Vulnerabilities and Security Threats in Structured Peer-to-Peer Systems: A Quantitative Analysis

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

Abstract
A number of recent structured Peer-to-Peer (P2P) systems are built on top of distributed hash table (DHT) based P2P 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 P2P system.

This paper studies several serious security threats in DHT-based P2P systems through three targeted attacks at the P2P 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 tight data placement scheme. We show that replication of data items, by itself, is insufficient to secure the data items. The third 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 an experimental validation and defenses to guard the DHT-based P2P systems and counteract such attacks.

1 Introduction

P2P computing is commonly perceived as an infrastructure offering both opportunities and threats. One way to minimize threats in P2P systems is to understand the potential threats and the level of damages they may cause to a P2P 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.

Attacks on a general P2P system can be targeted at three layers: the network layer (say, TCP/IP), the overlay network layer (say, lookup protocols) and the application layer. There are marked differences in the security issues concerned at each of these layers. Also, the algorithms used at the higher layers largely depend on the guarantees provided by the lower layers. For instance, the lookup protocols may assume that the network layer reliably and securely delivers a message to its intended recipient node. Breaking any of these guarantees provided by a lower level layer to a higher-level layer can disrupt the entire security infrastructure.

In this paper, we focus primarily on the vulnerability of P2P systems for *targeted* (well-planned) attacks at the P2P protocol/overlay network layer. Hence, we assume that the underlying network layer is reliable and secure. 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 the overlay network. We discuss three targeted attacks and present defenses for the same. We use quantitative analysis and experiments to demonstrate the extent of damage that can be caused by these attacks with and possible defense solutions.

The famous Sybil Attack paper [6] 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 [6] 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. One solution suggested to counter the Sybil attack in [6] 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 prohibitively expensive. Hence, one is forced to employ weak secure node IDs (with challenges to verify the node IDs); for example, several systems like CFS [5] 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.

However, the Sybil Attack paper provides neither a discussion nor any formal analysis on what concrete damage such Sybil attacks might ground and how quantitative analysis can help us understand the extent of damage such vulnerabilities may cause.

The research presented in this paper is built on the results from the Sybil Attack paper [6] with two novel con-

tributions. First, we identify the attack on the DHT-based routing schemes, which, to our knowledge, has not been discussed in existing literature. We provide experimental methods and quantitative analysis on the potential damage caused by such attack and effectiveness of possible defense mechanisms. Second, based on the results of the Sybil Attack paper (a faulty peer can counterfeit a large number of identifiers), we identify another two serious denial of information attacks that can happen in concurrence with (or in the presence of) a Sybil Attack – Attack on data placement scheme and Attack on ID mapping scheme. 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 P2P systems. Our formal model brings out the important concepts behind DHT-based systems that aid us in analyzing the vulnerabilities and security threats on structured P2P systems.

A typical DHT-based P2P protocol 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 [5] is a wide-area cooperative file system layered on Chord [23]; while OceanStore [10] is a distributed file system layered on Tapestry [3]. 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 [16] or SHA1 [20]).

(P2) ID Mapping Scheme defines a node identifier space S, such that $S = K$. For example, Chord uses a one-dimensional circular identifier space; while CAN uses a d-dimensional coordinate space. Each node p is assigned an identifier $ID(p) \in S$. Some DHT based systems ([18]) allow nodes to choose their identifier, while most others derive the identifier of a node p, namely $ID(p)$, as a strong one-way function of an external identifier (EID) of the node p. For example, $ID(p)$ could be equal to $hash(IP(p))$, where $IP(p)$ denotes the IP-address of node p. 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 p which maps a node p to a contiguous segment of identifier space S at time t, denoted as $Resp_t(p)$. At any given time instant t, $\{Resp_t(p) \mid p \in N(t)\}$ *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 p shares the identifier space equally; that is, at any time instant t, $sizeof(Resp_t(p)) \approx sizeof(S)/N(t)$. 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 segment x.

(P4) Data Placement Scheme specifies rules for mapping keys to nodes: A node p is responsible for a key $k \in K$ at time t if and only if $k \in Resp_t(p)$. This guarantees that any key k would always be found since the set $\{Resp_t(p) \mid p \in N(t)\}$ 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 q joins the network at time t, it typically contacts an existing node $p \in N(t)$ such that $ID(q) \in Resp_t(p)$. Note that there always exists such a node m since the set $\{Resp_t(m) \mid m \in N(t)\}$ partitions the identifier space S. The node q typically assumes responsibility over a portion of the identifier space mapped to node p; that is; $Resp_{t'}(p)$ and $Resp_{t'}(q)$ partitions the space $Resp_t(p)$ 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. Their tightly coupled nature and heavy dependency on the correctness of each node makes the DHT-based P2P systems more susceptible to well-planned attacks. In short, an attacker can potentially harm the system by targeting the delicately balanced structure enforced by the DHT-based systems.

3 Adversary Model

Adversaries refer to those nodes in the system, which either intentionally or unintentionally do not adhere to the distributed lookup protocol correctly. The adversaries try to

mislead the 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 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 includes a message authentication code (MAC)) then the malicious node may modify the packet in its own interest. (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 identifier 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, the introduction of IPv6 [9] could endow a malicious node with 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 P2P system 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. We also assume that the malicious nodes may be aware of other malicious nodes and hence, the malicious nodes may join hands (*collude*) in a conspiracy against the legitimate nodes.

4 Targeted Attacks and Defense Mechanisms

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

Attack on the Routing Scheme (Routing Anomalies): The malicious nodes can lie about the next hop when asked about the node id of a node responsible for a particular data item and return incorrect lookup results, thereby, increasing the probability of lookup failures or dramatically increasing the cost of a lookup operation. We identify the key properties of the lookup protocol that determine the extent of damage caused by such an attack.

Attack on the Data Placement Scheme (Information Decay): The malicious nodes corrupt the information stored in the system by repeatedly joining and leaving the network and every time corrupting the data they are assigned responsibility for. We show that even when replication with majority voting scheme are used, the information stored in the P2P network decays in course of time.

Attack on the ID Mapping Scheme: The malicious nodes plant an attack on a specific data item stored in the P2P network. We show that such an attack is very powerful, though it is quite expensive for the malicious nodes to execute such an attack.

