Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis

Mudhakar Srivatsa and Ling Liu
College of Computing, Georgia Institute of Technology
{mudhakar, lingliu}@cc.gatech.edu

Abstract [1]

A number of recent applications have been built on distributed hash tables (DHTs) based overlay networks. Almost all DHT-based schemes employ a tight deterministic data placement and ID mapping schemes. This feature on one hand provides assurance on location of data if it exists, within a bounded number of hops, and on the other hand, opens doors for malicious nodes to lodge attacks that can potentially thwart the functionality of the overlay network.

This paper studies several serious security threats in DHT-based systems through two targeted attacks at the overlay network's protocol layer. The first attack explores the routing anomalies that can be caused by malicious nodes returning incorrect lookup routes. The second attack targets the ID mapping scheme. We disclose that the malicious nodes can target any specific data item in the system; and corrupt/modify the data item to its favor. For each of these attacks, we provide quantitative analysis to estimate the extent of damage that can be caused by the attack; followed by experimental validation and defenses to guard the overlay networks from such attacks.

1 Introduction

A number of recent applications have been built on distributed hash table (DHT) based overlay networks. An example, of one such popular application is the distributed file storage application. These overlay network based applications are *serverless* and thus radically differ from traditional server-based applications. The server based applications have a single point of control and are thus optimized to provide very high performance. However, server based applications incur heavy administrative overheads and maintenance costs. A server requires a dedicated administrative staff, upon whose competence its reliability depends and upon whose trustworthiness its security depends [1]. Physically centralized servers are vulnerable to geographically localized faults, and their store of increasingly sensitive and valuable information makes them attractive, concentrated targets for subversion and data theft, in contrast to the inherent decentralization of desktop workstations.

A new breed of serverless applications have recently emerged like SETI@Home [14], Gnutella [6], CFS [4], Farsite [1] and etc. In contrast to traditional applications, they harness the resources available at desktop workstations that are distributed over a wide-area network. For example, SETI@Home uses the idle computing power available on personal computers to perform signal processing; applications like Gnutella, CFS and Farsite provide a distributed and decentralized file storage service. As the hardware resources are becoming cheaper and cheaper, these desktop personal computers are turning out to be more and more powerful. As a consequence, a large fraction of computing, networking and storage resources available at these desktop computers are underutilized [1]. The collective resources available at these desktop workstations amount to several peta-flops of computing power and several hundred peta-bytes of storage space. Their collective power is several order of magnitude larger than the most powerful supercomputer built so far.

These emerging trends have motivated the development of various applications on wide-area overlay networks. An overlay network is a virtual network formed by nodes (desktop workstations) on top of an existing IP-network. Overlay networks typically support a lookup protocol. A lookup (or a search) operation identifies the location of a file (say, the IP-address of a node that hosts the file) given its filename. There are two kinds of lookup protocol that have been popularly deployed, (i) unstructured overlay networks: Gnutella-like over-

[1] This research is partially supported by NSF CNS CCR, NSF ITR, DoE SciDAC, DARPA, CERCS Research Grant, IBM Faculty Award, IBM SUR grant, HP Equipment Grant, and LLNL LDRD.

Any opinions, findings, and conclusions or recommendations expressed in the project material are those of the authors and do not necessarily reflect the views of the sponsors.

lay networks, and (ii) structured overlay networks: distributed hash tables (DHT) based overlay networks. Unstructured overlay networks use an inefficient broadcast based lookup protocol that floods a search query over the entire overlay network. A DHT based lookup protocol intelligently routes a search query such that it reaches the destination node in a small and bounded number of hops. This has made DHT-based overlay networks a popular choice for file storage applications like CFS, Farsite and OceanStore.

Serverless applications are faced with the challenge of having to harness the collective resources of loosely coupled, insecure, and unreliable machines to provide a secure, and reliable file-storage service. To complicate matters further, some of the nodes in the overlay network could be malicious. Hence, serverless applications must be aware of the potential threats that could be caused by malicious nodes. One way to minimize threats in these systems is to understand the potential threats and the level of damages they may cause to the system and to increase the system's ability to defend itself from malicious intents, malicious behaviors, and potential threats incurred by known attacks or unpredicted attacks.

In this paper, we focus primarily on the vulnerability at the overlay networks layers for serverless application services. In particular, we focus on several attacks that can potentially thwart the functionality of the system, by preventing the nodes from locating or accessing data on a DHT-based overlay network. The research presented in this paper has two novel contributions. First, we identify attacks on the DHT-based routing schemes and identify some *critical properties* of the overlay network routing (lookup) protocol that determine the extent of damage caused by these attacks. Second, we extend in the results in the Sybil Attack paper [5] and show that if pseudo-spoofing is not controlled by the system then the malicious nodes can *easily* target any data item stored in the system. We provide an in-depth quantitative analysis on the extent of damage these attacks might cause, followed by experimental validations, and present counter measures against them.

2 Formal Model

In this section, we formally describe a set of common properties of structured overlay networks. Our formal model brings out the important concepts behind DHT-based systems like Chord [17], CAN [12], Pastry [13] and Tapestry [2] that aid us in analyzing the vulnerabilities and security threats on structured overlay networks.

