Eclipse Attacks on Overlay Networks: Threats and Defenses

Atul Singh*, Tsuen-Wan "Johnny" Ngan*, Peter Druschel†, and Dan S. Wallach*
*Department of Computer Science, Rice University
†Max Planck Institute for Software Systems

***Abstract*— Overlay networks are widely used to deploy functionality at edge nodes without changing network routers. Each node in an overlay network maintains connections with a number of peers, forming a graph upon which a distributed application or service is implemented. In an "Eclipse" attack, a set of malicious, colluding overlay nodes arranges for a correct node to peer only with members of the coalition. If successful, the attacker can mediate most or all communication to and from the victim. Furthermore, by supplying biased neighbor information during normal overlay maintenance, a modest number of malicious nodes can eclipse a large number of correct victim nodes.**

This paper studies the impact of Eclipse attacks on structured overlays and shows the limitations of known defenses. We then present the design, implementation, and evaluation of a new defense, in which nodes anonymously audit each other's connectivity. The key observation is that a node that mounts an Eclipse attack must have a higher than average node degree. We show that enforcing a node degree limit by auditing is an effective defense against Eclipse attacks. Furthermore, unlike most existing defenses, our defense leaves flexibility in the selection of neighboring nodes, thus permitting important overlay optimizations like proximity neighbor selection (PNS).

I. Introduction

Overlay networks facilitate the deployment of distributed application functionality at edge nodes without the need to modify existing network infrastructure. Overlays serve as a platform for many popular applications, including content distribution networks like BitTorrent, CoDeeN, and Coral [10], [16], [40], file-sharing systems like Gnutella, KaZaA, and Overnet/eDonkey [18], [23], [30] and end-system multicast systems like ESM, Overcast, NICE, and CoolStreaming [1], [8], [22], [42]. Moreover, a large number of research projects explore the use of overlays to provide decentralized network services [25], [31], [33], [38], [43].

Robust overlays must tolerate participating nodes that deviate from the protocol. One reason is that the overlay membership is often open or only loosely controlled. Even with tightly controlled membership, some nodes may be compromised due to vulnerabilities in operating systems or other node software [44]. To deal with these threats, overlay applications can rely on replication, self-authenticating content [26], Byzantine quorums [24], or Byzantine state machines [7] to mask the failure or corruption of some overlay nodes.

In an overlay network, each node maintains links to a relatively small set of peers called *neighbors*. All communication within the overlay, be it related to maintaining the overlay or to application processing, occurs on these links. The overlay's integrity depends on the ability of correct nodes to communicate with each other over a sequence of overlay links. In an *Eclipse attack* [5], [37], a modest number of malicious nodes conspire to fool correct nodes into adopting the malicious nodes as their peers, with the goal of dominating the neighbor sets of all correct nodes. If successful, an Eclipse attack enables the attacker to mediate most overlay traffic and effectively "eclipse" correct nodes from each others' view. In the extreme, an Eclipse attack allows the attacker to control all overlay traffic, enabling arbitrary denial of service or censorship attacks.

The Eclipse attack is closely related to the Sybil attack [14], where a single malicious node assumes a large number of different identities in the overlay. Clearly, a successful Sybil attack can be used to induce an Eclipse attack. However, Eclipse attacks are possible even in the presence of an effective defense against Sybil attacks, such as certified node identities [5]. In a decentralized overlay, nodes periodically discover new neighbors by consulting the neighbor sets of existing neighbors. Malicious nodes can exploit this by advertising neighbor sets that consist of only other malicious nodes. Thus, a small number of malicious nodes with legitimate identities is sufficient to carry out an Eclipse attack.

Castro et al. identify the Eclipse attack as a threat in structured overlay networks [5]. To defend against this attack, they propose the use of *Constrained Routing Tables* (CRT), which imposes strong structural constraints on neighbor sets. In this defense, nodes have certified, random identifiers and a node's neighbor set contains nodes with identifiers closest to well-defined points in the identifier space. The certified identifiers prevent Sybil attacks, and the CRTs thwart Eclipse attacks. However, this defense leaves no flexibility in neighbor selection and therefore prevents optimizations like *proximity neighbor selection (PNS)* [6], [20], an important and widely used technique to improve overlay efficiency.

This paper presents a defense against Eclipse attacks based on anonymous auditing of nodes' neighbor sets [35]. If a node has significantly more links than the average, it might be mounting an Eclipse attack. When all nodes in the network perform this auditing routinely, attackers are discovered and can be removed from the neighbor sets of correct nodes. The defense is applicable to homogeneous structured overlays; experimental results indicate that it is highly effective and efficient for overlays with low to moderate membership churn, i.e., with session times on the order of hours.

The rest of this paper is organized as follows. In the next section, we provide some background on overlay networks and their vulnerability to the Eclipse attack and present existing defense mechanisms. Section III describes our proposed defense. In Section IV, we discuss the auditing technique necessary to implement our defense. Section V presents experimental results on the impact of Eclipse attacks, the limitations of existing defenses, and the effectiveness of our auditing technique. Section VI discusses the results, Section VII covers related work and Section VIII concludes.

II. Background

In this section, we provide some background on overlay networks, and we discuss prior work in making overlays robust to Eclipse attacks.

Overlay networks consist of a set of nodes connected by a physical network like the Internet. Each node in an overlay maintains a neighbor set, consisting of a set of peer nodes with which the nodes maintains connections. The union of all participating nodes and their neighbor relations form an *overlay graph*.

In decentralized overlays, nodes receive membership information from their peers. If a malicious node advertises a neighbor set that consists only of other malicious nodes, it may bias the node selection of correct nodes. Correct nodes will in turn pass along this biased membership information to their peers. Over time, the malicious nodes attract a lot of correct neighbors, unless an appropriate defense is in place.

Eclipse attacks could easily be prevented by using a trusted, centralized membership service (e.g., the tracker in BitTorrent [10]). Such a service keeps track of the overlay membership and offers unbiased referrals among nodes that wish to acquire overlay neighbors. However, such a service is undesirable in many environments, since it requires dedicated resources and raises concerns about scalability, reliability and availability.

