., [22]). It is thus very likely that all searches will go through the same potentially malicious nodes, thus reducing the benefits of performing redundant searches. In fact, we will compare our approach to both the regular locate operation in Chord and the naive redundant approach.

3 High-Assurance LOcate (Halo) Protocol

In this section, we present a new approach for searching in p2p networks that significantly eliminates the problems due to malicious redirections in the resource-location process. Our method uses ℓ-redundancy, but—crucially against the adversary's success probability—it does so in a more elaborate way that creates ℓ independent searches that are disjoint with a high probability and thus have very low correlated rates of failure. We show that our searching method is a probabilistic algorithm that searches in a p2p network of size n and terminates with a claimed owner node that is correct with a very high probability, specifically with probability $1 - 0.25^{\ell}$, where $\ell \leq c \log n$. We demonstrate our technique focusing on Chord.

3.1 Halo applied to Chord

Given a search key k and parameterized by the desired redundancy ℓ, our searching technique, Halo, performs a composite search over the underlying DHT to realize ℓ disjoint and independent searches for k using only the primitive locate operation (provided by the DHT). Halo relies on a very simple, yet powerful idea: although the underlying routing graph in Chord is actually defined in a probabilistic manner, there is still a strong deterministic component in it. Namely, finger

Algorithm 1 The HA_locate Algorithm

Parameter: redundancy ℓ;
Input: a search key k, a subset of known nodes $f_1, \ldots, f_\ell$;
Output: an IP address, the owner of key k;

```
1: $C = \{\}$        /*initialization of set of candidate knuckles
                     /*perform k knuckle locations
2: for $i = 1$ to $\ell$ do
3:    $k_i \leftarrow k - 2^{m-i}$          /*compute exponential offset.
4:    $p_i \leftarrow$ locate$(k_i, f_i)$          /*locate possible knuckle as predecessor of $k_i$
5:    $t_i \leftarrow$ get_finger(SHA-1$(p_i), i)$
                     /*test knuckle correctness
6:    if $k$ is clockwise closer from SHA-1$(t_i)$ than $k_i$ is then
7:       $p_i \leftarrow$ get_successor$(k_i)$       /*improve estimate of correct knuckle
8:       $t_i \leftarrow$ get_finger(SHA-1$(p_i), i)$
9:    end if
10:   $C \leftarrow C \cup$ SHA-1$(t_i)$         /*add possible successor node in candidate list
11: end for
12: $t \leftarrow c_i \in C : c_i$ is clockwise closest from $k$   /*decide on the target owner
13: return $t$
```

nodes are defined by considering exponentially different offsets in the logical ring. Our approach is to try to exploit exactly this regularity in predicting a set of very important nodes for a target node v (the owner of a given resource): the nodes for which v is a finger node, or to preserve the analogy, the *knuckle* nodes of v.

Why are knuckles of any importance? Exactly because these nodes contain direct routing information for their finger nodes; therefore, the knuckle nodes of the target node v of any search all contain v's IP address in their routing tables. By the symmetry between a finger and a knuckle and by the routing properties of Chord, we expect that any node in Chord has on average $O(\log n)$ knuckles, all located in exponentially different distances from that node. Accordingly, given a target resource identifier r that maps to position (key) $k = f(r)$ in the ring and that is owned by target node v, if we could (deterministically) compute the $O(\log n)$ on average corresponding knuckle nodes of v, we could then directly contact these knuckles and ask for the appropriate entry in their routing tables. Ideally, if no malicious nodes are in place, these entries would all agree, being equal to v. But, it may be the case that one or more knuckle nodes are actually malicious, reporting incorrect routing information. We can still correctly decide on the correct target v, however, by simply choosing the node (IP address) falling in the clockwise-minimum position in the logical ring (with respect to the search key)—as long as there exists at least one honest knuckle node, we guarantee a correct search! To see why, recall that the malicious coalition returns the first malicious successor node v^* of the key k. If the target v is not malicious, it would necessarily be the case that $f(v) < f(v^*)$ and at least one honest knuckle would report v; otherwise, v is already the correct but malicious owner of resource r. Note that in this paper we assume that the Chord structure is consistent if there is no malicious activity; for example, our analysis does not capture routing-table inconsistencies due to dynamism and transient effects.

But how can we find the knuckle nodes? The idea here is quite intuitive: Halo uses the exponential steps that define the fingers nodes but *in reverse*. In particular, consider the case where we are searching for resource r, mapped to key $k = f(r)$, and the set P_k consisting of the $m = \log q$ positions in the ring that clockwise-precede k, that is, set $P_k = \{k - 2^0, k - 2^1, \ldots, k - 2^{m-1}\}$, $m = \log q$. When searching for key k, Halo deterministically computes the ℓ "most significant" positions in P_k, that is, positions $k - 2^{m-1}, k - 2^{m-2}, \ldots, k - 2^{m-\ell}$, as keys to search for the knuckles of k. Our searching technique then uses the following heuristic: the i-th knuckle node of v is approximated by the predecessor node (or sometimes the successor) of position $k - 2^{m-i}$. These ℓ candidate knuckle nodes can be found using our primitive resource-ownership operation: the regular locate operations applied for the ℓ computed "knuckle" positions. As we will discuss later, finding a correct knuckle node will succeed with probability .75, under reasonable assumptions for the Chord structure. The failure probability depends on the original placement of the network nodes on the Chord ring (through function $g(\cdot)$, the SHA-1 function). Thus, overall our approach amounts to successively (or in parallel, actually) locating these ℓ candidate knuckle nodes. These searches are easily seen to be more widely distributed over the Chord ring, making the effect of adversarial redirections less significant for the correctness of the search operation.

Note that by the construction of Chord, the $m = \log q$ positions in the ring that are defined by considering exponentially different offsets, are owned by $O(\log n)$ distinct candidate knuckles, which actually correspond to the positions of the longest offsets. This is why Halo searches exactly the ℓ most significant candidate knuckles. Also, because there are at most $O(\log n)$ candidate knuckles per each target node, we already have an upper bound on the redundancy parameter, i.e., $\ell \leq c \log n$, for some constant c. Furthermore, to ensure that each search is disjoint, in general the searches are started from different finger nodes of the initiator node, again limiting ℓ to $O(\log n)$. Figure 2 demonstrates this approach for $\ell = 2$.