A typical DHT-based overlay network consists of a routing table based lookup service. The lookup service maps a given key to a node (usually the IP-address of the node) that is responsible for the key. Storage protocols are layered on top of the lookup protocol. For instance, CFS [4] is a wide-area cooperative file system layered on Chord [17]; while OceanStore [8] is a distributed file system layered on Tapestry [2]. A generic DHT-based lookup service has the following properties:

(P1) A key identifier space, K. K is a m-bit identifier space where each data item is mapped to a unique identifier $d \in K$ using any standard hash function (like MD5 [11] or SHA1 [15]).

(P2) ID Mapping Scheme defines a node identifier space S. For example, Chord uses a one-dimensional circular identifier space; while CAN uses a d-dimensional coordinate space. Each node n is assigned an identifier $ID(n) \in S$. Some DHT based systems (CAN [12]) allow nodes to choose their identifier, while most others derive the identifier of a node n, namely $ID(n)$, as a strong one-way function of an external identifier (EID) of the node n. For example, $ID(n)$ could be equal to $hash(IP(n))$, where $IP(n)$ denotes the IP-address of node n. In this example, IP-address of a node is used as its external identifier.

(P3) Rules for dividing the node identifier space among the nodes. The DHT-based schemes define a responsibility function for every node n which maps it to a contiguous segment of identifier space S at time t, denoted as $Resp_t(n)$. At any given time instant t, $\{Resp_t(n) \mid n \in N(t)\}$ *partitions* the node identifier space S, where $N(t)$ refers to the total collection of all nodes in the system at time t. The algorithms also ensure that statistically every node n shares the identifier space equally; that is, at any time instant t, $sizeof(Resp_t(n)) \approx \frac{sizeof(S)}{N(t)}$. Note that the function $sizeof$ depends on the nature of the identifier space. For example, in Chord, $sizeof(x)$ could be defined as the length of the segment x; while in CAN, $sizeof(x)$ could be defined as the *volume* of the coordinate space spanned by x.

(P4) Data Placement Scheme specifies rules for mapping keys to nodes: A node n is responsible for a key $k \in K$ at time t if and only if $k \in Resp_t(n)$. This guarantees that any key k would always be found since the set $\{Resp_t(n) \mid n \in N(t)\}$ partitions the node identifier space S.

(P5) Routing Scheme uses the per-node routing tables. Routing table entries on every node maintain references to other nodes. More specifically, a distance metric is defined between any two identifiers i and j as $dist(i, j)$. For example, in Chord, $dist(i, j)$ may be simply defined as the length of the segment (i, j); while in CAN $dist(i, j)$ could be defined as the Cartesian distance between the points i and j in a d-dimensional coordinate space. When a node n is queried for key k, it returns a

node m that is *closer* to key k; that is, $dist(ID(n), k) \geq dist(ID(m), k)$.

(P6) Rules for updating routing tables as nodes join and leave. When a new node m joins the network at time t, it typically contacts an existing node $n \in N(t)$ such that $ID(m) \in Resp_t(n)$. Note that there always exists such a node n since the set $\{Resp_t(n) \mid n \in N(t)\}$ partitions the identifier space S. The node m typically assumes responsibility over a portion of the identifier space mapped to node n; that is; $Resp_{t'}(n)$ and $Resp_{t'}(m)$ partitions the space $Resp_t(n)$ for $t' > t$. Similarly, when a node leaves the network, it hands over its responsibilities to another node in the system.

The DHT-based systems guarantee location of any data item within a bounded number of application level hops. However, this advantage comes with a price: the DHT-based systems enforce a highly rigid structure and rely heavily on the correct functioning of (almost) all nodes in the system. In short, an attacker can potentially harm the overlay network by targeting these delicately balanced structures enforced by DHT-based systems.

3 Adversary Model

Adversaries refer to those nodes in the system, which either intentionally or unintentionally fail to follow the systems's protocols correctly. For example, an adversary may try to mislead non-malicious nodes by providing them with wrong information in the form of incorrect lookup results or providing invalid data through the data storage layer. We also assume that an adversary may be aware of other malicious nodes and hence, they may join hands (*collude*) in a conspiracy against the legitimate nodes.

We assume that the underlying IP-network layer is secure. Hence, (i) A malicious node has access only to the packets that have been addressed to it, (ii) All packets that can be accessed by a malicious node can be potentially modified (corrupted) by the malicious node. More specifically, if the packet is not encrypted (or does not include a message authentication code (MAC)) then the malicious node may modify the packet in its own interest, and (iii) The underlying domain name service (DNS), the network routers, and the related networking infrastructure is completely secure, and hence cannot be compromised by a malicious node.

We also assume that a malicious node may *own* one or more external identifiers (like IP-addresses). A malicious node may assume any of the external identifiers it owns. The number of external identifiers that could be owned by a node depends entirely on the nature of the external identifier. For example, with IP-address as the EID, introduction of IPv6 [7] could permit a malicious node to utilize virtually thousands of IP addresses. The same argument applies to dial-up users who obtain Dynamic IP-address from a huge ISP (Internet Service Provider). If the overlay network allows a node to choose its identifier, then it only makes it easier for a malicious node to attack the system. Therefore, our adversary model assumes that the system always derives a node's identifier from its EID.

Under the adversary model discussed above, a collection of malicious nodes can perform the following targeted attacks:

- *Attacking the Routing Scheme (Routing Anomalies)*: The malicious nodes return incorrect lookup results, thereby, increasing the probability of lookup failures or dramatically increasing the cost of the lookup operation. We identify the key properties of the lookup protocol that determine the extent of damage caused by such an attack. We also illustrate the vitality of these properties in resisting the attack through existing systems like Chord and CAN.
- *Attacking the ID Mapping Scheme*: The malicious nodes plant an attack on a chosen data item stored in the network. We show that such an attack is very powerful, though it is quite expensive for the malicious nodes to execute such an attack.