A. Unstructured Overlays

Unstructured overlays (e.g., [10], [18], [23]) impose no constraints on neighbor selection. Typically, a joining node starts from a bootstrap node and discovers additional neighbors by performing random walks in the overlay. Malicious nodes can trivially bias the neighbor selection of a correct nodes by steering random walks towards other malicious nodes. As such, unstructured overlays are vulnerable to Eclipse attacks.

B. Structured Overlays

Structured overlay networks (e.g., [25], [31], [33], [38], [43]) maintain a specific graph structure that enables reliable object location within a bounded number of routing hops. Generally, each node is assigned a unique, pseudo-random identifier from a large numeric space, e.g., the set of positive 160-bit integers. A node selects its neighbors from the set of nodes whose identifiers satisfy certain constraints relative to its own identifier. The constraints differ for different neighbor set slots and some are more restrictive than others. For example, the neighbor sets in Tapestry and Pastry form a matrix, where the i^{th} row refers to nodes whose ids share the a prefix of i digits, and the j^{th} column contains nodes whose $(i+1)^{\text{th}}$ digit in their ids is j. Thus, the identifier of a node determines its eligibility as a neighbor for any given other node. Obviously, nodes are more flexible in choosing neighbors for the low numbered rows since they require shorter prefix matches. It is this flexibility in the choice of neighbors that malicious nodes can exploit to mount an Eclipse attack on structured overlays.

C. Existing Defenses

Decentralized defenses against the Eclipse attack have been proposed that require additional constraints on neighbor selection. They fall into two categories: structural constraints and proximity constraints.

1) Stronger structural constraints: Overlays like CAN [31], the original Chord [38], and Pastry with a constrained routing table (CRT) [5] impose strong structural constraints on the neighbor sets. Each neighbor set member is defined to be the overlay node with identifier closest to a particular point in the identifier space. This constraint defeats Eclipse attacks under two conditions:

First, each node has exactly one unique, unforgeable identifier. This can be accomplished, for instance, by a trusted, off-line authority that issues cryptographically signed identifiers [5], thereby preventing Sybil attacks. Second, the overlay has a mechanism to locate the live node whose id is closest to a desired point in the id space. This mechanism ensures that a query for a randomly chosen id has about the same chance of yielding a malicious node as the probability that a randomly selected node is malicious. Castro et al. [5] described a routing primitive that accomplishes this through a combination of trusted id certification, id density tests, and redundant routing probes using the constrained routing tables. The main drawback of using strong structural constraints to defend against Eclipse attacks is that it removes flexibility in neighbor selection, preventing important overlay optimizations like PNS. Moreover, the secure routing primitive has significant overhead.

2) Proximity constraints: Hildrum and Kubiatowicz [21] describe a different defense against the Eclipse attack based on proximity neighbor selection. Each node selects as its neighbors the nodes with minimal network delay, among all the nodes that satisfy the structural constraints for a given neighbor set member. Since a small number of malicious nodes cannot be within a low network delay of all correct nodes, it is therefore difficult for them to mount an Eclipse attack.

This defense assumes that the delay measurements cannot be manipulated by the attacker. Also, it is effective only if pairs of nodes are sufficiently separated in the delay space. For instance, if the resolution of delay measurements is 1ms and a large fraction of overlay nodes are within 1ms of each other, then the defense is not effective. Indeed, it was observed that from the perspective of a typical Internet host, a large number of other nodes appear within a narrow band of delay [20]. Moreover, our experimental results in Section V-B

show that with realistic delay values measured in the Internet, the effectiveness of the PNS-based defense diminishes with increasing overlay size.

In summary, we observe that maintaining strict structural constraints provides an effective defense against Eclipse attacks, but it introduces additional overhead and prevents important performance optimizations like PNS. Network proximity based defenses, on the other hand, depend on accurate, high-resolution delay measurements and they may be effective only for small overlays.

III. Enforcing Degree Bounds

In this section, we describe a new defense against Eclipse attacks based on enforcing node degree bound. Our technique requires that each participating node carries a certificate, binding its node identifier to a public key. It is further assumed that the overlay supports a secure routing primitive using a constrained routing table (CRT), as described in the previous section.

A. Overview

The defense is based on a very simple observation: During an Eclipse attack, the in-degree of attacker nodes in the overlay graph must be much higher than the average in-degree of correct nodes in the overlay. Thus, one way to prevent an Eclipse attack is for correct nodes to choose neighbors whose in-degree is not significantly above average, among the set of nodes that satisfy any structural constraints imposed by the overlay protocol.

In the general case of an overlay where the neighbor relation is not reflexive, it is not sufficient to bound node in-degrees. Malicious nodes could try to consume all the in-degrees of correct nodes, thereby preventing correct nodes from choosing other correct nodes as their neighbors. Consequently, it is also necessary to bound the out-degree of nodes. Correct nodes choose neighbors whose in-degree and out-degree are below a certain threshold.

Next, we show that if each overlay node has the same degree, then the expected fraction of malicious nodes in the neighbor set of correct nodes can be bounded by $f/(1-f)$, where f is the fraction of malicious nodes in the overlay.

Let the expected out-degree of correct nodes be O_{exp}, the size of the neighbor set required by the overlay protocol. With $N(1-f)$ correct nodes, their total out-degree is $N(1-f)O_{\text{exp}}$. We further assume that we can successfully bound the in-degree of any given node, malicious or correct, to $I_{\max} = tO_{\text{exp}}$, for some constant $t \geq 1$. There are Nf malicious nodes, and so the total in-degree of malicious nodes is at most $NfI_{\max}$. The Eclipse attack is most effective when most of the out-degree of correct nodes are consumed by malicious nodes. Let f' be the fraction of out-degree of correct nodes consumed by malicious nodes. Then $f'N(1-f)O_{\text{exp}} \leq NfI_{\max}$, and $f' \leq ft/(1-f)$. If we bound the out-degree and in-degree of every node to the expected size of the neighbor set (i.e., $t = 1$), thus forcing every node to have the same degree, then the expected number of malicious entries in the neighbor sets of correct nodes is bounded by $f/(1-f)$.

B. Mechanisms to Enforce Degree Bound

The next important question is how to enforce the degree bound. The obvious solution of a centralized service that keeps track of each overlay member's degree suffers from the same problem as the centralized membership service discussed in Section II. A practical alternative is a distributed mechanism, where participating nodes are responsible for monitoring each other's degree.