Algorithm 1 presents our formalized *high-assurance* search algorithm, $\mathsf{HA\_locate}$, for locating resources in Chord. We use the easy-to-implement (if not already existing in a DHT implementation) $\mathsf{get\_finger}(u, i)$ operation that returns the i-th finger (IP address) of the network node mapped to position u (i-th entry of the routing table of node v, $u = g(v)$). Parameterized by the redundancy parameter ℓ, algorithm $\mathsf{HA\_locate}$ takes as input a key k (the position $f(r)$ in the ring of a target resource r) and also ℓ known nodes in the p2p network, which are used as starting points for the ℓ redundant searches, and outputs the owner of k. The primitive search operation locate takes two inputs: the search key (ID in logical space) and the node (IP address) initiating the search and outputs the predecessor (IP address) of the owner of the key. We assume that locate returns the predecessor node of a key because this is how it is implemented in Chord (Chord performs a $\mathsf{get\_successor}(i)$ operation on the predecessor to find the successor of a key). To increase the effectiveness (disjointness) of the ℓ redundant searches, we initiate the knuckle searches from distinct nodes. Note that $\ell \leq \log n$, thus there are always ℓ known nodes for any procedure executing our high-assurance composite search: either a node in the system is running the search in which case it already knows $O(\log n)$ nodes in the network (the ones in its routing table), or a node outside the network is running the search in which case we can simply assume the existence of $O(\log n)$ default known nodes of the system used for this purpose (in this case, the search can alternatively be forwarded to a random network node). We also use the primitive operations $\mathsf{get\_successor}(u)$ returning the owner of position u (i.e., the IP address of the successor node of position u). Finally, in our algorithm and our analysis, we use the following (rather intuitive) notation: (i) we say that position (or node) u_1 is *clockwise closer from* u than u_2 *is*, if, when starting from u and moving around the ring in the clockwise direction, we meet u_1 before u_2; and (ii) we say that position u *falls between* u_1 and u_2 in the clockwise direction, if, when staring from u_1 and moving clockwise we meet u before we meet u_2 (or equivalently, if u is clockwise closer from u_1 than u_2 is).

3.2 Correctness

In what follows we analyze Algorithm 1 for performing a high-assurance search over Chord, in terms of correctness and efficiency, and the improvement it provides over regular Chord searches with respect to the disjointness of the redundant searches. Note that the security properties of Halo rely solely on implementing exactly these redundant searches in a way such that the set of accessed nodes is distributed as uniformly as possibly over the set of participating nodes. Also note that to achieve this uniformity of accessed nodes, Halo solely relies on the prediction of the knuckle nodes of a given target. Thus, in our analysis we focus on this knuckle prediction and also we do not consider any malicious behavior from the participating nodes. For our analysis we refer to Figure 3.

Let k be the position of the target resource identifier that we want to locate and $s(k)$, $p(k)$ be respectively its successor and predecessor nodes. That is, $s(k)$ is the owner of k. Consider the i-th iteration of the algorithm, where we seek the i-th knuckle of $s(k)$. Let k' be the position corresponding to the i-th knuckle of k that is computed deterministically using offset $s_i = 2^{m-i}$ (step 3 in the algorithm), and let $p(k')$, $s(k')$ be respectively its predecessor and successor nodes. Let us examine how well $\mathsf{HA\_locate}$ performs by using the heuristic that the i-th knuckle node of k is what operation $\mathsf{locate}(k')$ returns, i.e., its predecessor $p'(k)$. Recall that the i-th knuckle node of k is a node that stores in the i-th position in its routing table the owner node of k.

Let d_1 be the distance in the logical ring between k and $p(k)$, d_2 be the distance between k and $s(k)$. Similarly, let d'_1, d'_2 be the distances between k' and $p(k')$ and $s(k')$ respectively. We consider two cases: Case I.1: if $d_1 > d'_1$, then $p(k')$ is indeed the i-th knuckle of k (because $p(k)$ falls between $p(k')$ and $p(k') + s_i$ in the clockwise direction, making $s(k)$ the i-th finger of

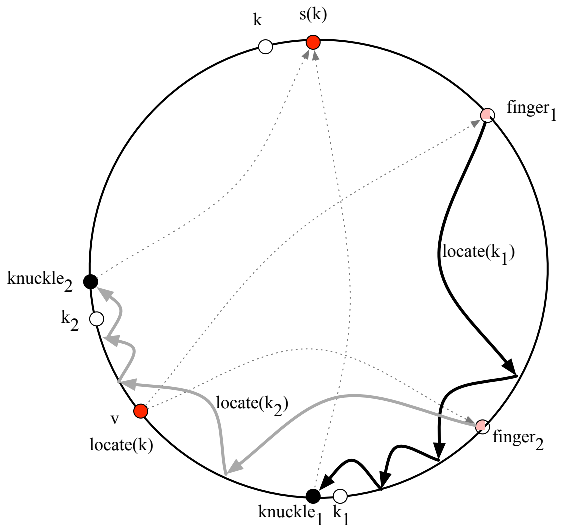


Figure 2. The key k for the locate operation is used to compute knuckles keys k_1 and k_2. Node v initiates two separate locate operations for k_1 and k_2 starting from fingers finger_1 and finger_2. These redundant searches locate the knuckles $\mathrm{knuckle}_1$ and $\mathrm{knuckle}_2$, whose routing tables contain the successor node and owner $s(k)$ of k.

$p(k')$; see Figure 3(a)) and our heuristic is correct. Otherwise, we have Case I.2, and the heuristic (initially) fails (because $p(k')$ is between $p(k') + s_i$ and k in the clockwise direction, making the i-th finger of $p(k')$ some node other than $s(k)$; see Figure 3(b)). But, at step 5 of the algorithm, we explicitly perform this test and identify whether we have a successful prediction for the i-th knuckle; indeed, we fail whenever $d_1 \leq d_1'$, in which case the i-th finger t_i of $p(k')$ is certainly not the owner of k, falling clockwise-before the k, which can be tested as in step 5.

In steps 6 and 7, however,our algorithm tries to rectify this false prediction, by trying the i-th finger of the successor $s(k')$ of k'. This new prediction turns out to be correct whenever $d_2 > d_2'$ (Case II.1). This is because, if $d_2 > d_2'$ then $s(k') + s_i$ is between k and $s(k)$ in the clockwise direction Otherwise, if $d_2 \leq d_2'$, we have Case II.2, and our i-knuckle prediction is incorrect. But, as we prove next, this is an *inherent property* of the underlying graph of Chord, not of our algorithm.