4 Attacks on the Routing Scheme

A typical DHT-based overlay network constructs a topology in which every node plays the role of a client, a server, a router, and a domain name server. Nodes act as router cum domain name server when they translate an identifier to the IP-address of a node that is responsible for the identifier (see Property P4 in Section 2). Malicious nodes can potentially exploit this feature to misguide legitimate nodes with incorrect lookups. For example, a malicious node can lie about the next hop when it is queried for some identifier. This could result in denial of information - a legitimate node is denied access to a data item; or result in sub-optimal performance of the lookup algorithm.

There are several possible defense mechanisms to counteract such vulnerabilities. Concretely, the properties of the distributed lookup algorithm can be used to ascertain whether a lookup for a given identifier is correct or not. For example, Sit and Morris [16] exploit the fact that: at each hop of the Chord lookup algorithm the query originator knows that the lookup protocol should lead him/her *closer* to the destination identifier (see Property P5 in Section 2). Hence, the query originator can check for this and detect an incorrect lookup. On sensing an in-

correct lookup, the query originator can choose an alternative (possibly sub-optimal) path towards the destination identifier. Informally, a *lookup path* from a source node n to a destination node m is the sequence of nodes through which the lookup operation succeeds. In view of the above discussion, the performance of a lookup algorithm (in the presence of malicious nodes) depends on the following three factors: (i) Existence of multiple alternate paths between any two identifiers, (ii) Lookup costs along alternate paths between any two identifiers, and (iii) Ability to detect incorrect lookups.

4.1 Alternate Lookup Paths

We first highlight the importance of alternate (possibly sub-optimal) paths in enhancing the performance and the robustness of a lookup algorithm in the presence of malicious nodes. We capture the notion of *alternate lookup paths* using the notion of *independence* of lookup paths. We formally define independence between two lookup paths as follows:

Definition *Independent Lookup Paths: Let P and Q be different lookup paths from node n to node m. Two lookup paths P and Q are said to be independent if and only if they do not share a common node other than the source node n and the destination node m.*

Hence, each independent lookup path [2] between a node n and a node m is a *statistically independent* route for a lookup with key $k \in Resp(m)$, originated at node n, to succeed. Note that the property of independence is stronger than that of alternate paths. For instance, there may exist multiple paths between node n and node m; however all these paths may happen to share a common node, thereby making no two of them independent.

Most of the DHT-based systems do not guarantee the existence of multiple independent lookup paths between any two identifiers. For instance, in Chord, all lookups for a key $k \in Resp(m)$ will succeed only through the node $pred(m)$, where $pred(m)$ denotes the predecessor of node m along the Chord identifier circle. Hence, the number of independent lookup paths between any node n and key $k \in Resp(m)$ is one, since all such lookup paths include node $pred(m)$. If the node $pred(m)$ were malicious, lookup for any key $k \in Resp(m)$ would fail. On the other hand, this situation is greatly mitigated in DHT-based schemes like CAN that have multiple independent lookup paths between any two identifiers. More specifically, a d-dimensional CAN topology has d independent lookup paths.

[2] In general one can estimate number of independent paths as follows: By Menger's theorem [9], the number of independent paths equals the vertex connectivity of a graph; and vertex connectivity can be measured using network flow techniques [10]

We use the *probability of lookup failure* as a metric for measuring the benefits of alternate lookup paths. A lookup for node m at node n results in a failure if *all* the lookup paths from node n to node m contain at least one malicious node. Intuitively, larger the number of independent lookup paths, smaller is the probability of lookup failure. In the following portions of this section we derive bounds on the probability of lookup failure in terms of the number of independent lookup paths.

Quantitative Analysis. Let ind denotes the number of independent lookup paths between the source and destination node, p denotes the fraction of malicious nodes in the system, and M denotes the number of hops required for a lookup to succeed. Given ind independent lookup paths of length M hops, one can show that the probability of lookup failure is bounded by Equation 1.

$$p^{ind} \leq Pr(lookup\ failure) \leq (1-(1-p)^M)^{ind} \quad (1)$$

Note that the existence of ind independent paths between a source node src and destination node dst implies that there exists nodes $\{n_1, n_2, \cdots, n_{ind}\}$ one of which occurs on all paths from the node src to node dst. The lower bound is derived from the fact that a lookup from node src for node dst is guaranteed to fail if all the ind nodes $\{n_1, n_2, \cdots, n_{ind}\}$ were malicious. Let $\{P_1, P_2, \cdots P_{ind}\}$ be any set of ind independent lookup paths between node src and node dst containing nodes $\{n_1, n_2, \cdots, n_{ind}\}$ respectively. The probability of a lookup succeeding on any lookup path P_i with M hops equals $(1-p)^M$, namely, the probability that all the nodes on that path were good. The upper bound follows from the independence of lookup failures along each independent lookup path [3]. For small values of p, the probability of lookup failure can be approximated to $(M*p)^{ind}$ $(M*p \ll 1)$. Intuitively, the longer a lookup path (M), the higher is the chance that at least one node on the lookup path turns out malicious. The statistical independence in lookup failures along multiple independent paths ensures that the probability of lookup failure decreases *exponentially* with the number of independent paths (ind).