We enforce the degree bound through distributed auditing. Each node in the system periodically audits neighboring nodes to ensure compliance with the degree bound. For this purpose, each node x in the overlay is required to maintain a set of all the nodes that have x in their neighbor set. We refer to this list as the *backpointer set* of x.

Periodically, x anonymously challenges each member of its neighbor set by asking it for its backpointer set. If the number of entries in that backpointer set is greater than the in-degree bound, or x does not appear in the backpointer list, then the auditee has failed the test, and x removes that member from its neighbor set. To prevent an attacker from consuming the in-degree of correct nodes, a similar auditing procedure is performed to ensure that the members of a node's backpointer set maintain a neighbor set of the appropriate size. When an auditing node finds one of its neighbors not in compliance with the degree limit, it immediately drops the connection to that neighbor. Therefore, the degree of such nodes naturally tends towards the allowed bound.

To ensure that replies to an audit challenge are fresh and authentic, x includes a random nonce in the challenge. The auditee includes the nonce in its reply, and digitally signs the response. When x receives the reply, it checks the signature and the nonce before accepting the reply. Asserting freshness and authenticity ensures that a correct node cannot be framed by a malicious node that fabricates or replays audit responses.

Moreover, it is essential that the identity of the auditor remains hidden from the auditee. Otherwise, a malicious auditee could easily produce a fake response of the allowed size that includes the auditor. Ensuring auditor anonymity is the subject of the next section.

IV. Anonymous Auditing

In this section, we describe a mechanism to preserve the anonymity of the auditor — a necessary building block for our distributed auditing mechanism to enforce degree bounds. We also analyze the effectiveness of our anonymous auditing technique.

A. Design

To enable anonymous auditing, we need a communication channel that hides the auditor's identity. Although there are many general-purpose anonymous channel mechanisms for overlay networks [13], [17], [27], [32], [45], the anonymity requirements of our auditing mechanism are weaker than those

provided by the above mentioned mechanisms. A node is audited regularly *only* by each member of its neighbor and backpointer set. Therefore, it is sufficient to ensure that any challenge is equally likely to come from any member of these sets. Furthermore, it is sufficient to ensure that the identity of the challenger is hidden only most of the time. An occasional failure of sender anonymity can only moderately increase the time to detect a node with excessive degree.

We designed an anonymous channel that exploits these weaker requirements to reduce overhead. To obscure the sender's identity, each challenge is relayed through an intermediate node. The mechanism used to choose such an *anonymizer node* is described in Section IV-C.

In the following, we consider a worst-case adversarial model, where all the malicious nodes are colluding to defeat auditing. We describe the design using an example. Suppose a correct node x wants to audit a node z in its routing table. Node x picks a random node y, called an *anonymizer*, to relay a challenge to node z. There are four cases to consider:

- **Case 1: z is malicious, y is correct:** z does not know the auditor's identity, so z can either guess or not respond at all.
- **Case 2: z is malicious, y is malicious:** In this case, y colludes with z, allowing z to craft a custom response for x. x will not detect that z is not in compliance.
- **Case 3: z is correct, y is correct:** z is in compliance and the audit always succeeds. From x's perspective, this case is indistinguishable from case 2.
- **Case 4: z is correct, y is malicious:** y can drop the challenge or response, causing x to falsely suspect z.

Consider a simple test where x marks node z suspicious if either (a) it did not receive any response after a sufficient timeout or (b) the returned set does not contain x or, finally, (c) the returned set size is greater than the degree bound. Intuitively, cases 1 and 2 suggest that auditing z only once is not sufficient since y could be malicious and colluding with z, or z guesses correctly, or the challenge or response was dropped by the network. Additionally, case 4 suggests that if z is correct and y is malicious, then x would falsely suspect z. Therefore, x needs to perform several audits through different anonymizers at random times before z can be considered to be in compliance or violation of the degree bound.

To prevent a malicious node from correlating different challenges from the same auditor, auditors randomize the interval between subsequent challenges. Likewise, an auditor waits for a random delay between discovering the anonymizer node and using the discovered anonymizer in relaying the challenges. This is to prevent a malicious node from inferring the source of an audit challenge by correlating the discovery traffic with the challenge traffic.

B. Analysis

Next, we determine the probability that a malicious node is detected and that a correct node is falsely blamed, respectively. This guides the choice of the various parameters. We assume that the adversary seeks to maximize the in-degree of its nodes. We further assume that the intermediate nodes used by an auditor to relay challenges are malicious with probability f.

Consider a strategy where a node is considered malicious if it answers less than k out of n challenges correctly. Since audits are independent of each other, the audits can be viewed as Bernoulli trials [12, Appendix C.4]. A correct node can fail an audit only if the intermediate node is malicious. Thus, the probability that a correct node is considered malicious in an individual audit, consisting of n challenges, is

$$\sum_{i=0}^{k-1} \binom{n}{i} f^{n-i}(1-f)^i \ . \tag{1}$$

Given an upper bound on f, we can fix n and pick k such that this probability is very small. For instance, by setting $n = 24$ and $k = 12$, and assuming $f \leq 0.2$, this probability is less than 0.02%.

Consider the possible adversarial strategies when a malicious node is audited via a correct anonymizer node. The malicious node can respond with a random subset of the maximal allowed size from its true set. Let r be the *overload ratio*, the ratio of the size of its true set versus the maximal allowed size. Then the probability that the randomly selected subset passes the audit is $1/r$, assuming that auditing is done completely anonymously. We further assume that when the malicious node is audited through a correct anonymizer node, the malicious node answers the challenge with a probability of c, and does not respond with a probability of $1-c$.

For each challenge, there are four possible cases:

1) With probability f, the anonymizer is colluding and the malicious node can pass the challenge.
2) With probability $(1-f)c/r$, the malicious node's random response includes the auditor and passes the challenge.
3) With probability $(1-f)c(1-1/r)$, the malicious node's random response does not include the auditor and fails the challenge.
4) With probability $(1-f)(1-c)$, the malicious node does not respond.