Proposition 1. *If $d_1 \leq d_1'$ and $d_2 \leq d_2'$, then there does not exist an i-th knuckle for the target node $s(k)$. This condition arises with probability 0.25.*

Proof. (Sketch.) The i-th fingers of nodes $p(k')$ and $s(k')$ are respectively $p(k)$ or a node at a clockwise-earlier position in the ring and a node at a clockwise-later position than $s(k)$. Since $p(k')$ and $s(k')$ are adjacent nodes, there cannot exist a node whose i-th finger is $s(k)$, since that node must be between $p(k')$ and $s(k')$.

Since nodes are uniformly distributed (and assuming that all four nodes are distinct), the probability that Case I.2 arises ($d_1 \leq d_1'$) is 0.5 (comparing the lengths of two randomly chosen segments). Applying the same argument to Case II.1, we have that the probability that both cases arise simultaneously is 0.25. Cases I.2 and II.1 are independent because the distances d_1, d_1', d_2, d_2' are independent and identically distributed uniform random variables. $\square$

Therefore, if, during the i-th iteration and for the Chord p2p network, $d_1 \leq d_1'$ and $d_2 \leq d_2'$, then our searching technique would not correctly predict the i-th knuckle (because it simply *does not exist*). The probability with which the entire algorithm fails is, therefore, 0.25^ℓ, which arises when none of the ℓ predicted knuckles are found. We note that a high-assurance locate operation as shown in this algorithm is used in conjunction with a regular Chord locate operation, thereby ensuring that Halo performs no worse than Chord.

Overall, at step 11 our algorithm computes and returns the i-th finger of the predicted i-th knuckles of the target node $s(k)$ that is clockwise-closest to the target position k. As we have argued earlier, this guar-

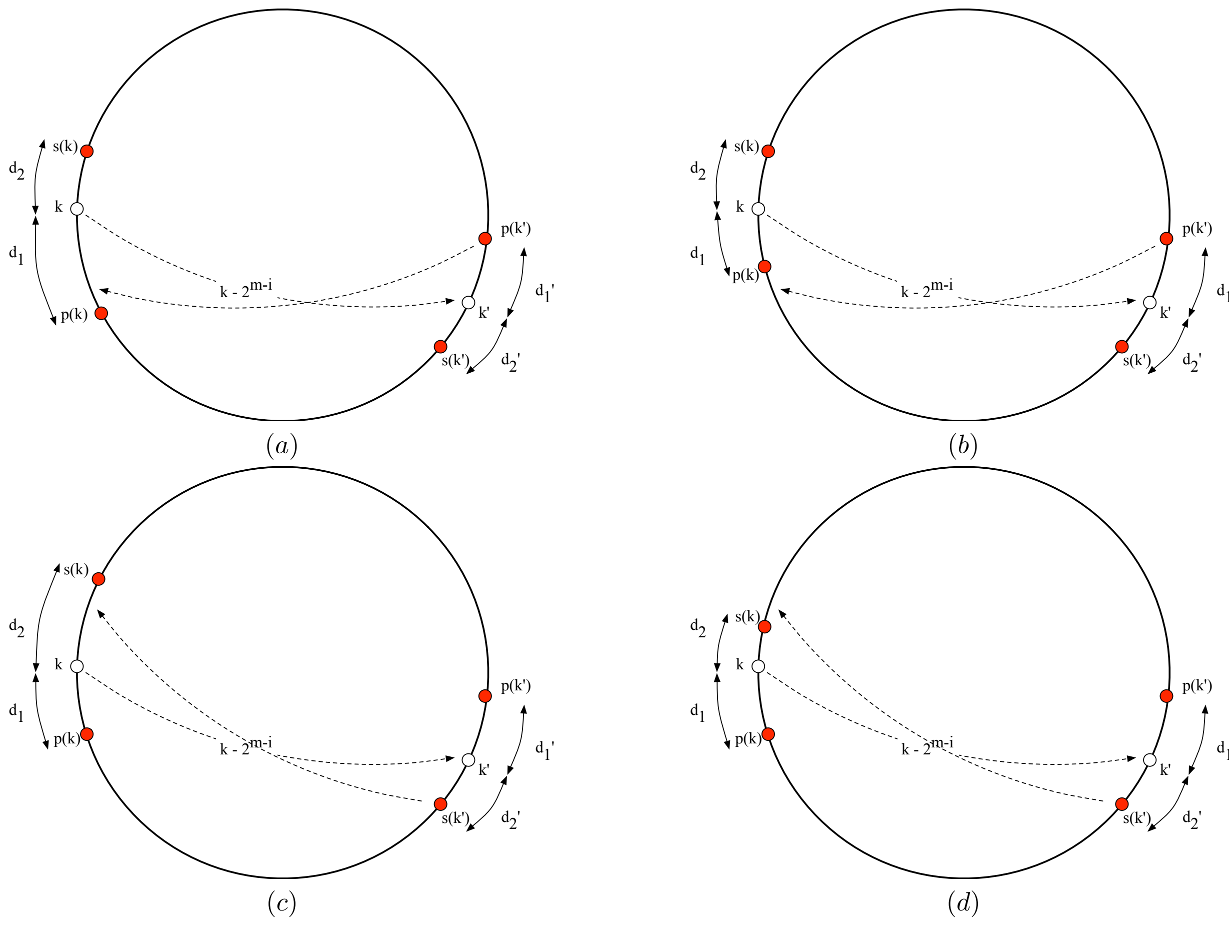


Figure 3. (a) **Case I.1:** $p(k')$ **is the correct knuckle of** $s(k)$ **if** $d_1 > d_1'$**.** (b) **Case I.2:** $p(k')$ **is not the correct knuckle of** $s(k)$ **if** $d_1 \le d_1'$**; then, try** $s(k')$ **instead and see case II.** (c) **Case II.1:** $s(k')$ **is the correct knuckle of** $s(k)$ **if** $d_2 > d_2'$**.** (d) **Case II.2:** $s(k')$ **is not the correct knuckle of** $s(k)$ **if** $d_2 \le d_2'$**.**

antees that our algorithm will return the *correct target node* $s(k)$ as long as there exists *at least* one successful prediction for a knuckle of $s(k)$ that is found through a search path in the ring that does not intersect the malicious coalition. Thus, our searching algorithm always returns an output that with some high probability is correct. In the next section we analyze our algorithm and estimate the (as we show small) failure probability of our searching technique in the face of attack, showing that Halo is indeed a *high-assurance* approach.

3.3 Recursive Halo

Algorithm 1 implements an ℓ-redundant search over a DHT by predicting and locating network nodes that are likely to have direct routing information for the target node. In contrast to the straightforward approach that performs ℓ location operations starting from different nodes, our ℓ redundant searches have significant less correlation and therefore can better tolerate adversarial redirections. These ℓ searches themselves, however, have a high rate of failure since they are based on a regular Chord locate operation.