4.2 Alternate Optimal (less costly) Lookup Paths

Yet another important issue to be addressed with regard to alternate lookup paths is the cost of these alternative paths themselves. Ideally, the alternate paths should be *alternate optimal* paths; otherwise, choosing highly sub-optimal alternate paths may degrade the performance of

[3] This is an upper bound because the presence of alternate (but not independent) lookup paths may decrease the probability of lookup failure

a lookup algorithm. Unfortunately, most of the DHT-based schemes do not address such issues. For illustration, in Chord, say a node n queries a good node x for key k and obtains the result as node y. Now, node n issues a query for key k to node y. If node y were malicious, it would return an incorrect lookup result. If node n were to detect the invalid result, the best choice it has is to ask node x (previous node on the lookup path) for its *next best choice* for the query with key k. Now, node x has to return a sub-optimal result, since it is *not aware* of any node that is closer to key k than the malicious node y. Since Chord maintains pointers to nodes at distance that are an integer power of 2, it is likely that the next best choice proceeds only half the distance along the identifier circle when compared to the optimal choice. On the other hand, in CAN it is quite possible for the alternate paths to be *near optimal*. Consider the same scenario described above. Assume, without loss of generality, that the identifier of node x and key k differ along coordinates $\{c_1, c_2, \cdots, c_l\}$. Now, if node x and node y varied along a coordinate c_j (for some j, $1 \leq j \leq l$), then node x could choose other neighboring nodes that vary along coordinates other than c_j. In comparison with Chord, the alternate choices provided for a lookup in CAN, is likely to be much closer to optimality. We defer further discussion on alternate-optimal paths to the end of this section.

4.3 Detecting and Recovering from Invalid Lookups

Having highlighted the importance of good alternate paths, we now study the importance of detecting incorrect (malicious) lookups. In the discussion that follows, we assume two failure modes for nodes in our system: *Crash failures* and *Byzantine failures*. When a node has crash failed it does not return any results for lookup queries. Under Byzantine failure, a node can return a potentially malicious value for any lookup query. In the following portions of this section, we quantitatively analyze the cost of lookup operation under both these failure modes. For the sake of simplicity of analysis, we assume that the DHT-based scheme has multiple alternate-optimal paths between any two identifiers (like CAN). Hence, the results obtained from the results of our analysis below can be viewed as lower bounds on the lookup costs.

Quantitative analysis. Let M denote the mean number of hops required to perform a lookup operation. For instance, in Chord, $M = \frac{1}{2}log(N)$; in CAN, $M = \frac{d}{4}N^{\frac{1}{d}}$ where N is the number of nodes in the system and d represents the dimensionality of CAN's coordinate space. Let p denote the percentage of bad nodes in the system. Also assume that the bad nodes are uniformly spread throughout the node identifier space. Let $f(x)$ be a function that maps the number of hops required for a lookup when all nodes are good to the number of hops required when $p\%$ of the nodes are malicious. In other words, if a lookup would require x hops when all nodes are good, it would require $f(x)$ hops when $p\%$ of the nodes are bad.

Crash Failures. Assuming crash failures for nodes, $f_1(x)$ (the mapping function for crash failures) satisfies the following recurrence relation.

$$f_1(x) = 1 + (1-p)f_1(x-1) + pf_1(x) \qquad (2)$$

When a node n queries a node m for the next hop towards a key identifier k, node n expends one hop. If node m is a good node (probability = $1-p$) then it would return a correct lookup result. Hence, node n would have to traverse $x-1$ more hops to reach the key identifier k in the scenario where no node has crashed. In the presence of crash-failed nodes, this would require $f_1(x-1)$ hops by the definition of function f_1. If node m had crashed (probability = p), it would not return any lookup result. Node n on detecting this (though a time-out mechanism) can choose an alternate-optimal path towards key k. Hence, node n would have to traverse x more hops to reach the key identifier k in the scenario where no node has failed. In the presence of crash-failed nodes, this would cost node n additional $f_1(x)$ hops (it is still possible to reach key k in x hops in spite of ruling out one lookup path, since we have assumed the presence of alternate-optimal pathss).

Solving the recurrence relation, we get,

$$f_1(x) = \frac{x}{1-p} \qquad (3)$$

Hence, the expected (average) number of hops required for a lookup operation is, $E[f_1(x)] = \frac{M}{1-p}$ since, $M = E[x]$ by definition.

Byzantine Failures. Assuming Byzantine failure of nodes, the cost of a lookup operation depends on certain properties of the DHT-based system. In most of the DHT-based systems it is possible to detect invalid lookups with a reasonable degree of certainty since the lookup at each hop is *supposed* to get closer to the destination identifier [16]. Hence, the query originator can check for this and detect an incorrect lookup. Upon finding an incorrect lookup, the query originator can choose an alternative (possibly sub-optimal) path towards the destination identifier. However, in certain cases like CAN's RTT optimization, the lookup results cannot be verified since the intermediate lookup results are not available to the source node (query originator). In view of the above discussion, we consider the following two scenarios:

- Scenario 1: An incorrect lookup can always be detected.

- Scenario 2: An incorrect lookup cannot be detected; and hence, the querier blindly follows the lookup result.

Observe that *Scenario 1* is simply equivalent to that of crash failure of nodes. Note that if a non-malicious node n receives an incorrect lookup from a malicious node m then node n can simply assume that node m has crash failed. Hence, the lookup cost in scenario 1 is given by Equation 3.