For a malicious node to pass an audit consisting of n challenges, there must be at least k instances of case 1 or 2 and not a single instance of case 3 (i.e., all remaining cases are of type 4). Thus, the probability that a malicious node passes the audit undetected is

$$\sum_{i=k}^{n} \binom{n}{i} [f + (1-f)c/r]^i [(1-f)(1-c)]^{n-i} \ . \tag{2}$$

Fig. 1 and Fig. 2 show the probability that a malicious node is detected for different settings of n, k, and f, assuming that malicious nodes have an overload ratio $r = 1.2$. The overload ratio r is the ratio of the true size of the target's set to the maximum allowed size. Note that the larger the overload ratio, the less likely a randomly selected subset would contain the auditor, thus the easier a malicious node would get detected. All results reflect a setting that is, from the attacker's perspective, optimal, i.e., with the least probability that the

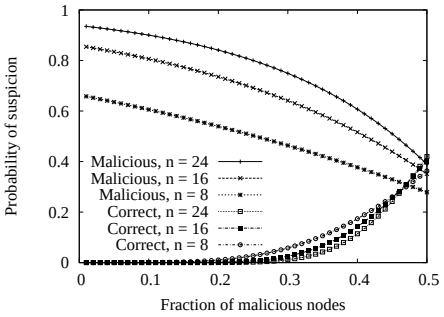

Fig. 1. Probability of marking a malicious/correct node suspicious vs. fraction of malicious nodes, $k = n/2$.

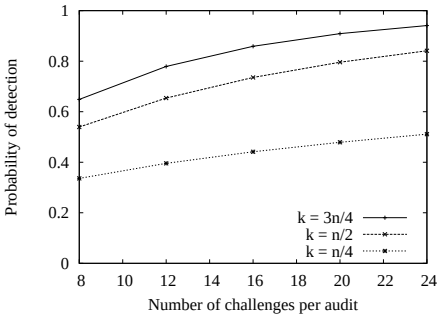

Fig. 2. Probability of detecting a malicious node with different numbers of challenges per audit, $f = 0.2$.

malicious node would be detected. This optimal setting was determined empirically for each data point, by finding the k and c that minimized the probability of detection.

Thus, given an upper bound for f, one can pick the values for n and k to minimize the probability of falsely blaming correct nodes, while effectively detecting malicious nodes. For example, assume $f \leq 0.25$. By setting $n = 24$ and $k = 12$, the probability of falsely blaming a correct node is around 0.2%, but malicious nodes with overload ratio $r \geq 1.2$ are detected with probability at least 95.9%. Therefore, we pick $k = n/2$, requiring a node to answer at least half of the audits correctly.

C. Discovery of Anonymizer Nodes

A critical element of our distributed anonymous auditing mechanism is the selection of anonymizer nodes. The selection mechanism must ensure that the anonymizer node is malicious with probability no more than f and that the identity of the anonymizer node used by an auditor reveals nothing about the identity of the auditor.

We consider three techniques to select the anonymizer nodes:

Random node: A node is chosen randomly from the entire population of the overlay before each audit, shown by Fig. 3(a). This can be done by drawing a random number in the id space and using secure routing to select the live overlay node with id closest to that chosen number. The problem with this approach is that a malicious node could potentially overhear the routing request from the auditor. If the chosen node is subsequently used in an audit challenge, the attacker can infer the source with high probability.

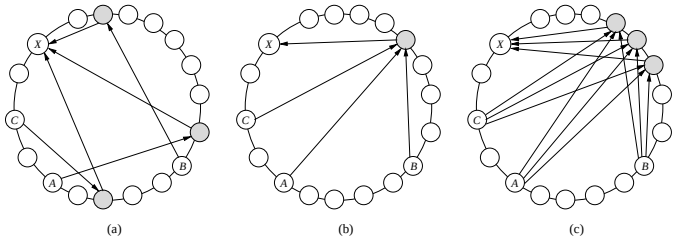

Fig. 3. Different ways to select anonymizer nodes. Nodes A, B, and C audit node X. Dark nodes represent the anonymizer nodes.

Node closest to $H(x)$: When a node wishes to audit x, it selects the node with id numerically closest to the hash $H(x)$ (e.g. SHA-1) as the anonymizer node, shown in Fig. 3(b). Once the anonymizer node is found, it can be used continuously, as long as it remains the node with id closest to $H(x)$. Since all auditors of a given node use the same anonymizer node and have random challenge times, the challenges will be mixed together. Malicious nodes that observe discovery routing traffic will learn nothing about the source of a particular challenge. However, if the chosen node happens to be malicious, then every audit of node x is ineffective.

Random node among the ℓ closest to $H(x)$: Here, auditors use a random node among the ℓ nodes with ids closest to $H(x)$ (see Section II-C) as the anonymizer for a given challenge, shown by Fig. 3(c). Per our assumption that nodeIds are assigned randomly and cannot be forged, this anonymizer set represents a random set of nodes in the overlay with only a fraction f being malicious, exactly as if anonymizers were chosen at random from the full overlay. However, because all auditors use the same set of nodes to select anonymizers and their challenges are interleaved, malicious nodes that observe traffic will learn nothing useful about who is performing any given audit.

Since the last technique is the most robust, we used it in our auditing implementation. In order to maintain a sufficient size of the anonymizer set under churn (i.e., the arrival and departure of overlay nodes), all auditors periodically refresh their list by determining the latest set of ℓ closest nodes to $H(x)$. Moreover, we set ℓ to be equal to the number of challenges per audit, n.

How big must an anonymizer set be? For overlays with a sufficiently large number of nodes, the distribution of malicious nodes in a vicinity of a point in the identifier space set can be approximated by a binomial distribution. The probability that at least half of the nodes in an anonymizer set of size n is malicious can be bounded above by

$$\sum_{i=\lceil n/2 \rceil}^{n} \binom{n}{i} f^i (1-f)^{n-i} \ .$$

For example, only 1% of all anonymizer sets of size 16 contain at least one-half malicious nodes, while only 0.1% of size 24

contain at least one-half malicious nodes in the anonymizer set. Also, the probability drops exponentially with increasing anonymizer set size.