4.1 Attack on the Routing Scheme: Routing Anomalies

A typical DHT-based P2P system constructs an overlay topology in which every node plays the role of a client, a server, a router, and a domain name server. When nodes act as domain name servers - translating an identifier to the IP-address of a node that is responsible for it (see Property P4 in Section 2), the malicious nodes can potentially exploit this feature to misguide legitimate nodes with incorrect lookups. This could result in denial of information - a legitimate node is denied access to a data item; or sub-optimal performance of the lookup algorithm. For example, a malicious node can lie about the next hop when asked about the node id of a node responsible for a particular data item and return incorrect lookup results. In the absence of no alternate paths any malicious node along the only route can block all requests (misguide), thereby, increasing the probability of lookup failures. In case there are alternative routes, this attack could delay the routing of requests to correct nodes, thereby dramatically increasing the cost of a lookup operation.

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, [22] exploits the fact that: at each hop of the Chord lookup algorithm the query originator knows that each step in the lookup should lead him/her to get *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 incorrect lookup, the query originator can choose an alternative (possibly sub-optimal) path towards the destination identifier. Informally, a *lookup path* from node n to node m is the sequence of nodes through which a lookup operation succeeds. We call node n the source and node m the destination of the path. In the presence of malicious nodes the performance of a lookup algorithm 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.

4.1.1 Alternate Lookup Paths

We first highlight the importance of alternate (possibly sub-optimal) paths in enhancing the performance and robustness of the lookup algorithm in the presence of attacks. We cap-

ture the notion of *alternative 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 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 responsible for the key k; 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.

We use the *probability of lookup failure* as the 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 contains at least one malicious node. Intuitively, the number of independent lookup paths the 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 is 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 a 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 the 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 [1]. 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).

4.1.2 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 the 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 as the result. 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.

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

4.1.3 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 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 [22]. 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). Therefore, for our quantitative analysis we consider the following two extreme 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.

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 the most optimistic case for the scenarios described above.

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.

Scenario 1: Assuming that detecting an incorrect lookup is always possible and that there always exists an alternate-optimal path, $f_1(x)$ (the mapping function for scenario 1) satisfies the following recurrence relation.

$$f_1(x) = 1 + (1-p)f_1(x-1) + pf_1(x) \tag{2}$$

When a node n queries a node m for the next hop towards a key identifier k, node n expends one hop. Now, the success of the lookup depends entirely on whether node m is good or bad. If the node m is a good node (probability = $1-p$) then it would have returned 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 all nodes are non-malicious. In the presence of malicious nodes, this would require $f_1(x-1)$ hops by the definition of the function f_1. If the node m were a bad node (probability = p) then it would have returned an incorrect lookup. However, node n detects this and chooses an alternate-optimal path towards key k. Hence, node n is still x hops away from key k in the scenario where all nodes are good. In the presence of malicious 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 paths).

Solving the recurrence relation, we get,

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

Hence, the expected (average) number of hops required for a lookup operation is,

$$E[f_1(x)] = \frac{M}{1-p} \tag{4}$$

since, $M = E[x]$ by definition.

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

$$f_2(x) = 1 + (1-p)f_2(x-1) + pf_2(M) \tag{5}$$

Note that the first two terms in the expression for f_2 follows from the same arguments for scenario 1; 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 the bad nodes in terms of their bandwidth for answering repeated queries. Hence, we assume that the 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 5 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 6 show that this approximation is acceptable. Hence,

$$E[f_2(x)] \approx f_2(M) = \frac{1-(1-p)^M}{p(1-p)^M} \quad (6)$$

In order to make it easier to interpret this Equation we approximate the expression in Equation 6 for small values of p as follows:

$$E[f_2(x)] \approx \frac{M}{1-M*p} \quad (7)$$

Hence, the blow-up observed in the average lookup cost for scenario 1 is $\frac{1}{1-p}$ and for scenario 2 is $\frac{1}{1-Mp}$. Clearly, scenario 2 pays higher penalty for its inability to detect and recover from invalid lookups. Intuitively, in scenario 1, 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 $(1-p)^{-M}$.

4.1.4 Enhanced Invalid Lookup Detection Algorithm

The basic invalid lookup detection algorithm relies on the fact that: at each hop of the lookup algorithm the query originator knows that the lookup is supposed to get *closer* to the destination identifier. We enhance this strategy with an algorithm that checks whether a node m is indeed responsible for a given key k. If node m claims that it is responsible for key k, node n can verify if the key k is *reasonably close* to $ID(m)$, that is, $dist(ID(m),k) \leq thr$ for some distance threshold thr. This scheme has scope for both *false positives* and *false negatives*. A false positive occurs when a node n mistakenly believes that node m cannot be the node that is responsible for key k, when node m is actually responsible for key k ($dist(ID(m),k) > thr \wedge k \in Resp(m)$). A false negative occurs when a node n mistakenly believes that a node m is responsible for key k, when node m is actually not responsible for it ($dist(ID(m),k) \leq thr \wedge k \notin Resp(m)$).

Quantitative Analysis. Let Z be a random variable that denotes the distance between a key k and the node m that is responsible for key k. Let $f_Z(x)$ denote the probability distribution function (pdf) that the node p that is responsible for any key k is within a distance of x (on Chord's unit circle, $0 \leq x \leq 1$) from the identifier k, i.e, $dist(ID(p),k) \leq x$ and $k \in Resp(p)$. Assuming N denotes the total number of nodes in the system, one can show that

$$f_Z(x) = N*(1-x)^{N-1} \quad (8)$$

By the uniform and random distribution properties of the hash function the identifier of a node will be uniformly and randomly distributed between (0, 1). Hence, the probability that the identifier of any node falls in a segment of length x is equal to x. Equation 8 follows from the fact that the probability that there exists a node between distance x and $x+dx$ from key k is $N*dx$, and the probability that there is no other node within a distance x from key k is $(1-x)^{N-1}$. Using Equation 8 one can show that, $Pr(Z > \frac{thr}{N}) = e^{-thr}$.