Assuming that incorrect lookups can neither be detected or corrected, $f_2(x)$ (the mapping function for scenario 2) satisfies the following recurrence relation,

$$f_2(x) = 1 + (1-p)f_2(x-1) + pf_2(M) \qquad (4)$$

Note that the first two terms in the expression for f_2 follows from the same arguments for crash failures; it costs unity for node n to query m and $f_2(x-1)$ additional hops if node m were good. However, if node m were malicious, it would return an incorrect lookup and node n would blindly abide by node m's result. Note that a collection of malicious nodes X may keep circulating the query among nodes in X, thereby ensuring that the query never succeeds. However, this would cost bad nodes in terms of their bandwidth for answering repeated queries. Hence, we assume that bad nodes return a random node in the system as the next hop for key k to the query originator node n. Now, since the random node could be located anywhere in the network, it would be M hops away from the key k in the scenario where all nodes are good. Hence, in the presence of malicious nodes, the lookup operation would cost node n additional $f_2(M)$ hops.

Using the recurrence relation 4 we compute the average number of hops required for a lookup operation as follows. We approximate $E[f_2(x)]$ to $f_2(E[x])$ which is equal to $f_2(M)$ (since $M = E[x]$). Note that $f_2(M)$ denotes the lookup cost for scenario 2 in the presence of malicious nodes when it would have required M hops in the absence of malicious nodes. We observed that in most DHT-based systems the mean number of hops M is also the most probable number of hops between any two random nodes in the system. Hence, such an approximation does not significantly perturb our analytical results. Further, our experimental results in Figure 3 show that this approximation is acceptable. Hence,

$$E[f_2(x)] \approx f_2(M) = \frac{1-(1-p)^M}{p(1-p)^M} \qquad (5)$$

Summary. Clearly, scenario 2 pays higher penalty for its inability to detect and recover from invalid lookups. Intuitively, in scenario 1 (or under crash failures), the lookup makes one successful hop with a probability $1-p$; hence, each hops translates into $\frac{1}{1-p}$ hops. On the other hand, scenario 2, pays heavily for every failed lookup. Let us say that we start with a state S where the lookup operation is M hops from its target. Now, this lookup operation succeeds if all the nodes in the path to the target are good ($Pr = (1-p)^M$); else it is back to its original state S. Hence, the probability that a lookup succeeds in any given attempt starting from state S is $(1-p)^M$; and hence the lookup cost is varies as $(1-p)^{-M}$.

DHT	Lower Bound	Expt Result	Upper Bound
Chord	0.1	0.29	0.40
CAN-2	0.01	0.09	0.16
CAN-3	0.001	0.04	0.06
CAN-4	0.0001	0.015	0.028
CAN-10	0.0	10^{-4}	10^{-4}

Table 1: Probability of Lookup Failure ($p = 10\%$): Quantitative Analysis

4.4 Experimental Validation

We have so far identified and quantitatively analyzed the importance of multiple independent paths, alternate optimal paths, the ability to detect and recover from invalid lookups. Now we present two sets of experiments to validate the above analysis. First, we study the dependency between the number of independent paths and the probability of lookup failure. Second, we measure the lookup cost in the presence of malicious nodes and evaluate the performance of the proposed defense mechanisms regarding to the two scenarios: An incorrect lookup (1) can always be detected or (2) cannot be detected.

Experiment I. In this experiment, we demonstrate that having multiple independent lookup paths indeed decreases the probability of lookup failures. We simulated the working of a P2P system using the Chord lookup protocol with 1024 nodes. The average lookup cost when there are no malicious nodes is 5, i.e., $M = 5$. We also constructed CAN systems with approximately the same average lookup cost; a 2-dimensional CAN with 100 nodes ($M = 5$), a 3-dimensional CAN with 216 nodes ($M = 4.5$), a 4-dimensional CAN with 625 nodes ($M = 5$), and a 10-dimensional CAN with 1024 nodes ($M = 5$). A random set of $p\%$ of the nodes were chosen to behave maliciously. From a practical standpoint, we associate a time-to-live (TTL) with every lookup operation. Hence, a lookup operation is successful only if it terminates correctly within TTL overlay network hops.

Table 1 compares the experimental results with the bounds obtained from our analytical model (Equation 1, Section 4.1) when $p = 10\%$. Although the bounds obtained from our quantitative analysis are not abso-

lutely tight (because Equation 1 does not consider alternate but not independent paths), the trends revealed by our analysis closely reflect the results obtained from our experiments. Most importantly, observe that the upper bound on the probability of lookup failure sharply decreases with increase in the number of independent lookup paths. This motivated us to experiment with overlay network with different number of independent paths.

Figure 1 shows the probability of lookup failure with $TTL = 100$ and $p = 10\%$ and varying M. Observe that the probability of lookup failure increases with the mean number of hops (M). Note that for any lookup path to succeed, all the nodes on the lookup path must be non-malicious; longer the path, higher is the probability that at least one malicious node appears on that path.