Therefore, we can dynamically set the size of anonymizer sets to balance the probability of having an anonymizer set with at least one-half malicious nodes and the overhead of maintaining such an anonymizer set. To ensure that the expected number of anonymizer sets with at least one-half malicious nodes is less than one, in an overlay of size N, we need to choose the size of the anonymizer set n to satisfy

$$N \sum_{i=\lceil n/2 \rceil}^{n} \binom{n}{i} f^i (1-f)^{n-i} < 1 \quad .$$

For instance, assume $f = 0.2$, for $N = 100$, 1000, and 10000, we need to choose n to be 13, 21, and 29, respectively.

V. Evaluation

In this section, we set out to answer the following questions empirically:

- How serious are Eclipse attacks on structured overlays?
- How effective is the existing defense based on proximity neighbor selection (PNS) against Eclipse attacks?
- Is degree bounding a more effective defense? What is the impact of degree bounding on the performance of PNS?
- Is distributed auditing effective and efficient at bounding node degrees?

A. Experimental Setup

We use MSPastry [28], which comes with a packet-level discrete-event simulator. Two different physical network topology models were used in the simulations: The GT-ITM transit-stub network topology model, consisting of 5050 routers [41], and a set of measured pair-wise latency values for up to 10,000 real Internet host, obtained with the King tool [19]. Unless stated differently, the GT-ITM model was used in the simulations.

Pastry was configured with routing base $b = 4$ and leaf set size $\ell = 16$. Pastry implements proximity neighbor selection (PNS) [6] and an optimized version of Castro et al.'s secure routing primitive, described in [36]. The overlay membership is static unless otherwise stated.

The fraction of malicious nodes is set to $f = 0.2$ unless otherwise specified. Malicious nodes collude to maximize the number of entries referring to malicious nodes in the neighbor sets of correct nodes. In particular, malicious nodes misroute join messages of correct nodes to each other and supply references exclusively to other malicious nodes as part of the overlay maintenance protocol. Malicious nodes initialize their routing tables to refer to good nodes whenever possible, in order to consume as much of the in-degree of good nodes as possible. Malicious nodes maintain an out-degree of 16 per routing table row, which is the optimal strategy from the perspective of the attacker. Results of simulations we performed show that any attempt of using a larger out-degree leads to a quicker detection of the malicious nodes, and diminishes the overall impact of the attack.

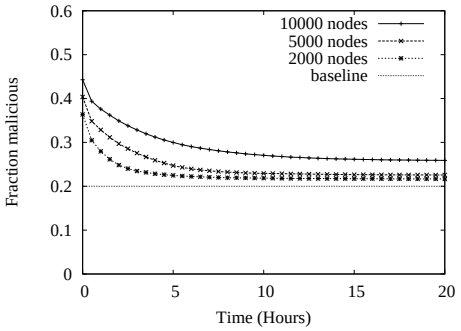

Fig. 4. Fraction of malicious nodes in routing tables of correct nodes, at different network sizes for GT-ITM topology.

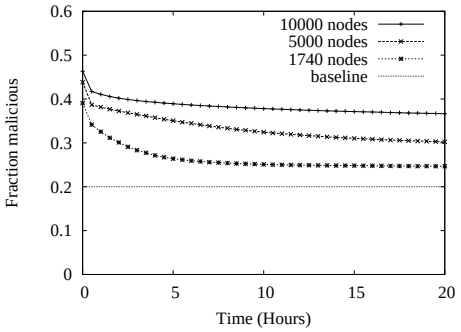

Fig. 5. Fraction of malicious nodes in routing tables of correct nodes, at different network sizes with King latencies.

B. Eclipse Attacks

First, we evaluate the impact of an Eclipse attack on a Pastry overlay when PNS is turned off, i.e., when nodes pick neighbors regardless of the delay in the physical network. In this case, an Eclipse attack is extremely effective, as expected. When the overlay network stabilizes, the fraction of malicious neighbor set entries is over 70% for a 1000-node overlay and more than 80% for overlays with 5000 nodes. In the top row of the routing table, where the constraints on a neighbor's identifier are weakest, the fraction is over 90% for a 1000-node overlay and approaches 100% for overlays of 10,000 nodes or more.

Next, we evaluate the effectiveness of Eclipse attacks on a Pastry overlay with PNS. Fig. 4 shows the fraction of malicious nodes at different overlay sizes vs. the simulation time. Observe that with time, the average fraction of malicious nodes over the full routing table drops to less than 30% within 10 hours of simulation for all system sizes. The fraction for the top row drops from 78% to 41% for a 10,000 node network within 10 hours. These results suggest that PNS, by itself, reduces the impact of an Eclipse attack, particularly in small overlays.

This experimental setup, however, benefits from a good separation of nodes in the delay space of the GT-ITM topology model. To see if the result holds up when real delays measured in the Internet are used, we repeated the experiment with the King delay sets. Fig. 5 shows that the effectiveness of PNS in defending against Eclipse attacks is significantly reduced.

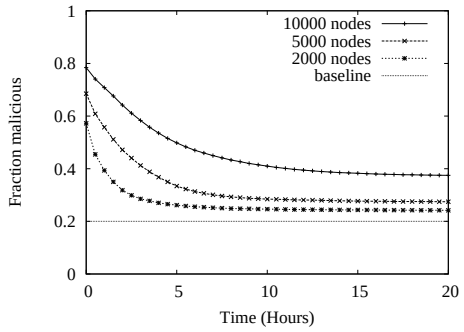

Fig. 6. Fraction of malicious nodes in top row of routing table of correct nodes, at different network sizes for GT-ITM topology.

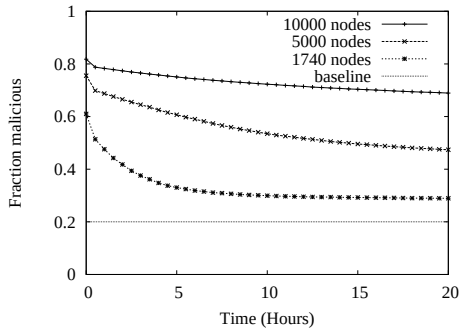

Fig. 7. Fraction of malicious nodes in top row of routing table of correct nodes, at different network sizes with King latencies.

As the overlay size increases, PNS becomes less effective. In the actual Internet, a large fraction of nodes lie within a small delay band, making PNS less effective as a defense mechanism.