Accordingly, we can extend our knuckle-based high-assurance locate by applying Algorithm 1 recursively to locate with high-assurance the candidate knuckle nodes. That is, the idea is to replace at the step 4 in the algorithm the operation $p_i \leftarrow \mathsf{locate}(k_i, f_i)$ with operation $p_i \leftarrow \mathsf{HA\_locate}(k_i, f_1, \ldots, f_{\ell_2})$, where $f_1, \ldots, f_{\ell_2}$ are the ℓ_2-most significant fingers of the node initiating the search; ℓ_2 is the recursive-redundancy parameter. With this approach, recursive Halo succeeds in locating the correct knuckles with higher probability than the non-recursive version, since a knuckle is located using

HA_locate instead of locate, leading to higher success rates in finding the target node. In the next sections we experimentally demonstrate the power of this idea for degree-2 recursion by getting significantly better results and achieving higher assurance. Note that although the node calling our algorithm can contact $O(\log n)$ nodes in total, these can be used more than once since for every level of the recursion a search for a different target node can be chosen.

3.4 Generalization

Although demonstrated for Chord, we believe that our high-assurance knuckle-based searching techniques can be generalized to other DHTs as long as the following conditions are met:

1. The underlying graph representing sharing of direct routing information between nodes allows the successful prediction of the $O(\log n)$ knuckle nodes of the target node. In Chord, the exponentially different offsets for defining fingers allow this prediction.

2. The ownership relation is well-defined according to some distance metric that allows successful and secure selection of the correct target node from a set of claimed target nodes. In Chord, the "clockwise closest" relation is used.

3. Network nodes are mapped to the logical identifier space in a random uniform-like way, i.e., it is infeasible for malicious colluders to control their location in the logical identifier space. In Chord, the use of SHA-1 ensures this property.

We believe that these properties, or similar properties that make our Halo technique applicable, are satisfied by most DHTs, for instance, DHTs that use the hypercube routing method (e.g., Pastry [28] and Tapestry [36] using the technique in [24]), DHTs that partition a d-dimensional space hierarchically into zones (e.g., CAN [25]), or DHTs that use a tree-like hierarchy (e.g., [21, 22]. In the future, we hope to demonstrate the effectiveness of Halo for some of these DHTs.

3.5 Bootstrapping and Join/Leave operations

All dynamic operations for Chord (e.g., for joining or leaving the network) employ the use of the primitive locate operation. Accordingly, we can achieve high-assurance dynamic operations by using our high-assurance locate operation. The only requirement is that a joining node runs the augmented join operation using a predefined set of $O(\log n)$ publicly known and trusted peer nodes. This condition is easy to achieve, since a short list of such nodes can be made available, for instance on a web page. These nodes are used only as starting points for join and leave operations.

4 Analysis

We develop an analytical approximation of the expected failure rates for a regular Chord locate(k) operation and its high-assurance counterpart. We will show how this analytical approximation fits our simulation data closely. Recall that we have defined locate(k) as the locate_successor(k) operation (it returns the owner node of k), although in the real Chord implementation locate(k) is actually implemented using the locate_predecessor(k) operation (it returns the predecessor of the owner) followed by the find_successor operation. In what follows we use this fact. HA_locate is our operation described in Algorithm 1; Halo search with ℓ-redundancy makes use of a regular Chord locate_successor(k) operation followed by HA_locate with redundancy parameter $\ell - 1$.

4.1 Chord

Let $X = P[\text{locate\_predecessor}(k) \text{ fails}]$, where X is the probability with which a locate_predecessor(k) operation fails in regular Chord. The locate operation fails if the successor returned by the locate operation is not the true successor of k. X is a random variable because this failure probability depends on the number of nodes traversed during the locate operation, which is a random variable. Let K be the number of nodes traversed in a locate_predecessor(k) operation. We know that K is binomially distributed with parameters $(\log n, \frac{1}{2})$ and has mean $\frac{1}{2}\log n$. If c is the fraction of malicious nodes in the network, then we have that a given node is malicious with probability c (since nodes are mapped onto the Chord ring randomly, adversaries cannot control their location in the logical ring). The probability that all the nodes traversed by a locate_predecessor(k) operation are non-malicious is $(1-c)^K$, and therefore the probability that at least one node is malicious is

$$X = 1 - (1-c)^K.$$

We approximate the expectation of this probability as $E[X] \approx 1 - (1-c)^{\frac{1}{2}\log n}$. We observe (as in Artigas et al. [1]) that $E[X]$ is bounded by this approximation. Since we assume that the target successor is not malicious in our simulations (we aim to assess how many potentially successful searches are subverted by malicious nodes), $E[X]$ is also the expected probability of success of a $\mathsf{locate\_successor}(k)$ operation.

4.2 High-assurance search

Now, let $Y = P[\mathsf{HA\_locate}(k) \text{ fails}]$, where $\mathsf{HA\_locate}(k)$ involves one iteration of our proposed algorithm. This operation succeeds if a regular Chord $\mathsf{locate\_predecessor}(k')$ succeeds for the knuckle $p(k')$ with estimated key k'. However, this search will succeed only if Case I.1 holds (see Figure $3(a)$ in Appendix 3.2). Note that since nodes are uniformly distributed (and assuming that all four nodes are distinct), this probability is 0.5 (comparing lengths of two randomly chosen segments). Now if Case I.2 applies (with probability 0.5), then it is possible that the successor of this node $s(k')$ is a knuckle for k. Applying the same argument to Case II.1 (Figure $3(c)$ in Appendix 3.2), $s(k')$ will contain $s(k)$ with probability $(1-c)0.5$ (given that Case I.2 applies) since we must also have that $s(k')$ is not malicious. This gives us the overall success probability of a $\mathsf{HA\_locate}(k)$ operation as

$$(1-c)^K(0.5 + 0.5(1-c)0.5)$$

and we have that

$$E[Y] \approx 1 - \left\{(1-c)^{\frac{1}{2}\log n}(0.5 + (1-c)0.25)\right\}.$$

Assuming that $\ell - 1$ redundant searches are disjoint, the expected probability that $\ell - 1$ $\mathsf{HA\_locate(k)}$ operations fail is approximately

$$\left(1 - \left\{(1-c)^{\frac{1}{2}\log n}(0.5 + (1-c)0.25)\right\}\right)^{\ell-1}.$$

Since we perform one regular Chord search with $\ell - 1$ $\mathsf{HA\_locate(k)}$ operations for a redundancy parameter of ℓ, we get an overall expected failure probability of our Halo search with redundancy parameter ℓ is

$$E[Z] \approx \left(1 - \left\{(1-c)^{\frac{1}{2}\log n}(0.5 + (1-c)0.25)\right\}\right)^{\ell-1} E[X]$$

where Z is the probability that Halo search fails.