Experiment II. In this experiment, we illustrate the performance of DHT-based systems for the scenarios 1 and 2 discussed in section 4.3. Figure 2 shows the average lookup cost for scenario 1 wherein, the legitimate nodes verify whether the lookup result *appears* to be valid by checking if the new node is indeed closer to the key k. On detecting an invalid lookup, node n gets the next best alternative node and forwards the query to it. Note that this strategy for detecting invalid lookups has scope for an *one-sided error*; it may conclude that an incorrect (or a sub-optimal) lookup result is correct. The estimates from our quantitative analysis are labeled as 'check_Ehops' (Equation 3, Section 4.3); they denote the lower bounds obtained on the expected number of hops assuming a large number of independent paths. The lines labeled 'chord_check' and 'can$d$_check' refer to the cases wherein the lookup protocol checks for validity as a part of the Chord, and the d-dimensional CAN protocols respectively. We shall discuss the implications of Figure 2 in conjunction with Figure 3. Figure 3 shows the average lookup cost for scenario 2 wherein, the legitimate nodes do not test the validity of lookup results. The line labeled 'nocheck_Ehops' (Equation 5, Section 4.3) shows the results obtained from our quantitative analysis. The lines labeled 'chord_nocheck' and 'can$d$_nocheck' refer to the cases where the Chord and the d-dimensional CAN protocols were used without checking for the validity of lookup operations. Based on Figures 2 and 3, we can testify the following statements:

Validate our quantitative analysis. Observe that 'check_Ehops' and 'nocheck_Ehops' act as lower bounds for the 'check' and the 'nocheck' versions of the lookup protocol respectively. Also, observe that the results for a 10-D CAN closely matches our quantitative analysis, since our analysis was specifically targeted at obtaining lower bounds on lookup costs assuming a large number of independent paths between any two identifiers.

Checking the validity of a lookup result becomes very important for large values of p. However, for small values of p, checking the validity of a lookup result is not very vital in the presence of multiple independent paths. In fact, for $p \leq 40\%$, a 10-D CAN with no validity checks incurs no more than 5-8% higher lookup cost than a 10-D CAN that performs validity checks. But, for large values of p (around $p = 70\%$), lookup protocols that do not include validity checks incur as high as 40-50% (for 10-D CAN) to about 200% (for Chord) (Note that Figure 3 shows the results only up to $p = 50\%$).

Importance of good alternate paths. Since 'can10_check' has multiple near-optimal alternate paths, its lookup cost is within twice of the optimal lookup cost even when p equals 70%. On the other hand, 'chord_check' shows much poorer performance, primarily because of the fact that Chord does not provide multiple independent lookup paths. Further, the vitality of alternate paths is more blatantly revealed by Figure 3 from the fact that, 'chord_check' performs much worse (3-4 times) than 'can10_nocheck'.

5 Attacking the ID-to-Key Mapping Scheme

In this section, we present the ID-to-Key mapping attack (ID Mapping for short) that show how the malicious nodes could exploit the identifier-to-key mapping to corrupt a *chosen* data item stored in the system by spoofing multiple identities (*pseudo-spoofing*). We quantify and analyze the cost of attacking a chosen data item and discuss some approaches to mitigate this problem.

The famous Sybil Attack paper [5] showed that entities (nodes) can forge multiple identities for malicious intent, and hence, a small set of faulty entities may be represented through a larger set of identities. Douceur concludes in [5] that for direct or indirect identity validation (without using a centralized trusted agency), a set of faulty nodes can counterfeit an unbounded number of identities (pseudo-spoofing). One solution suggested to counter the Sybil attack in [5] is using secure node IDs that are signed by well-known trusted certifying authorities. However, as Douceur pointed out himself that mandating that every node must possess a certificate would turn out expensive. Hence, one is forced to employ weak secure node IDs (with challenge-response schemes to verify the node IDs); for example, several systems like CFS [4] use the IP-address of a node as its weak secure ID. Therefore, it becomes very important to quantify and analyze the security trade-offs when weak secure IDs are used. This section discusses an extension of the pseudo-spoofing attack which results in the loss of a chosen data item (d) assuming the identity of the data item ($ID(d)$) is known.

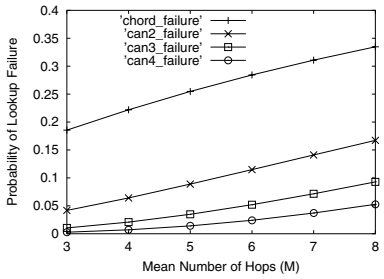

Figure 1: Probability of a lookup failure: Initial $TTL = 100$ and $p = 10\%$

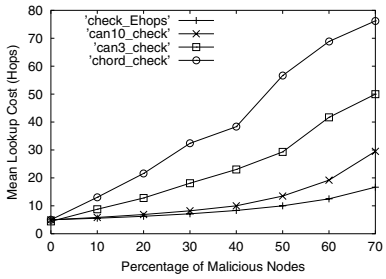

Figure 2: Lookup Costs: Scenario 1 with $M = 5$

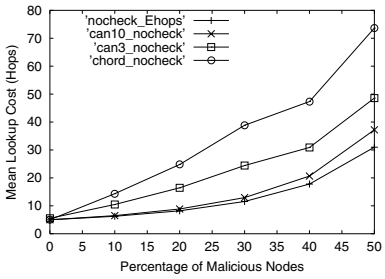

Figure 3: Lookup Cost: Scenario 2 with $M = 5$

Almost all DHT-based system use a strong one-way hash function (like MD5 [11] or SHA1 [15]) to derive a node identifier from its external identifier (EID) (refer Section 3); and, any node p has direct access to a data item d if and only if $ID(d) \in Resp(p)$ (see properties P2 and P4, Section 2). Therefore, for a malicious node n to target a data item with identifier d, it needs to select an EID such that if node n joins the system with $ID(n) = hash(n.EID)$, it will be made responsible for the target data item d. However, we show that the cost of attacking a specific data item $d \in K$ (key identifier space K) is much *easier* than inverting the ID mapping hash function. Recall that by birthday paradox [18], the cost of inverting a strong one-way hash function is $O(\sqrt{sizeof(S)})$, where S is the hash space (in our case, the identifier space). In this section, we present a $O(G)$ attack for attacking any chosen data item stored in the system. This attack assumes significance because the number of good nodes G is of the order of a few thousands, while $sizeof(S)$ for say MD5 is 2^{128}.