It is easiest for the malicious nodes to supply suitable entries for routing table rows with weak constraints on the identifier, which explains the high fraction of entries to malicious nodes in the results for row zero, as shown in Fig. 6 and 7. Unfortunately, the entries in row zero are used most often in routing a node's own messages, because they are typically used for the first hop. Thus, an Eclipse attack can be highly effective at intercepting overlay traffic.

We conclude that a PNS-based defense against Eclipse attacks alone will not be effective in the real Internet, requiring better mechanisms to defeat such attacks.

C. Effectiveness of Degree Bounding

To measure the effectiveness of degree bounding, we first perform an idealized experiment where the degree bounds are perfectly maintained. We use an oracle to determine if a node has exceeded its degree bounds; no anonymous auditing is performed. This allows an evaluation of an "ideal" degree bounding defense, independent of any particular implementation of bounds enforcement.

Malicious nodes attempt to maximize their in-degree, but correct nodes will check a node's in-degree before adding it to the routing table during a routing update. Correct nodes, then, will refuse to increase the in-degree of a node that is already over the limit. Experimentally, we varied the in-degree limit from 16 through 64 entries per routing table row. Table I presents the average fraction of malicious routing table entries in the entire routing table. The fraction of malicious routing table entries for each row was very similar.

TABLE I

FRACTION OF MALICIOUS NODES IN CORRECT NODES' ROUTING TABLES WITH DIFFERENT DEGREE BOUNDS PER ROW.

	Number of nodes			
Bound	1,000	5,000	10,000	20,000
16	0.24	0.24	0.24	0.24
32	0.24	0.29	0.31	0.37
48	0.24	0.31	0.35	0.45
64	0.24	0.33	0.38	0.48
unlimited	0.24	0.35	0.42	0.50

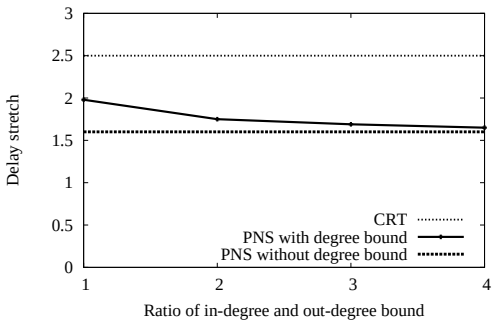

Fig. 8. Delay stretch versus the ratio of in-degree and out-degree bound for a network with 20,000 nodes.

The results show that in-degree bounding is effective at maintaining a low fraction of malicious routing table entries. As expected, this fraction is approximately $ft/(1-f) = 0.25$ for $t = 1$, i.e., when the bound is set to the expected average degree. The effectiveness of our defense decreases when the overlay is large and the in-degree bound t is loose, since malicious nodes are able to exploit the loose in-degree bounds to consume a higher fraction of the out-degree of correct nodes.

We also evaluated the impact of in-degree bounding on routing delays under PNS. We measure the delay stretch, which is the ratio of overlay routing delay versus the direct IP delay. Since degree bounding puts extra constraints on the choice of neighbors, it can increase routing delays even in the absence of attacks. We observed a delay stretch increase of approximately 25% in an overlay with 20,000 nodes with no malicious nodes and an in-degree bound of 16 per row. The penalty decreases to about 8% for a bound of 32, as shown in Fig. 8. We conclude that in-degree bounding is highly effective against Eclipse attacks so long as a tight in-degree bound is enforced, and that the resulting increase in delay stretch is tolerable.

D. Effectiveness of Auditing

Next we evaluate the effectiveness of auditing in defending against the Eclipse attacks. In this experiment, we simulate a

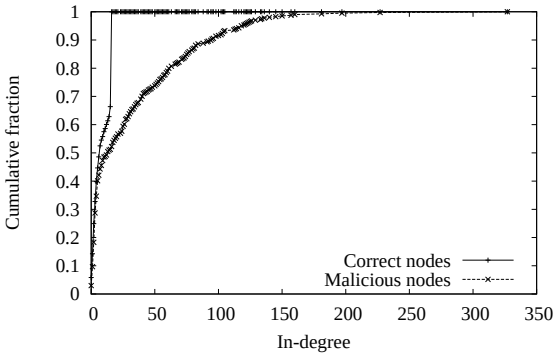

Fig. 9. In-degree distribution before auditing starts.

network with 2000 nodes. We set $n = 24$ to ensure that the expected number of anonymizer sets with at least one-half malicious nodes is less than 1, as described in Section IV-C. Therefore, out of the 24 challenges, the number of correct responses has to be at least 12 to pass an audit iteration. A new iteration is then started immediately. Correct nodes require an in-degree bound of 16 per routing table row and backpointer set row, respectively, while malicious nodes do not impose any limit.

We assume malicious nodes employ the optimal strategy to maintain high in-degree without getting caught, i.e., they respond to audits coming through correct anonymizer node with the optimal probability, as described in Section IV-B. About once every two minutes, a node audits each of its overlay neighbors, with each being audited at a random instant in the 2 minute interval.

To evaluate the effectiveness of auditing, we simulated both static membership and churn scenarios. We simulated 0%, 5%, 10%, and 15% churn every hour. The average lifetime of a node in our simulation is indefinite, 20, 10, and 7 hours, respectively, for the above churn rates. These churn rates are higher than those reported by Bolosky et al. [3] for a corporate environment, where the average session time was reported as 37.7 hours, but much lower than the session times reported for some file-sharing networks [2]. Since the auditing rate is once every 2 minutes in our simulations and it takes 24 challenges to decide whether a node is suspicious, a node needs to be alive for an hour for meaningful audit data to be collected. As we will show, higher churn would require a higher auditing rate and proportionally higher overhead. Therefore, the target environment for our defense is an overlay with low to moderately high churn.

One concern is that a malicious node could leave the overlay just before auditing finishes, to avoid being detected once an audit completes. However, such a node would loose all its overlay connections, which would defeat the goal of an Eclipse attack. Alternatively, the node could try to reestablish the connections to its previous neighbors; however, in this case, the neighbors would continue their audits where they left off. Caching audit state for a few days is sufficient for this purpose.