We describe our simulations in the next section, and the interested reader may refer to Figures 6 and 7 in the Appendix to observe the closeness of fit for this analytical model for a network of 1,000 and 10,000 nodes respectively. In summary, the model fits the data fairly accurately as long as the disjoint-path assumption holds (making the events that the individual search paths fail independent). For higher values of redundancy (7 for 1,000 nodes and 13 for 10,000 nodes), the probability that two separate locate operations share a same node increases, and our model underestimates the probability of failure.

5 Experiments

We evaluate our approach for high-assurance search in the context of Chord. We simulate various adversarial environments, and show how different levels of redundancy can be used to attain security in these situations. First we describe our simulation setup, and then present our simulation results.

5.1 Simulation setup

We built our own simulator for Chord using the Java programming language. This simulator models routing in Chord, including adversarial rerouting of locate requests, and does not model network dynamics such as join and leave operations. Our simulation takes the parameters $\langle n, c, \ell_1, \ell_2, i, j \rangle$ as input and does the following:

It creates n nodes in the Chord network, and randomly marks cn of these nodes as malicious. Routing tables are constructed based on Chord's algorithm, however, malicious nodes communicate within themselves and subvert searches by reporting the closest *malicious* predecessor for a search key instead of the closest legitimate predecessor. The simulation instantiates i different Chord networks, and within each network simulates j random locate queries. Each locate query originates in a randomly chosen start node, for a randomly chosen key k such that both the start node and successor of k are not malicious. Each Halo search is performed with redundancy parameters ℓ_1 and ℓ_2. We vary the colluding fraction c from 0 to 0.3. We believe that for values of $c > 0.3$, real successors of keys are malicious with a high probability (equal to c) and improving the success rate of a potentially-successful search has little meaning even if this failure rate is close to zero. We note that Nambiar and Wright [22] use the same reasoning to simulate failure rates for $c \leq 0.2$.

Each point in our simulation graphs corresponds to the average failure rate of searches across the $i = 100$ simulated Chord networks, where the failure probability for each instantiation of a Chord network is the average failure rate of $j = 1000$ searches. The error bars correspond to one standard deviation.

5.2 Simulation results

In interpreting our results, we say that a search for a key's successor is *secure* if at most 1% of searches for honest successors fail. As we will see, for varying levels of malicious nodes, the level of redundancy can be increased to provide the requisite security.

Figure 6 in the Appendix shows the performance of Halo search and the closeness of fit of the predicted probabilities for 1,000 nodes (Likewise, Figure 7 in the Appendix for 10,000 nodes). The graphs show that Halo search vastly outperforms Chord in locating nodes, and is much better than the naive redundant search in Chord. Depending on the level of security required, users can pick the appropriate redundancy. For example, in a network with 10,000 nodes, a redundancy of 3–5 may be used for security against 0–5% colluders. A redundancy of 7–13 could be used for security against 5–12% colluders. For these levels of redundancy the probability of a failed search (assuming an honest target) is approximately 1%. Similar levels of security are obtained for a network of 1,000 nodes, except that the redundancy is limited to 10 redundant searches. Figures 4(a) and 4(b) summarize the results for three different Halo searches for 1,000 and 10,000 nodes respectively.

Next, we study the effect of recursive redundant searches, and observe in Figures 5(a) and 5(b) that security is achieved for much higher numbers of adversaries in the network. In particular, recursive Halo search is secure for up to 22% malicious nodes, with only 2–3% searches failing for 25% colluding nodes.

5.3 Comparison with other approaches

It is certainly possible to get better results with more storage at the nodes. More storage equates to shorter search paths, and a lower probability that a search is subverted by a malicious node. For example, Salsa [22] divides n nodes into G groups. All nodes maintain information about all other nodes in the same group, resulting in $\frac{n}{G}$ entries in the routing table. Furthermore, each node maintains one contact for $\log G$ other groups, resulting in $O(\frac{n}{G} + \log G)$ storage at each node. A search in Salsa has length $O(\log G)$, which is considerably smaller than $O(\log n)$. Therefore comparing Salsa with our scheme would be unfair.

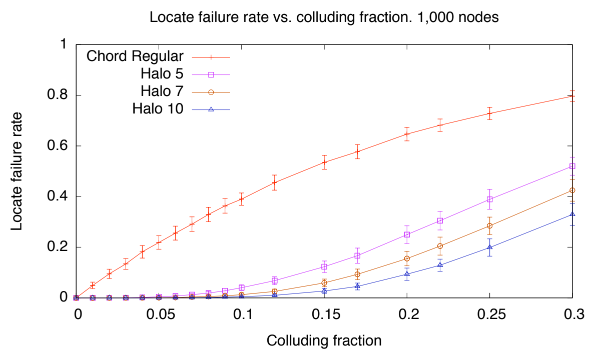
Similar to Salsa (although we note that Cyclone predates Salsa), Cyclone [1] subdivides a Chord network into ℓ smaller networks, resulting in $O(\ell + \log \frac{n}{\ell})$ storage at each node, which results in much larger storage than in our scheme (and more storage than in Salsa). Search paths are of length $O(\log \frac{n}{\ell})$, and therefore comparable to Salsa. Again, we do not believe that Cyclone can be compared fairly with our scheme because of its increased storage. In fact, our Halo search can be applied to each Chord subnetwork of Cyclone to further improve Cyclone's performance without impacting Cyclone's storage requirements.

In short, we provide a novel technique to perform redundant searches in Chord *without* requiring any extra storage at the nodes, and more specifically $O(\log n)$ storage (lower than Salsa and Cyclone). Indeed, we don't require *any* changes in existing Chord nodes, and our Halo search can be initiated by any nodes with the augmented search algorithm, making use of existing Chord constructs at the other nodes. As a result, the storage complexity of Chord is not affected, and the length of each redundant search remains $O(\log n)$.