Concretely, given the identifier of a data item d a malicious node can locate the node at which the data item d is stored using the lookup protocol. Let the data item d be stored at node q at time t, namely, $ID(d) \in Resp_t(q)$. A malicious node p gains access to the target data item d as follows: First, it attempts to pick an EID that hashes into $Resp_t(q)$. As we have described in section 2 (P6), when a node p with $ID(p) \in Resp_t(q)$ joins the network, it would share the responsibility of node q. Given that the node p gets assigned a portion of the node q's responsibility, there is a good chance that the target data item d indeed gets assigned to node p. Observe that if the system did not enforce restrictions on a node's identifier, a malicious node p could trivially choose its node identifier to be $ID(d)$ thereby ensuring that the target data item d is assigned to it.

5.1 Quantitative Analysis

Let G denote the number of good nodes in the system. Assume for the sake of simplicity that there are no malicious nodes in the system. Now, every node in the system would be responsible for approximately $1/G^{th}$ portion of the identifier space. The identifier space can be viewed as if it were divided into G equal sized buckets. Hence, the probability that a random identifier falls into a given bucket q is $1/G$. The probability that a malicious node hits upon an EID that hashes into bucket q in its k^{th} attempt is given by,

$$Pr(k\ attempts) = \left(1 - \frac{1}{G}\right)^{k-1} \frac{1}{G} \tag{6}$$

where the first $k-1$ attempts fails to fall into bucket q, while the k^{th} attempt succeeds. Observe that the number of attempts required in Equation 6 follows a Geometric Distribution. The malicious nodes could choose q such that their target data item d is currently held by node q, i.e., $ID(d) \in Resp_t(q)$. Using the standard properties of a Geometric distribution, it follows from Equation 6 that the expected number of attempts required to obtain an identifier that hashes into node q is G. Also, the probability that more than G attempts are required is $\frac{1}{e}$ for substantially large values of G. But, having obtained an identifier that hashes into node q's identifier space does not guarantee that the malicious node p is assigned the target data item d. (Note $Resp_{t'}(p)$ and $Resp_{t'}(q)$ partitions $Resp_t(q)$, for $t' > t$ (see property P6 in Section 2). Since the data item d can lie anywhere in the identifier space assigned to node q, the probability that node p gets to store data item d after it joins the network is $\frac{1}{2}$. Hence, with a reasonably large probability, a malicious node p can obtain access to a target data item d in G attempts. Hence, an adversary that possesses G pseudo-identifiers can obtain access to the target file replica f_i with probability $Pr(N_{EID} \leq G) = 1 - \frac{1}{2}\left(1 - \frac{1}{e}\right)$ (equivalently, $Pr(N_{EID} > G) = 1 - \frac{1}{2}\left(1 - \frac{1}{e}\right) = \frac{1}{2}\left(1 + \frac{1}{e}\right)$). Consequently, one can improve the chances of this attack in $O(G)$ attempts since, for some integer $c > 0$:

$$Pr(Number\ of\ attempts > cG) = \left(\frac{1}{2}\left(1 + \frac{1}{e}\right)\right)^c \tag{7}$$

In a system where R replicas are maintained for each data item, a group of B malicious nodes may join hands to corrupt a data item d irrecoverably as follows. Each of the malicious nodes performs an ID Mapping attack

G	Mean time (ms)	E[N_{EID}]	Pr($N_{EID} \leq 4G$)	Pr($N_{EID} \leq 8G$)
1024	2.32	2112	0.80	0.96
2048	4.22	4301	0.76	0.95
4096	8.47	8157	0.76	0.97
8192	16.08	16421	0.84	0.99

Table 2: Attack on ID Mapping Scheme

using the above strategy on any of the R replicas of data item d. When the malicious nodes succeed in gaining control over cr copies of the data item d, they can corrupt it and leave the system.

Consider the case wherein the IP-address of a node is used as its EID. Given the fact that IPv6 can potentially provide every node with thousands of addresses, it is quite feasible, though expensive, for a malicious node to perform this attack. The same argument is also applicable to dial-up users who obtain Dynamic IP-address from a huge ISP (Internet Service Provider). Also note that computing $O(G)$ hashes is computationally feasible. As a simple example, using the standard OpenSSL library, a typical 900MHz Pentium III takes about 1 second to compute one million hashes (MD5).

5.2 Experimental Validation

We simulated the Chord lookup protocol [17] with varying number of good nodes. We chose 100 random data items to attack. Table 2 summarizes our observations on the number of EIDs required for a successful attack on these data items. Our observations very closely match the results from our analysis (Equation 7): $Pr(N_{EID} \leq 4G) = 0.79$ and $Pr(N_{EID} \leq 8G) = 0.96$.