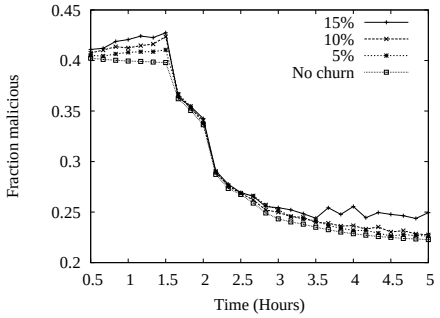

Fig. 10. Fraction of malicious nodes over the full routing table of correct nodes.

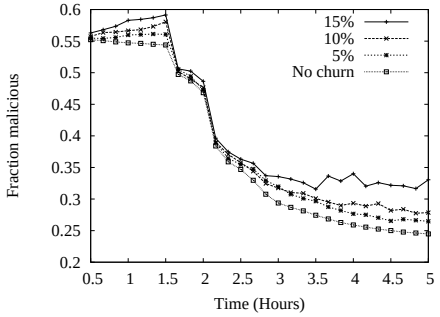

Fig. 11. Fraction of malicious nodes in the top row of the routing table of correct nodes.

1) In-degree distribution: Fig. 9 shows a cumulative distribution of in-degree for both correct and malicious nodes during an Eclipse attack, but before auditing has started. Clearly, malicious nodes have been able to get in-degrees far larger than 16. Assuming static membership (i.e., no churn), we simulated approximately 10 hours of operation with auditing enabled. After this, every node in the system, correct or malicious, has in-degree equal or below the allowed bound of 16. This shows that our auditing scheme has successfully caught each and every case of a malicious node with excessive in-degree.

2) Detecting malicious nodes: Fig. 10 and Fig. 11 show the fraction of malicious neighbors over time in the entire routing table and in just the top row, respectively. To show the impact of auditing more clearly, auditing starts after 1.5 hours of simulated time, whereas the malicious nodes begin an Eclipse attack immediately. Before auditing has begun, the routing tables start with around 40% of the entries of correct nodes referring to malicious nodes (versus 20% of the nodes being malicious). Note that correct nodes enforce an in-degree bound of 16 per row from the start of simulation while malicious nodes do not. Under churn, the newer correct nodes point to malicious nodes rather than correct nodes, causing the fraction to increase with time for different churn settings until auditing is started.

Within 2 simulated hours after auditing has begun, the fraction of malicious nodes drops to below 30% and 25% for the top row and overall, respectively, with static membership as well as with a churn of 5% and 10% per hour. However, the

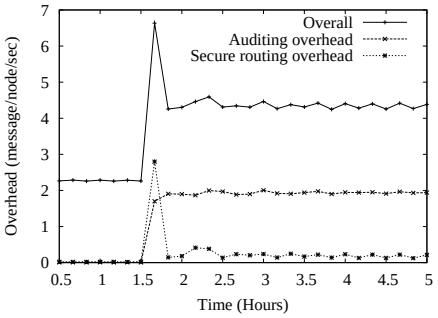

Fig. 12. Total cost of maintaining the overlay.

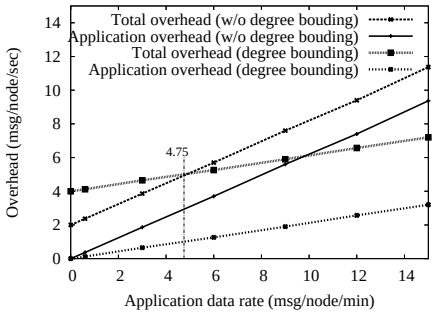

Fig. 13. The cost of delivering application messages at different rates with and without degree bounding.

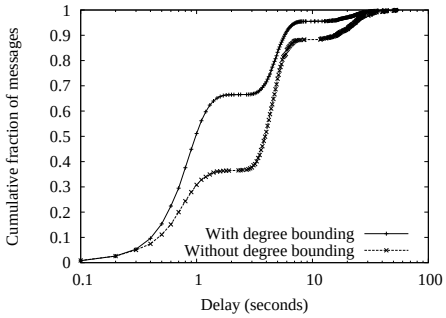

Fig. 14. Cumulative distribution of delays observed with and without degree bounding.

fraction of malicious nodes in the top row for 15% churn is above 30%. To explain why the fraction does not drop further, we considered the in-degree distribution of malicious nodes relative to the lifetime of these nodes. As expected, nodes with higher in-degree are nodes that are relatively new and thus have not yet been sufficiently audited.

We then doubled the auditing rate to once every minute. We observe that the fraction of malicious nodes in the top row stabilized around 27%, an improvement of 3%. Auditing at a rate of once every minute under a 15% churn rate has the same effect as auditing once every 2 minutes at a churn of 10% per hour. This shows the fundamental tradeoff between churn rate and auditing rate. Higher churn requires more auditing.

3) Communication overhead: Fig. 12 shows the total overhead for maintaining the overlay. This includes the basic cost for maintaining the Pastry overlay with PNS, the additional cost of maintaining the constrained routing table (CRT), and the cost of auditing. Recall that the CRT is necessary to implement secure routing, which is in turn needed to securely find anonymizer sets.

The auditing cost, shown separately, is proportional to the auditing rate, which is set to one audit every 2 minutes in our simulation. A lower auditing rate would reduce the cost, but would also require a longer time to detect malicious nodes and would likewise be less effective under churn. As before, we first allow the system to run for 1.5 hours and then enable auditing. The spike in secure routing overhead at this time is the result of every node searching for the anonymizer nodes it will subsequently use. The figure shows that the absolute auditing cost (2 msg/node/sec), the cost of secure routing (0.2 msg/node/sec), and the total maintenance cost (4.2 msg/node/sec) are very low.

We therefore conclude that auditing is effective, with a small increase in overhead.

4) False positives: One concern for our auditing scheme is false positives. After 10 hours of simulation, we observed that approximately 100 connections between correct nodes were incorrectly marked as suspicious and removed from neighbor sets. This is out of the roughly 96,000 total connections. Thus, our auditing scheme has only a 10^{-3} false positive rate after 10 hours of auditing. To ensure that the false positives do not become a concern in a long running system, suspicious marks are discarded at a rate similar to the false positive rate, i.e. we unmark 1 suspicious node every week in a 2000 node network. This introduces a small background churn in the system since these unmarked nodes are treated as new nodes and are freshly re-audited. The false positive rate could be reduced further by increasing the value of n (i.e., performing more audits before reaching a conclusion).