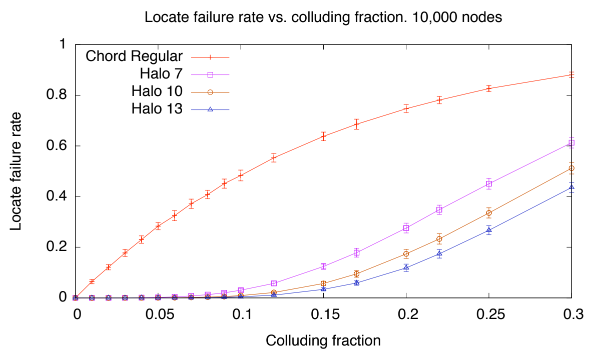
6 Related Work

There is a large and growing literature on p2p overlay networks. One popular class of overlay networks is that of distributed hash tables (DHTs). These structures make use of consistent hashing to efficiently support queries for exact matches with data keys. Examples of distributed hash tables include Chord [13, 33], Koorde [18], Pastry [28], Scribe [29], Symphony [20], and Tapestry [36] and others (e.g., [24, 25]), with Chord being one of the most representative and most studied DHTs. As an example of performance, Chord, in its original form, supports queries using $O(\log n)$ messages and $O(\log n)$ words of memory corresponding to the $O(\log n)$ degree of the underlying graph. On top of these DHTs many distributed systems have been built that are supporting a wide-range of real-life applications (e.g., PAST [8], CAN [25], CFS [5], PIER [17], OpenDHT [27]). Also, other p2p architectures with similar efficiency provide more elaborate functionality over p2p networks; for instance, skip-graphs [2] and their extensions, e.g., [16] and [14], support searches over ordered sets of resources.

A large set of security issues have been studied in p2p systems and DHTs. General issues are considered in [31, 35]. In Castro *et al.* [4] the first schemes for battling adversarial behavior in routing are given. The scheme used $O(\log n)$ messages per query in the absence of faulty behavior, and was resilient to limited adversarial attacks. However, the scheme makes use of an external certification authority (CA) to provide verifiable random ID values to network addresses. Moreover, the system was not robust to certain types of at-

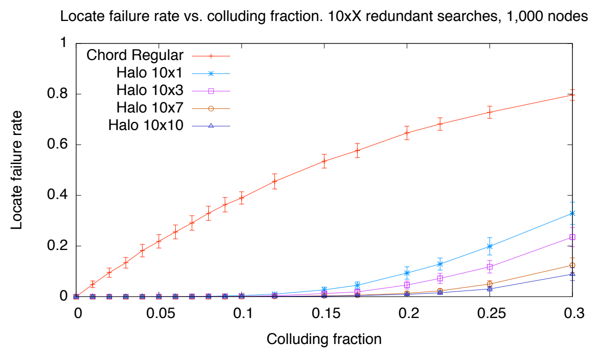


(a) Locate failure rate vs. colluding fraction for a 1,000 node network

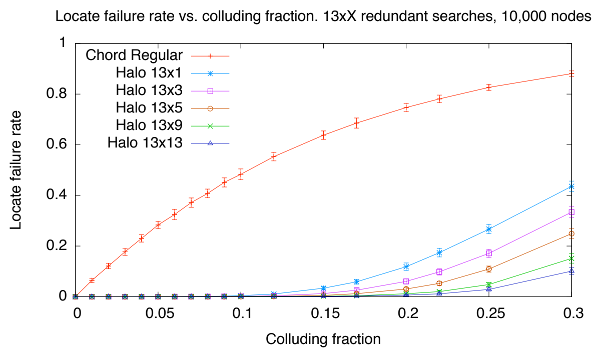


(b) Locate failure rate vs. colluding fraction for a 10,000 node network

Figure 4. Comparing various levels of redundancy for Halo searches. We can see that the failure rates are negligible for up to 12% colluding nodes.



(a) Locate failure rate vs. colluding fraction for a 1,000 node network



(b) Locate failure rate vs. colluding fraction for a 10,000 node network

Figure 5. Comparing various levels of recursion for Halo searches. We can see that the failure rates can be made negligible for up to 22% colluding nodes. Only 2–3% searches fail for 25% colluding nodes, and 10% fail for 30% colluding nodes.

tacks by the adversary, in particular the Sybil attacks of Douceur [7], in which the adversary acquires numerous ID values which it uses to obtain a concentrated presence in one portion of the network.

With respect to routing and searching, numerous DHTs have been shown to tolerate significant network-node failures—random (e.g., [18, 25, 28, 32, 36]) or malicious (e.g., [3, 9, 10, 19, 23, 30]). The structures that deter adversarial behavior do so either by augmenting the DHT connectivity structure and communication complexity of the routing algorithms by at least a logarithmic factor (e.g., [9, 10, 23, 30]) or by using assumptions about external trusted parties (e.g., supervisor in [19]). Other schemes (e.g., [3, 10]) achieve security properties by forming suitably large and random blocks of machines that take the place of each individual machine in the data structure. A majority-voting scheme is then used to prevent faulty behavior from adversarial nodes that are the minority. All of these majority-voting schemes incur also an increase of at least a logarithmic size in communication overhead.

With respect to data authentication and content integrity, most p2p systems (e.g., [5, 8, 24, 25, 27]) support an elementary authentication service for retrieved data using individual signatures on the stored data objects. For the static case, storage authentication often involves the so-called *self-certified data* [12], where large data items (e.g., a file system) get partitioned into blocks, which are stored as separate objects in the system and are bound together using collision-resistant hashing in some tree-like hierarchy, and where the root-block is signed. For the dynamic case, a recent technique for distributed data authentication [34] can be used, where dynamic data sets stored in p2p networks can be efficiently authenticated. Over any DHT and using only the location search operation, a distributed version of Merkle tree is realized, and using this in a network with n nodes, it is showed how to efficiently authenticate content membership in a fully dynamic set of m data elements in $O(\log n \log m)$ time using $O(m \log m)$ storage, with similar amortized complexities for supporting insertions and deletions. This technique however cannot be used to achieve authentication of routing information, since routing information, in contrast to data resources is collectively computed and cannot be signed by a single entity. Finally, privacy and anonymity issues (e.g., [11, 22, 26]) or other security issues (e.g., the Sybil attack [6, 7]) related to p2p systems have been studied. We have discussed Cyclone [1] and Salsa [22] in Section 5.3.

7 Conclusion

We presented a novel scheme called Halo for performing disjoint redundant searches in DHTs such as Chord. Instead of performing multiple redundant searches directed towards a target node, Halo searches for the "knuckles" of the target node. These knuckles contain the target node in their routing tables, and are spread over the DHT such that searches for these knuckles are disjoint with high probability. We showed the effectiveness of our approach by presenting both analytical models and a simulation-based evaluation of Halo. We found that our scheme can significantly increase the integrity of searches, by allowing only 1% of searches to be subverted by up to 22% malicious colluding nodes in the network.

As future work, we plan to further study our knuckle-based high-assurance search, exploring its theoretical bounds and the power of t-depth recursion, as well as to apply our technique to other DHT structures.