Now, suppose that node n uses a threshold thr and agrees that node q is responsible for key k only if $dist(ID(p),k) \leq \frac{thr}{N}$. The probability of false positive $FP = Pr(Z > \frac{x_{thr}}{N})$ can be derived from the probability distribution function of the random variable Z. Hence,

$$FP(thr) = e^{-thr} \quad (9)$$

Now, we derive the probability of a false negative as a function of the threshold thr and p_m the fraction of malicious nodes known to node m. Intuitively, if the threshold thr were to be large, node m may be able to locate a malicious node q known to node m that is within the threshold. We derive the probability of a false negative FN as the probability that there exists a malicious node known to node m in segment of length $\frac{thr}{N}$. Following the same lines of arguments used to derive Equations 8 and 9 the probability of a false negative is given by Equation 10.

$$FN(thr,p_m) = 1 - e^{-thr*p_m} \quad (10)$$

Note that the results shown in Equations 9 and 10 are applicable to the case wherein node identifiers are chosen uniformly and randomly (and hence, it does not apply directly to CAN-RTT; CAN with RTT optimization). Observe that the probability of false positive exponentially decreases with the length of threshold segment and is independent of the fraction of malicious nodes in the system.

Figures 1 and 2 show the probability of false positives (fp) and false negatives (fp) for different values of threshold (thr) and the fraction of malicious nodes in the system (p). Note that 'fp' denotes the probability of false positive and 'fn-p' denotes the probability of false negative when $p\%$ of the nodes in the system are malicious. Figure 1 shows the results for $p_m = p$, that is, node m is aware of all the bad nodes in the system. Figure 2 shows the results for a more realistic (less pessimistic) scenario wherein we assume that only 20% of the malicious nodes collude ($p_m = \frac{p}{5}$). Observe from Figure 2 that for $thr = 2.5$, the false positive and the false negative probabilities are both under 0.1 even for $p = 20\%$. Hence, this demonstrates that a node can correctly decide on whether a lookup result is valid or not with a probability of 90%. Given the fact that fp is independent of p, in both Figures 1 and 2 we plot the fp curve to show the measurements obtained on how the probability of false positives varies over different values of threshold (thr).

Verifiable Result Caching. Caching is a popular technique to enhance the performance of any system. For instance, a *basic result caching* scheme for a DHT-based P2P system would cache lookup results at various nodes in the system; a node upon receiving a lookup request, may lookup its local cache and return appropriate cached results. However, a major drawback with the basic caching scheme is that a malicious node may misguide other nodes by returning invalid lookup results as if they were valid cached results. One could overcome this problem by employing a *verifiable result caching* scheme that uses the enhanced invalid lookup detection algorithm to identify (with fairly high probability) the correct lookup results from the incorrect/malicious ones. In short, one can use result caching to enhance both the performance of the system and the verifiability to protect the system from abuse by malicious nodes.

4.1.5 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, and verifiable result caching in hardening targeted attacks on the routing algorithm. Now we present four 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. The third set of experiments demonstrates the benefits of the verifiable result caching scheme. Finally, we discuss some extensions of concepts like alternate lookup paths to unstructured P2P networks.

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 4.1.1) when $p = 10\%$. Although the bounds obtained from our quantitative analysis are not very tight, the trends revealed by our analysis closely reflect the results obtained from our experiments. For instance, 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 a number of DHT-based P2P systems with varying number of independent lookup paths.

Figure 3 shows that the probability of lookup failure with $TTL = 100$ [2], $M = 5$ and varying p. Observe that 50% of the lookups fail in Chord when about 25% of the nodes are malicious, while a *4-D* CAN with 25% malicious nodes records less than 10% lookup failure. This amply verifies the fact that the probability of lookup failure increases steeply with the decrease in the number of independent paths.

Figure 4 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 P2P systems for the scenarios 1 and 2 discussed in section 4.1.3. Figure 5 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 a *one-sided error* of concluding that an incorrect (or a sub-optimal) lookup result as a correct result. The estimates from our quantitative analysis are labeled as 'check_Ehops' (Equation 4, Section 4.1.3); 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 5 in conjunction with the Figures 6 and 7. Figure 6 shows the average lookup cost for scenario 2 wherein, the legitimate nodes do not test the validity of the lookup results. The line labeled 'nocheck_Ehops' (Equation 6, Section 4.1.3) 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. Finally, Figure 7 shows the mean lookup cost in the presence of $p = 10\%$ malicious nodes for varying values of M (the mean number of hops in the absence of malicious nodes). Note that all failed lookups ter-

[2] TTL was set high enough to ensure that most of the alternate lookup paths are explored by the lookup algorithm

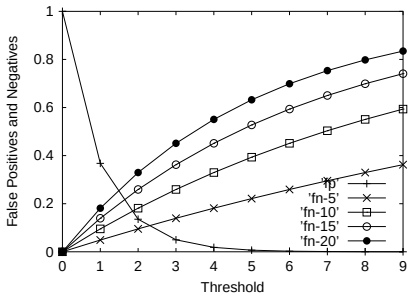

Figure 1: Probability of False Positives and False Negatives with 100% Collusion

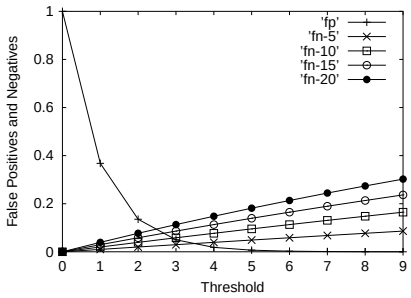

Figure 2: Probability of False Positives and False Negatives with 20% Collusion

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

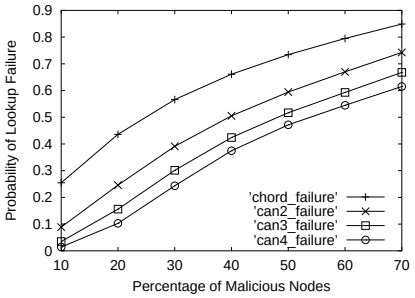

Figure 3: Probability of a lookup failure: Initial $TTL = 100$ and $M = 5$

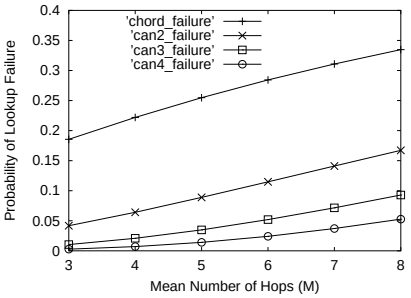

Figure 4: Probability of a lookup failure: Initial $TTL = 100$ and $p = 10\%$

minate when their TTL expires. Based on Figures 5 and 6, 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 and large values of M. 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 $p = 70\%$), the 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 6 shows the results only up to $p = 50\%$). Also, observe from Figure 7 that for $M \leq 8$ 'can8_nocheck' incurs about 30% lower lookup cost than 'can4_check'.