5.3 Defense

The Sybil attack paper [5] suggests the use of trusted certification authorities to strictly bind an identity to an entity (node). However this requires every node to reveal its identity in order to join the system. It may also discourage a few nodes from joining the overlay network in the first place. In order to reduce certification costs and motivate users to join the overlay network, one could employ weak secure identifiers, like the node IP-address, which can be challenged and verified easily.

There are other approaches that could mitigate this problem. When a new node joins the system, it typically contacts a publicly known and trusted bootstrap server to obtain an *entry point* node to bootstrap itself into the system. In our solution, the bootstrap server assigns a random $id \in S$ to a node (and issues a certificate with a short life-time) when it joins the system. Observe that if a malicious node joins the system $O(G)$ times, it gets assigned a given target data item with a large probability as in Equation 7. However note that contrary to the techniques that derive a node's ID from its EID, a malicious node cannot *offline* determine the EID that can be used to gain control over the target data item. Instead, the malicious node has to physically attempt joining the system $O(G)$ times, each time contacting the bootstrap server. Note that contacting the bootstrap server $O(G)$ takes significantly longer time; but, it does not prevent a malicious from eventually gaining control over a target data item. To prevent this, one could implement weak security checks through the bootstrap server; for example, the bootstrap server could detects frequent attempts by a node from a single *IP-domain* [4] to join the system. In conclusion, the overlay network may use weak secure IDs as long as spoofing a large number of pseudo-identifiers over a short duration of time is very hard.

6 Related Work

Sit and Morris [16] discuss several security considerations in a distributed hash table based overlay networks. They present a framework for performing security analysis of structured overlay networks and discuss a wide range of possible attacks including: routing table maintenance and network partitioning; application layer attacks on file storage; and ad hoc attacks like denial-of-service attacks by rapidly joining and leaving the network. Their paper suggests the use of repeated checking as a defense against routing attacks on the system. Our work not only quantifies the lookup costs using repeated checks but also shows that merely performing repeated checks does not help especially in the absence of multiple independent lookup paths.

Castro *et al.* [3] discuss several issues on secure routing in DHT based overlay networks. They suggest redundant routing as a solution for strengthening the routing scheme using a routing failure test based on the density of node identifiers. In redundant routing, the query source sends lookup query through different routes with a hope that at least one them eventually reaches the destination node. Note that our analysis on multiple near optimal independent paths is applicable to the case where redundant routing is used. Also note that having multiple independent paths improves the probability of success and lowers the lookup cost for both repeated checking and redundant routing techniques.

The Sybil attack paper [5] showed that entities (nodes) can forge multiple identities for malicious intent, thereby having a set of faulty entities represented through a larger set of identities. The paper also suggests that one solution to the Sybil attack is using secure node IDs that are signed by well-known certifying agents. However, requiring every node to possess a certificate would turn

[4] Multiple IP-addresses owned by a node hopefully fall into the same IP-domain

out quite expensive; and may discourage even the good nodes from joining the system in the first place. Hence, one is forced to employ weak secure node IDs (that significantly reduce the number of identities owned by an entity). Therefore, this paper (Section 5) assessed the risks and security threats when such weak secure IDs are deployed in a DHT-based overlay network.

7 Conclusion

We have studied vulnerabilities of the overlay network layer in a DHT-based overlay networks through investigating two targeted attacks and suggesting possible defense mechanisms. We have identified the key properties of a DHT-based system that determine the hardness of these attacks. First, we have highlighted the importance of multiple independent paths, alternate optimal paths, the ability to detect and recover from invalid lookups. Second, we discussed an attack on the ID Mapping Scheme and showed that attacks on chosen data items are quite plausible on DHT-based systems; we have also demonstrated the dependency between the hardness of such an attack and the number of external identifiers owned by a node. We have presented quantitative analysis to estimate the extent of possible damages caused by these two types of attacks and used experimental methods to demonstrate the validity of the attacks identified. In conclusion, we strongly believe that incorporating these security features in the overlay network layer will provide a secure infrastructure for building large-scale distributed and decentralized applications.

References

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, available and reliable storage for an incompletely trusted environment. In *5th Symposium on OSDI*, 2002.

[2] J. K. B. Zhao and A. Joseph. Tapestry: An infrastructure for fault-tolerance wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, 2001.

[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proceedings of the 18th SOSP*, October 2001.

[5] J. Douceur. The sybil attack. In *2nd Annual IPTPS Workshop*, 2002.

[6] Gnutella. The gnutella home page. http://gnutella.wego.com/, 2002.

[7] IPv6. The ipv6 information page. http://www.ipv6.org/, 2002.

[8] J. Kubiatowics, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[9] MathWorld. Menger's theorem. http://mathworld.wolfram.com/MengersTheorem.html, 2002.

[10] MathWorld. Network flow. http://mathworld.wolfram.com/NetworkFlow.html, 2002.

[11] MD5. The md5 message-digest algorithm. http://www.ietf.org/rfc/rfc1321.txt, 1992.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.

[13] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[14] SETI@home. Seti@home: Search for extraterrestrial intelligence at home. http://setiathome.ssl.berkeley.edu/.

[15] SHA1. Us secure hash algorithm i. http://www.ietf.org/rfc/rfc3174.txt, 2001.

[16] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of IPTPS*, 2002.

[17] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, August 2001.

[18] Wikipedia. Birthday paradox. http://www.wikipedia.org/wiki/Birthday_paradox.