Acknowledgments

This research program is primarily a part of the Institute for Security Technology Studies, supported by the U.S. Bureau of Justice Assistance under grant 2005-DD-BX-1091, and by the Institute for Information Infrastructure Protection (I3P) under an award from the Science and Technology Directorate at the U.S. Department of Homeland Security. Research was additionally supported by the Center for Algorithmic Game Theory at the University of Aarhus, Denmark, under an award from the Carlsberg Foundation. The views and conclusions in this paper do not necessarily reflect the views of the sponsors. Simulations were performed on cluster machines supported under NSF grant EIA-98-02068.

References

[1] M. S. Artigas, P. G. Lopez, J. P. Ahullo, and A. F. G. Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 49–56, Washington, DC, USA, 2005. IEEE Computer Society.

[2] J. Aspnes and G. Shah. Skip graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.

[3] B. Awerbuch and C. Scheideler. Towards a scalable and robust dht. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms*

and architectures, pages 318–327, New York, NY, USA, 2006. ACM.

[4] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of Usenix Symposium of Operating Systems Design and Implementation (OSDI)*, 2002.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[6] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson. Sybil-resistant DHT routing. In *Proceedings of the 10th European Symposium On Research In Computer Security*, Milan, Italy, September 2005.

[7] J. R. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 251–260, 2002.

[8] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HOTOS '01: Proceedings of Eighth Workshop on Hot Topics in Operating Systems*, page 75, Washington, DC, USA, 2001. IEEE Computer Society.

[9] A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proceedings of Symposium on Discrete Algorithms*, 2002.

[10] A. Fiat, J. Saia, and M. Young. Making chord robust to byzantine attacks. In *Proceeding of European Symposium of Algorithms*, pages 803–814, 2005.

[11] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 193–206, New York, NY, USA, 2002. ACM Press.

[12] K. Fu, M. F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.

[13] P. Ganesan and G. S. Manku. Optimal routing in Chord. In *Proceedings of 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 169–178, 2004.

[14] M. T. Goodrich, M. J. Nelson, and J. Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393, New York, NY, USA, 2006. ACM Press.

[15] C. Harvesf and D. M. Blough. The effect of replica placement on routing robustness in distributed hash tables. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 57–6, Washington, DC, USA, 2006. IEEE Computer Society.

[16] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, Lecture Notes in Computer Science, 2003.

[17] R. Huebsch, B. Chun, J. Hellerstein, B. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *Proceedings of 2nd Conference on Innovative Data Systems Research (CIDR)*, pages 28–43, 2005.

[18] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[19] K. Kothapalli and C. Scheideler. Supervised peer-to-peer systems. In *Proceedings of 2005 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, 2005.

[20] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[21] P. Maymounkov and D. Mazires. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 53–65, Mar. 2002.

[22] A. Nambiar and M. Wright. Salsa: a structured approach to large-scale anonymity. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 17–26, New York, NY, USA, 2006. ACM Press.

[23] M. Naor and U. Wieder. Novel architectures for p2p applications: The continuous-discrete approach. *ACM Trans. Algorithms*, 3(3):34, 2007.

[24] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[25] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, pages 161–172, 2001.

[26] M. Rennhard and B. Plattner. Practical anonymity for the masses with MorphMix. In *Proceedings of Financial Cryptography*, 2004.

[27] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of 2005 ACM SIGCOMM Conference*, pages 73–84, 2005.

[28] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329, 2001.

[29] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[30] J. Saia, A. Fiat, S. D. Gribble, A. R. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 270–279, London, UK, 2002. Springer-Verlag.

[31] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of International Workshop on P2P Systems*, pages 261–269, 2002.

[32] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[33] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM '01*, pages 149–160, San Diego, California, August 2001.

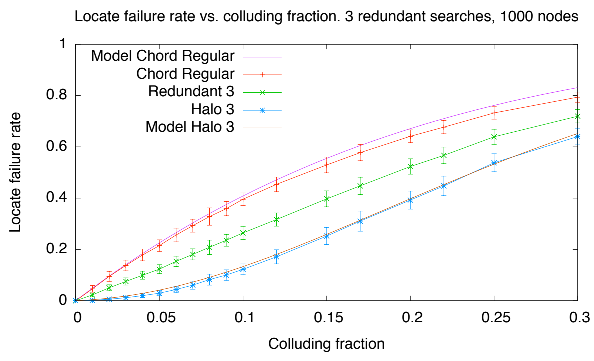
[34] R. Tamassia and N. Triandopoulos. Efficient content authentication in peer-to-peer networks. In *Proceedings of Applied Cryptography and Network Security*, pages 354–372, 2007.

[35] D. S. Wallach. A survey of peer-to-peer security issues. In *Proceedings of International Symposium on Software Security*, 2002.

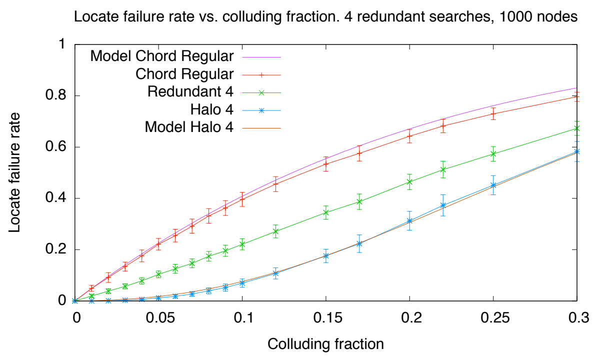
[36] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.

A Additional figures

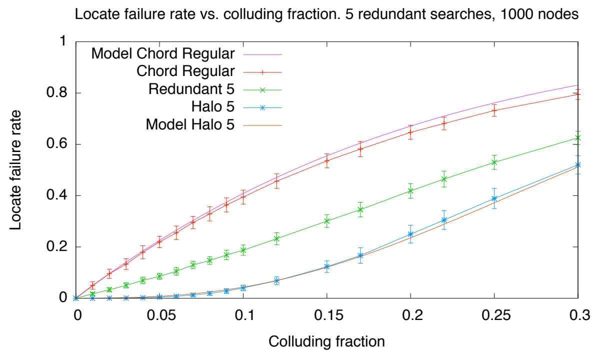
Please refer to the next page.