Importance of good alternate paths. Since 'can3_check' has multiple near-optimal alternate paths, its lookup cost is within thrice 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. The same argument also explains the better performance of 'can8_nocheck' over 'can4_check' in Figure 7 (for $M \leq 8$). Further, the vitality of alternate paths is more blatantly revealed by the figure from the fact that, 'chord_check' performs worse than 'can3_nocheck'.

Finally, the readers should keep in mind that it is indeed possible to strengthen the Chord or the CAN protocol to make up for certain deficiencies that we have pointed out in this paper. The readers should consider our viewpoint of these protocols as specific instances that were suitable for us to illustrate the key properties of a DHT-based P2P system.

Experiment III. In this experiment, we illustrate the performance enhancement that we achieve when verifiable pointer caching is employed by a DHT-based P2P system. Figure 8 shows the mean lookup cost for varying fractions of malicious nodes using the Chord lookup protocol on 1024 nodes under the following three scenarios: (i) No pointer caching, (ii) Basic pointer caching and (iii) Verifiable pointer caching. For this experiment, we chose the caching parameters as follows: the per-node cache size was set to 64 routing table entries (pointers) with a simple LRU based replacement strat-

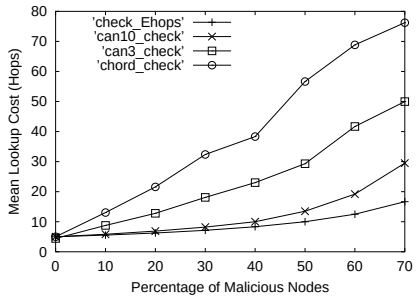

Figure 5: Lookup Costs: Scenario 1 with $M = 5$

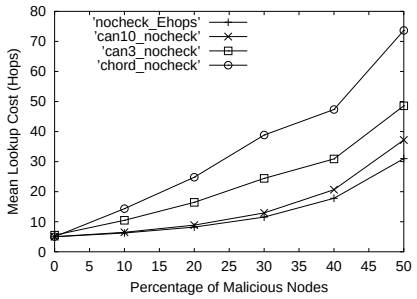

Figure 6: Lookup Cost: Scenario 2 with $M = 5$

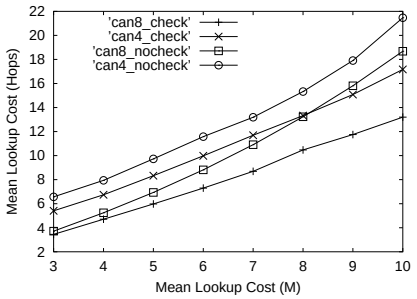

Figure 7: Lookup Costs Vs M: $p = 10\%$

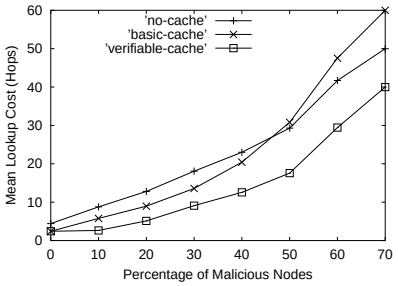

Figure 8: Verifiable Result Caching

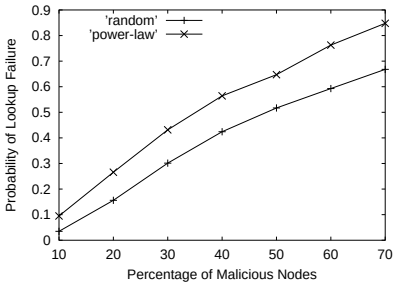

Figure 9: Unstructured P2P Networks

egy, and we employed path caching (a lookup result is cached at all nodes along its lookup path). With verifiable result caching in place, a malicious node is largely constrained to return correct lookup results. This significantly reduces the mean lookup cost by about 40-60% when compared to the basic caching scheme. Observe that in the presence of a large population of malicious nodes, the basic caching scheme breaks down; one could attribute this to the same reasons that cause the lookup algorithms with no checks (scenario 2 in section 4.1.3) to fail in the presence of large population of malicious nodes.

Experiment IV: Extensions to Unstructured P2P Networks. We briefly compare the structured and the unstructured P2P systems with respect to their susceptibility to routing anomalies. Unstructured P2P networks like Gnutella [8] use a broadcast-styled lookup algorithm, which has lot of redundancy built into it (and hence reduces the probability of lookup failure [3]) but largely increases the messaging traffic. One could build such redundancy into a DHT-based lookup protocol by broadcasting a lookup query along all alternate-optimal paths. Nevertheless, even when a broadcast-based lookup protocol is used, the notion of the number of independent paths plays a crucial role in determining the probability of a successful lookup. Figure 9 shows the probability of a successful lookup on a 1024 node unstructured P2P network built on a random topology with uniform node degree 3 and a random power-law topology with mean node degree 3 [19]. We picked 100 random pairs of nodes and measured the number of independent paths between them [4]. For a random topology, we observed that the mean number of independent paths was $ind_{random} = 2.91$ and for a power-law topology, the mean number of independent paths was $ind_{powerLaw} = 1.94$. From Figure 9, it is apparent that the random power-law topology pays with 40-70% higher probability of lookup failure for having a fewer number of independent paths. This experimental result assumes lot of practical relevance, since the actual topology of the Gnutella network is observed to be a random power-law topology [19].

4.2 Attacking the Data Placement Scheme

In this section, we show a mechanism through which the malicious nodes could corrupt the data stored in the P2P system. In particular, we quantify and analyze the extent to which malicious nodes can corrupt data and discuss plausible solutions to harden the system against such an attack.

Consider a DHT-based data storage system. A typical storage system built on a DHT-infrastructure incorporates replication as a mechanism to guarantee high availability of data. We argue that though replication is a very good mech-

[3] A malicious Gnutella node simply drops all incoming query request without broadcasting them.

[4] By Menger's theorem [14], the number of independent paths equals the vertex connectivity of a graph; and vertex connectivity can be measured using network flow techniques [15]

anism to ensure high reliability and high availability of data, replication *alone* is insufficient to tolerate attacks by malicious nodes.

We illustrate this scenario in detail in the context of a distributed P2P trust management system. A P2P trust management system such as [1, 25] provides schemes to estimate the trustworthiness of any node in the P2P system. Trust information about every node is replicated on several other nodes with a hope that using a majority decision drawn from the trust values reported by all the replicas could defend malicious behavior of some of the replica holders. Also, a trust management system usually assigns a low (default) trust value to a newly joining node [25]. Hence, the good nodes typically stay longer in the system to preserve the reputation they have earned so far, whereas the bad nodes would enter and leave the system more frequently to erase the poor reputations they have had. Now we show that replication and majority voting alone cannot secure a trust system from malicious nodes.

Let G be the number of good or legitimate nodes, B be the number of bad or malicious nodes, R be the number of replicas maintained by the system for every data item (usually, $R \ll B < G$) and N_{EID} denote the number of external identifiers owned by each node. Recall that in the adversary model (section 3) we assume that every bad nodes can *own* several EIDs, the bad nodes can *decay* the data stored in the system through the following simple strategy: The bad nodes repeatedly join and leave the system each time using a random EID (from the pool of EIDs they own) and corrupt the data assigned to them. Choosing different EIDs enables the bad nodes to be assigned entirely different node identifiers each time (assuming uniform random properties of the hash function), thereby corrupting different data sets each time (see DHT properties P2 and P4 outlined in Section 2). On the other hand, the good nodes attempt to validate data before they accept data from any leaving node by using a majority decision (see property P6 in Section 2). We use quantitative analysis and experiment validation to show that replication and majority voting alone are insufficient to guard a P2P system against such an attack.

4.2.1 Quantitative Analysis

Let cr be the corruption threshold, namely, the number of replicas that are required to be corrupted so as to ensure that the data item cannot be recovered. Note that cr depends on the properties of the data item stored. First, we argue that in the context of a trust management system, cr should be equal to $\lceil R/2 \rceil$ (a simple majority decision). Most trust systems like [1, 25] allow nodes to transact with each other and submit an evaluation of the transaction to R replica nodes. Also, the trust value for a node is evaluated incrementally, and does not change dramatically on the addition of a new report. Given this property of the trust value, one need not worry much about inconsistent updates at different replicas (some replicas might have received more recent reports), since it anyway does not significantly alter the node's trust value. One could simply use the median of the trust values reported by the replicas as the actual trust value of the node. Hence, the only way cr incorrect trust values could radically alter the median of R trust values (given that the remaining $R - cr$ trust values are approximately the same) is for $cr \geq \lceil R/2 \rceil$.

On the other hand, for data items whose updates can significantly affect their values, one would have to choose $cr \geq \lceil R/3 \rceil$, based on the famous Lamport's Byzantine generals problem and Byzantine quorum systems [12, 13]. For encrypted and authenticated data, the corruption threshold cr equals R, since any data item can be successfully recovered as long as there exists one uncorrupt copy of the data item in the system. In short, the hardness of corrupting a replicated data item (cr) depends on the nature of the data item.

We estimate the extent of damage caused by malicious nodes in two steps. First, we derive the extent of data corruption that can be caused by malicious node at any given snapshot or time instant. Then, we estimate the fraction of data items that get corrupted over a long period of time. Just for the simplicity of the analysis assume that all the bad nodes operate in lockstep; that is, all of them join the system at the same time (or within a short interval) and all of them corrupt data and leave the system at the same time (or within a short interval). Therefore, when all the bad nodes are a part of the system, the system consists of G good nodes and B bad nodes. Usually, the replicas for a data item d is located at identifiers $\{hash(d \parallel 0), hash(d \parallel 1), \cdots, hash(d \parallel R-1)\}$ [5]. Since a DHT-based scheme divides the identifier space uniformly among all the nodes, the probability that any given data item d is stored at a malicious node is $\frac{B}{B+G}$. At this time instant, the probability that a data item d cannot be recovered is the probability that at least cr replicas of the data item d are stored at bad nodes, denoted by $Pr_{decay}(B, G)$. It can be defined as follows:

$$Pr_{decay}(B, G) = \sum_{q \geq cr}^{R} binom(\frac{B}{B+G}; q, R) \qquad (11)$$

Where $binom(p; q, R)$ denotes the probability in a binomial distribution for q successes from R trials where the probability of success in any trial is p.

As bad nodes join and leave the system, they corrupt data items stored at different portions of the identifier space. Given B identifiers, the malicious nodes can corrupt $Pr_{decay}(B, G)^{th}$ portion of the identifier space. Let the number of EIDs owned by a bad node be denoted by

[5] Note that there is a small probability that two or more replicas hash to the same physical node

N_{EID}. The number of data items that would be decayed after *many* iterations by the malicious nodes is equivalent to the number of data items that would have decayed if there were $B_{eff} = N_{EID} * B$ bad identifiers in the system. Thus the number of data items that are eventually corrupted (after possibly infinite rounds by the malicious nodes) can be estimated by Equation 12.

$$Pr^{\infty}_{decay}(B, G) = Pr_{decay}(B_{eff}, G) \quad (12)$$

There are three remarks on the above equations. First, the probability that at least cr replicas of the data item d are stored at bad nodes in a given round (Equation 11), denotes the probability that a data item survives in this round of bad nodes joining/leaving the system. Second, the probability that a data item survives across multiple rounds of bad nodes joining/leaving depends on how EIDs are chosen by the bad nodes in a given round. More specifically, if the malicious nodes do not choose their EIDs randomly and independently (from their pool of EIDs) then Equation 12 may not be a good approximation for the expected number of nodes that will not survive the attack. In other words, if the EIDs $\{EID_1, EID_2, \cdots, EID_{cr}\}$ are required to corrupt a data item d, then at *some round* a set of cr malicious nodes must choose their EIDs as $\{EID_1, EID_2, \cdots, EID_{cr}\}$ respectively. Third, we have assumed that the good nodes do not enter or leave the system frequently. Observe that in a more realistic scenario, when the good nodes enter and leave the system, the expected number of data items that can survive this attack would be even smaller than the result obtained from Equation 12. Also, note that if a node were allowed to choose its node identifier then N_{EID} is practically infinite and hence all the data items in the system would eventually decay.

4.2.2 Experimental Validation

We have so far identified and quantitatively analyzed the importance of the corruption threshold, the number of replicas, the fraction of malicious nodes, and the number of EIDs in hardening targeted attacks on the data placement scheme. Now we present two sets of experiments. The first set of experiments demonstrate the data placement scheme attack and show the rate at which data items are corrupted in the system. The second set shows the relationship between the nature of the data item and the effort required to corrupt it.