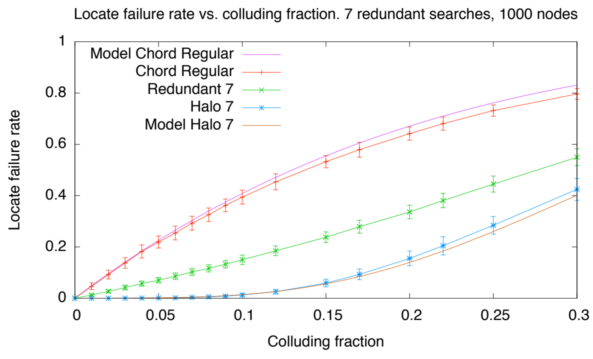
(a) Comparing failure rates for redundancy = 3



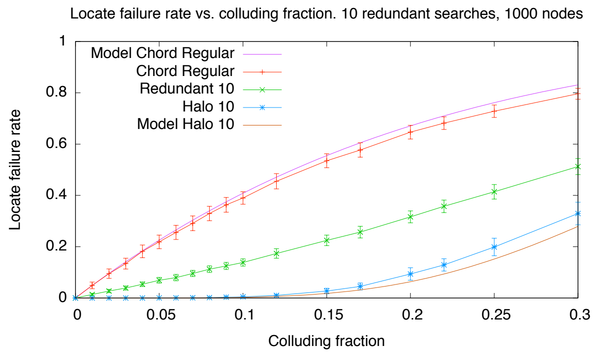
(b) Comparing failure rates for redundancy = 4



(c) Comparing failure rates for redundancy = 5

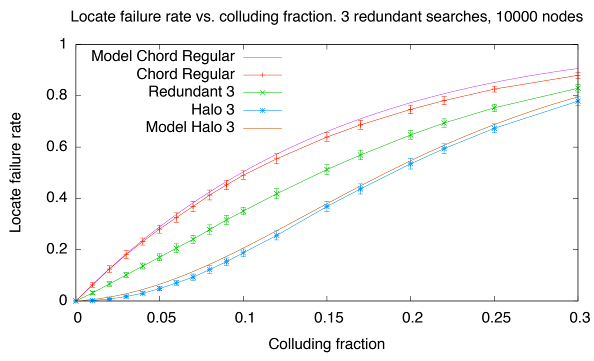


(d) Comparing failure rates for redundancy = 7

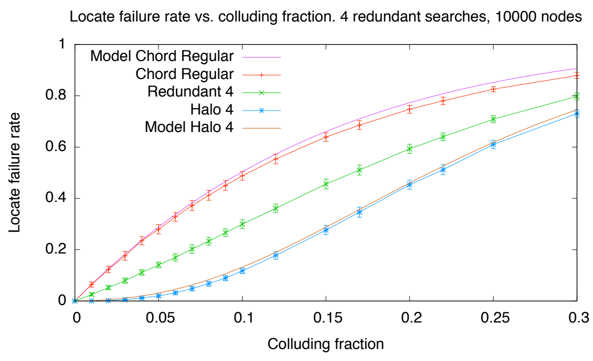


(e) Comparing failure rates for redundancy = 10

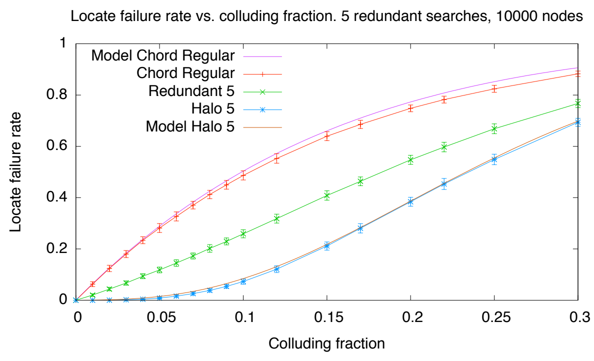
Figure 6. Locate failure rates vs. colluding fraction for various levels of redundancy in a network of 1,000 nodes. These graphs also show the closeness of fit for the analytical models for Chord and Halo. Furthermore, we can see that Halo search outperforms the naive redundant search for the same level of redundancy.



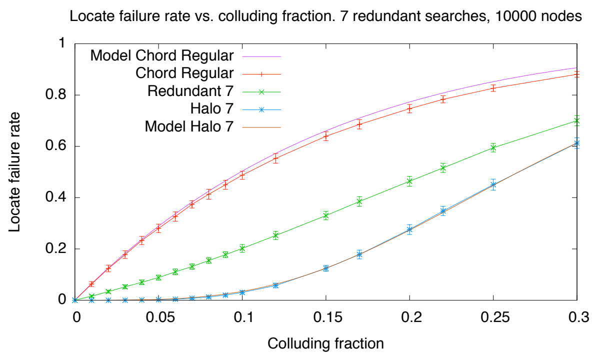
(a) Comparing failure rates for redundancy = 3



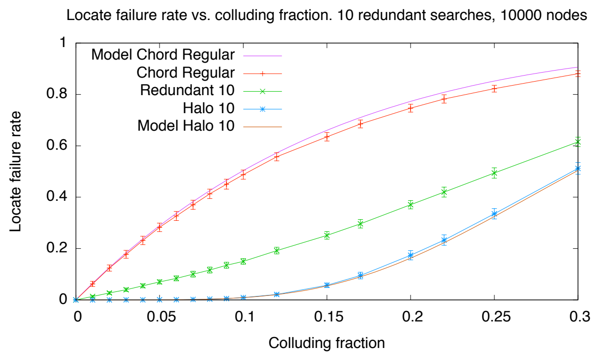
(b) Comparing failure rates for redundancy = 4



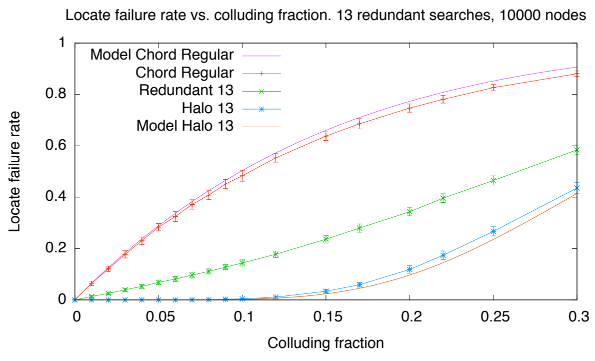
(c) Comparing failure rates for redundancy = 5



(d) Comparing failure rates for redundancy = 7



(e) Comparing failure rates for redundancy = 10



(f) Comparing failure rates for redundancy = 13

Figure 7. Locate failure rates vs. colluding fraction for various levels of redundancy in a network of 10,000 nodes. These graphs also show the closeness of fit for the analytical models for Chord and Halo. Furthermore, we can see that Halo search outperforms the naive redundant search for the same level of redundancy.