Experiment I. In this experiment, we demonstrate the data placement scheme attack. We simulated the working of a P2P system using the Chord lookup protocol [23] with 1024 good nodes. We allow each node a maximum of 20 EIDs. The results from our simulation are shown in Figures 10 and 11. Figure 10 shows the rate at which information decays for different populations sizes of the malicious nodes (using $R = 7$ replicas for each data item). Observe that with $p = 20\%$, 50% of the data items in the system is corrupted in less than 20 iterations. Figure 11 illustrates the rate of information decay for different values of R (number of replicas) for an extremely small population of malicious nodes = 2% of the system size. Observes that even when 11 replicas ($R = 11$) are maintained for every data item, the malicious nodes can corrupt 15% of the data items in about 125 iterations. The results obtained from our quantitative analysis (Equation 12) closely matches the experimental results for the fraction of corrupted data items at $time = 600$. Also, the initial decay rate ($time < 20$) closely matches our estimate from Equation 11; For example, from Figure 10 for $\%bad = 30$, the initial decay rate from our experiments is about 12.3% per unit time and that from our quantitative analysis is about 12.5% per unit time. For intermediate time intervals, the decay rate falls below 12.5% per unit time since the probability of data item decay in any two iterations are no longer independent. These results clearly indicate that information decays very rapidly in course of time. Note that a trust management system becomes *useless* when more than $B/(B + G)$ of the trust values are incorrect. In fact, a large fraction of incorrect trust values backfires on the system since the bad nodes would be considered good and vice-versa. Note that in the absence of a trust management system, even when a node makes a random choice on other nodes with whom it can transact, the probability of choosing a bad node is only $B/(B + G)$.

Experiment II. In this experiment, we show the relationship between the nature of a data item and the effort required to corrupt it. Figure 12 shows the number of data items that are eventually corrupted in the P2P system for different values of N_{EID} for three data protection characteristics: cryptographically secure schemes ('encrypt), trust systems ('trust-majority-vote) and Byzantine read-write quorums ('rw-quorums). From Figure 12 is apparent that the cryptographic schemes ($cr = R$) shows much higher tolerance to malicious nodes than a trust systems with majority vote based data recovery ($cr = \lceil R/2 \rceil$), which in turn is better than a simple Byzantine group based read-write quorum ($cr = \lceil R/3 \rceil$). Interestingly, our analytical results match the experimental results very closely (with a standard deviation of the order of 10^{-2}); hence, for the sake of clarity we removed the analytical results from Figure 12.

4.2.3 Defense

The key problems using the replication-based security scheme are: (i) When the trust values are stored in plain-text, the bad nodes can toggle the trust values to make good nodes appear bad and vice-versa, and (ii) When a data item gets corrupted irrecoverably, it can never be detected (or corrected). We can solve this problem by using cryptographic tools to provide confidentiality and authentication guarantees. This

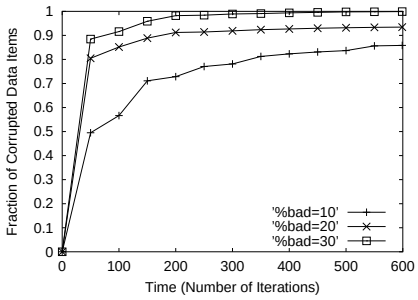

Figure 10: Decay rate for $R = 7$

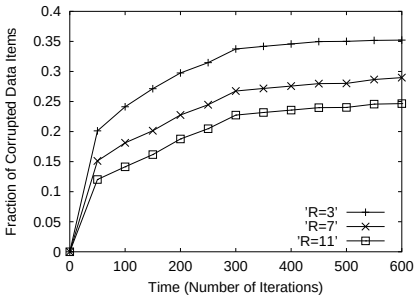

Figure 11: Decay rate for $\%bad = 10$

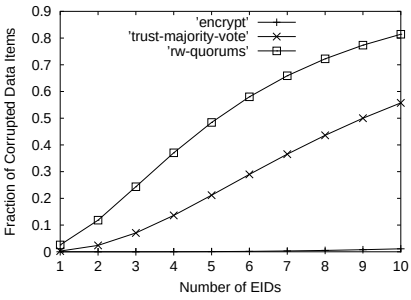

Figure 12: Corruption of Different Data Types: $\%bad = 10$ and $R = 7$

requires any transaction performed in the system to be signed by all the parties involved in the transaction. This solution clearly improves the corruption threshold cr to R, since any data item can be recovered as long as at least one signed copy of the data item survives and the decreases the probability of decay of any data item to,

$$Pr_{decay}(B, G) = \left(\frac{B}{B+G}\right)^R \tag{13}$$

Observe that using the same analysis as described in Appendix B, the bad nodes can repeatedly join and leave the system and amplify the effective number of bad nodes B_{eff} to $B * N_{EID}$. Further, it seems that information does decay using this scheme, though to a much lower extent, when compared to the case wherein the data items are not encrypted. However, by cryptographically signing data items, we prevent good nodes from being misguided by a corrupt data item. Hence, the fact that the legitimate nodes can figure out whether a particular data item is indeed corrupted, at least protects them against such invalid data items. Further, when a good node is assigned a corrupted data item, it can take corrective measures on the data item. In the context of a trust system, a good node may reinitialize corrupted trust values to a base (default) value. Note that such corrective measures do not always exist; in say the file sharing scenario, there is no useful default value that can be associated with a file, in the event that all its copies are corrupted.

Figure 13 shows the efficacy of a trust management in terms of the percentage of successful transactions in the P2P system. We compare three versions of the trust management system: (i) 'majority-vote'-a basic majority vote based trust management system, (ii) 'encrypt-nodefault'-a cryptographically secure trust system, and (iii) 'encrypt-default'-a cryptographically secure trust system that assigns a default value to any corrupted trust value. Figure 13 shows that the trust management system gains significantly from cryptographic techniques; further, the luxury of having a default value to reset any corrupted data item improves the mean transaction success rate by a little over 5% (when compared to the 'encrypt-nodefault' case).

Figure 14 shows the fraction of data items that are eventually corrupted versus the number of replicas maintained by the system using the majority-vote based trust management system ('majority-vote') and a cryptographically secure system ('encrypt'). Note that with $N_{EID} = 10$ and the percentage of bad nodes = 5% and 20% of the number of good nodes (G), the effective percentage of bad nodes turns out to be $B_{eff} = 50\%$ and 200% of the number of good nodes respectively. Observe that the majority vote based algorithm *fails* when $B_{eff} > G$, that is, with increasing number of replicas the percentage of corrupted data items *increases*. However, using a cryptographically secure system, one can *always* reduce the percentage of corrupted data items by increasing the number of replicas (recall Equations 12 and 13).

So far, we have demonstrated the ability of the cryptographic techniques (over the basic replication) to secure the system from the data placement scheme attack. However, using cryptographic solutions in a large scale distributed system brings in issues such as key management and performance overheads. CFS [5] and Farsite [2] present solutions to this problem in the context of a distributed file system. In CFS (a distributed read-only file system), the root block is identified by a public-key and signed by the corresponding private key; the other blocks are identified by cryptographic hashes of their contents. In Farsite, a directory is implemented on a collection of machines using a Byzantine-fault-tolerant protocol and file contents are stored encrypted on file hosts.

In addition to these techniques, one can protect data items stored in the system by *hiding* the keys used to encrypt them. Say, the owner of a data item d encrypts it using a key k and stores the key at a location identified by some random identifier $rand\_id$. The owner distributes $rand\_id$ only to those nodes that have the right to access the data item d. The node that is responsible for the identifier $rand\_id$ (say, node n) returns the key k when some node (say, node m) presents the secret identifier $rand\_id$. Assuming that $rand\_id$ is hard to guess (sufficiently long), only the nodes that have been

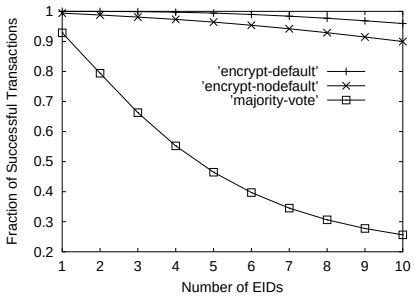

Figure 13: Fraction of Successful Transactions Vs Number of EIDs ($\%bad = 10$ and $R = 7$)

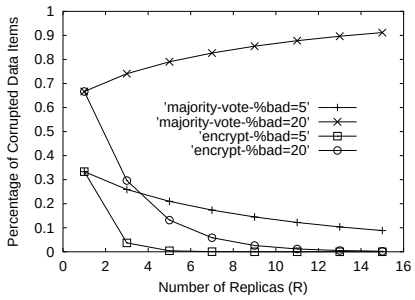

Figure 14: Fraction of Corrupted Data Items Vs Number of Replicas ($N_{EID} = 10$

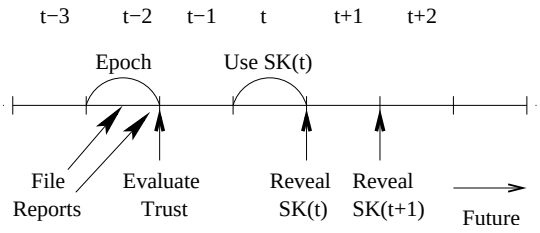

Figure 15: Trust Evaluation Mechanism

permitted access to the data item d can retrieve its key (if node n were honest). However, node n may misuse the key k to illegally gain access to data item d. One could get rid of complete dependence on a single node by storing the key at r identifiers, say $\{rand\_id_1, rand\_id_2, \cdots, rand\_id_r\}$. However, replicating the key k at r different nodes [6] does not help (in fact, it makes the situation worse); hence we need to divide the key k into r secret shares [21] and store one share at each of these r nodes. If mal denotes the number of malicious nodes (in the set of r nodes) and $mal < thr \leq r$ is the threshold used for constructing secret shares, and $mal \leq r/3$ then the key k would be secure [12].

The cryptographic techniques described above are suitable for cases wherein access to a collection of data items had to be limited to a group of nodes in the system. However, in other scenarios like the trust management system, a report about a transaction can be filed by *any* node. In such scenarios, it is very important to preserve the authenticity and non-repudiability of the reports. Hence, one would have to rely on public-key cryptography based algorithms to provide the required level of security (a shared symmetric key neither provides authentication or non-repudiation, since any node that knows the key could have generated the report). However, a 1024-bit RSA [11] signature costs about 8ms; and signature verification costs about 1ms on 900 MHz Pentium III processor using the standard OpenSSL library [17]. But one could use several interesting properties of a trust management system to avoid using public-key cryptography. Most trust evaluation mechanisms operate in epochs (rounds); all the reports filed in a round is evaluated (by incompletely trusted agents) only at the beginning at the next round (for various performance reasons). Further, every report is used exactly once, to evaluate the new trust value of nodes at the beginning of each epoch. For an illustration of a typical trust evaluation mechanism see Figure 15. Hence, at epoch t, a node n may encrypt all its reports using a cheap symmetric key encryption algorithm (DES [7] is about 1000 times faster than RSA [11]) with key $SK_n(t)$. At the beginning of the next epoch $t + 1$, the node n authenticates its reports by sending $SK_n(t)$ to the incompletely trusted trust evaluation agents; and, node n uses a new key $SK_n(t + 1)$ for filing reports in epoch $t + 1$. The trust evaluation agents may use a Byzantine-fault-tolerant protocol to evaluate the trust value of nodes; however, they cannot misuse node n's symmetric key $SK_n(t)$ after node n reveals it at epoch $t + 1$ since, node n would anyway not reuse the key $SK_n(t)$ in the future epochs (see Figure 15 for a graphical illustration).

Although using encryption/decryption mechanism secures the system to some extent as we have shown in this subsection, we believe that it is helpful to develop an analytical model for the "amount of security that one can obtain and compare it with experimental results. Our research and experiments on these cryptographic techniques are ongoing. Due to space restrictions we skip the analysis and experimental results on these techniques.

4.3 Attacking the ID-to-Key Mapping Scheme

The Sybil attack paper [6] shows that there is really no scalable solution to prevent the malicious nodes from forging identities. In this section, we present the ID-to-Key mapping attack (ID Mapping for short) - a major vulnerability in the DHT-based systems as a direct effect of the Sybil attack. We show how the malicious nodes could exploit the peer identifier-to-key mapping to corrupt a specific data item stored in the P2P system. We quantify and analyze the cost of attacking a specific data item and discuss some approaches to mitigate this problem.

Almost all DHT-based system use a strong one-way hash function (like MD5 [16] or SHA1 [20]) to derive a node identifier from its EID; and, any node p has direct access to a data item d if and only if $d \in Resp(p)$ (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

[6] Note that we ignore the small probability that two or more random identifiers hash into the same 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 [24], 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 specific 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 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, $d \in Resp_t(q)$. A malicious node p gains access to the target data item d through the following two steps: 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 $ID(n) \in Resp_t(q)$ 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 d thereby ensuring that the target data item d is assigned to it.

4.3.1 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 m is $1/G$. The probability that a malicious node hits upon an EID that hashes into bucket m in its k^{th} attempt is given by,

$$Pr(k\ attempts) = \left(1 - \frac{1}{G}\right)^{k-1} \frac{1}{G} \quad (14)$$

where the first $k-1$ attempts fails to fall into bucket m, while the k^{th} attempt succeeds. Observe that the number of attempts required in Equation 14 follows a Geometric Distribution. The malicious nodes could choose m such that their target data item d is currently held by node m, i.e., $d \in Resp_t(m)$. It follows from Equation 14 that the expected number of attempts required to obtain an identifier that hashes into node m 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 m's identifier space does not guarantee that the malicious node n is assigned the target data item d. (Note $Resp_{t'}(n)$ and $Resp_{t'}(m)$ partitions $Resp_t(m)$, for $t' > t$ (see property P6 in Section 2). Since the data item d can lie anywhere in the identifier space assigned to node m, the probability that node n gets to store data item d after it joins the network is $\frac{1}{2}$. Hence, with a reasonably large probability, a malicious node n can obtain access to a target data item d in G attempts. In fact,

$$Pr(Number\ of\ attempts \leq G) = \frac{1}{2}\left(1 - \frac{1}{e}\right) \quad (15)$$

Consequently, one can improve the chances of this targeted attack in $O(G)$ attempts since,

$$Pr(Number\ of\ attempts > cG) = \left(\frac{1}{2}\left(1 + \frac{1}{e}\right)\right)^{c} \quad (16)$$

for some integer $c > 0$.

In a P2P 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 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).

4.3.2 Experimental Validation

We simulated the Chord lookup protocol [23] 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 16): $Pr(N_{EID} \leq 4G) = 0.79$ and $Pr(N_{EID} \leq 8G) = 0.96$.

G	Average time (millisec)	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

4.3.3 Defense

The Sybil attack paper [6] suggests the use of trusted certification authorities to strictly bind an identity to an entity (node). However, employing trusted certification authorities turns out prohibitively expensive. In order to reduce certification costs, one could employ weak secure identifiers, like the node IP-address, which can be challenged and verified easily.

Yet another approach to solve this problem would be to mitigate it as follows. 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 this solution, the ID for a node is not derived from any of the EIDs owned by the node. Instead, the bootstrap server assigns a random $id \in S$ to the node (and issues a certificate with a short life-time) when a node 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 16. 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* [7] to join the system.

The key advantage of using EIDs to derive node identifiers is that every node has a unique identity in the system as long as the node continues to use the same EID. Hence, the system can maintain *persistent* information about the node across multiple joins/leaves by that node. For example, the trust value of a node could be retained in the system even after the node leaves the system (provided the node uses the same EID). In comparison, the bootstrap server based random node ID generation technique, maintains complete anonymity of a node (does not even require an EID) but is incapable of maintaining persistent node information. But, when the number of EIDs that could be owned by a node (N_{EID}) is very large it is more desirable to use non-EID dependent techniques with weak checks by the bootstrap server, since it at least avoids the *cheap* offline attack.

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

5 Related Work

Sit and Morris [22] discuss several security considerations in a distributed hash table based P2P system. They present a framework for performing security analysis of P2P networks and discuss a wide range of possible attacks including: routing layer attacks like node lookup, 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. It also suggests using replication to handle storage and retrieval attacks. Our work not only quantifies the lookup costs using repeated checks but also shows that merely performing repeated checking does not help especially in the absence of multiple independent lookup paths. Also, we showed that replication is a good tool for ensuring high reliability and availability of data; but replication alone is incapable of securing data.

Castro *et al.* [4] discuss several issues on secure routing on DHT based P2P systems. 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 of multiple near optimal independent paths is applicable to the case where redundant routing is used. Also observe that having multiple independent paths improves the probability of success in a smaller number of hops in both repeated checking and redundant routing techniques.

The Sybil attack paper [6] 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 concludes that for direct or indirect validation, a set of faulty nodes can counterfeit an unbounded number 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 out prohibitively 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 4.2 and 4.3) assessed the risks and security threats when such weak secure IDs are deployed in a DHT-based P2P system.

6 Conclusion

We have studied vulnerabilities of the overlay network layer in a DHT-based P2P system through investigating three tar-

geted 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, and verifiable pointer caching in hardening the routing (lookup) algorithm. Second, we described an attack on the Data Placement Scheme and showed that replication is good for high reliability and availability; but replication in conjunction with encryption tools is essential for security. Third, we discussed an attack on the ID Mapping Scheme and showed that attacks on specific 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 present the quantitative analysis to estimate the extent of possible damages caused by these three types of attacks and use the 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 (coupled with a secure physical network layer and a trust-support enabled application layer) will provide a secure infrastructure for large-scale decentralized P2P applications.

References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the 10th International Conference of Information and Knowledge Management*, 2001.

[2] 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.

[3] 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.

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

[5] 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.

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

[7] FIPS. Data encryption standard (des). http://www.itl.nist.gov/fipspubs/fip46-2.htm.

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

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

[10] 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.

[11] R. Laboratries. Rsa cryptography standard. http://www.rsasecurity.com/rsalabs/pkcs/.

[12] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. In *IEEE Computer Society Press*, 1982.

[13] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Springer Verlay (Distributed Computing)*, 1998.

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

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

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

[17] OpenSSL. Openssl. http://www.openssl.org/.

[18] 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.

[19] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Technical Report TR-2001-26, University of Chicago, 2001.

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

[21] A. Shamir. How to share a secret? In *Communications of the ACM*, 1979.

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

[23] 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.

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

[25] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *IEEE Conference on E-Commerce (CEC'03)*, 2003.