E. Comparison with Previous Technique

The rest of this section compares degree bounding with the previously proposed defense by Castro et al. [5] based on a constrained routing table. We compare both network overhead and communication delay. In this experiment, the auditing rate is once every 2 minutes and the churn rate is set to 5% per hour. In both techniques, an application message is first routed through the normal PNS-based routing table, which is either degree-bounded or not. If it fails, the message is then re-sent using secure routing using the CRT.

1) Network overhead: The goal of this experiment is to compare the total cost of delivering a given application message to the correct destination, including the maintenance overhead of the overlay. We simulate an application that sends traffic at a constant rate. We then measure the total number of message exchanged, which includes the maintenance cost and the secure routing cost for the two techniques. Without degree bounding, the maintenance cost of the overlay is the cost of maintaining two routing tables: one PNS and one CRT. With degree bounding, there is an additional cost of anonymous auditing.

Fig. 13 compares the total cost in messages per node per second of routing the application messages. The figure shows that the total overhead with auditing is lower, unless an application rarely sends messages, with an average of less than 4.75 msg/node/min. Also observe that the auditing technique reduces the reliance on (and thus the overhead of) the secure routing mechanism by more than half in most cases. In other words, the cost of auditing pays off, unless the application message rate is very low.

2) Communication delay: Another metric we used to compare the two schemes is the delay observed by the application to obtain the correct destination for a given key. Fig. 14 presents the cumulative distribution of delays observed for the two techniques. The knees in the curves are due to the fact that in our implementation, secure routing is done in discrete iterations [36]. The average delay with degree bounding is 3.05 seconds, while it is 5.4 seconds without degree bounding. Moreover, more than 90% of messages were correctly delivered within 5.8 seconds with degree bounding, while delivering the same fraction without degree bounding took 16 seconds.

VI. Discussion

A. Limitations of auditing

With the proposed defense, each node independently discovers malicious nodes. An attacker node can remain in the system after attacking a subset of nodes while appearing to behave correctly to others. If a correct node, after detecting a malicious node, could present a verifiable proof of misbehavior to other correct nodes, then it would be possible to remove the malicious node from the system as soon as it is detected. However, generating such proofs could require complex cryptographic operations or Byzantine agreement between correct nodes, and could also cause problems when the auditing system has false positives. Our technique, on the other hand, does not rely on any cryptographic functionality other than authenticated audit messages and certified node identifiers, and is robust against auditing errors.

B. Adversary response strategy

In our experiments, a malicious node responds to an anonymous challenge with a probability of less than one; if it responds, it presents a subset of its true neighbor / backpointer set. An alternative strategy would be to always return a fixed subset of the maximal allowed size. Unlike the random subset strategy, such a malicious node's degree never drops *below* the bound, since the nodes included in the fixed subset will never suspect the malicious node. As a result, malicious nodes can in some cases achieve a total degree that is 1–2% higher than with the random strategy. Neither strategy, however, allows malicious nodes to maintain a degree that is significantly above the average degree in the overlay.

C. Eclipse attacks on hierarchical overlay systems

Our technique uses deviations in the degree of a given node as an indication of a possible Eclipse attack. Therefore, the defense is not directly applicable to systems that use asymmetry deliberately for performance reasons. For example, superpeers in KaZaA aggregate index information and maintain connections to a large number of ordinary nodes. Unless such superpeers can be authenticated and implicitly trusted, they pose a security threat since they are in a position to eclipse the entire overlay. Securing such heterogeneous p2p overlays is an interesting research problem and may require a different set of solutions.

D. Auditing in unstructured overlays

In this paper, we have limited ourselves to defending against Eclipse attacks in structured overlays. For auditing to work in unstructured overlays, there needs to be a mechanism to securely select auditor sets. A straightforward solution is to maintain an additional structured overlay alongside the unstructured overlay, solely for the discovery of anonymizer nodes. The cost of maintaining such a structured overlay can be made very low, as shown by Castro et al. [4]. As a related example, modern BitTorrent clients optionally use a Kademlia [25] structured overlay to maintain a "distributed tracker", which locates peers for the otherwise unstructured exchange of data [39].

E. Localized attacks

Instead of trying to attack all correct nodes in the overlay simultaneously, an adversary could attempt to mount a *localized* Eclipse attack. In this attack, malicious nodes attempt to occupy row zero of the routing tables of only a small set of victim nodes. Such an attack may not be detected by degree bounding, because it does not require malicious node to have a significantly higher-than-average node degree.

However, the colluding nodes have to have low network latency to the victim nodes. (Recall that row zero is the most flexible row, and to be included in that row the malicious nodes have to have sufficiently low delay to the target nodes under PNS). So, this type of attack is possible only if the victim nodes are locally surrounded by malicious nodes in the physical network. Effective defenses against such localized attacks in an overlay are an open research problem.

VII. Related work

Sit and Morris [37] consider routing under attack in peer-to-peer systems. They propose *iterative routing*, where the source node repeatedly ask for the next hop of a message, and contacts the nodes in the path successively. Castro et al. [5] propose the use of a routing failure test and redundant routing to improve the chance of successful routing. They also put strong structural constraints on the neighbor set to thwart Eclipse attacks.

Despite the structural constraints enforced by the overlay membership protocol (e.g. prefix match), Castro et al. [6] and Gummadi et al. [20] show that proximity neighbor selection can provide good network locality properties in structured peer-to-peer overlay. However, the strong constraints on the neighbor sets required to defend against Eclipse attacks leave

no flexibility in neighbor selection and therefore prevent such optimizations.

Hildrum and Kubiatowicz [21] propose the use of *wide paths*, where they add redundancy to the routing tables and use two nodes for each hop. They show that this provides better fault-tolerance per redundant overlay node than multiple paths, while still allowing flexibility in neighbor selection. However, as noted by Chun et al. [9], the performance improvement from exploiting network proximity or node capacity comes at the price of increased vulnerability against targeted attacks.

Recently, Condie et al. [11] proposed a novel defense against Eclipse attacks based on induced churn. The idea is to periodically reset the PNS routing table to a constrained routing table (CRT), rate limit the updates of routing tables, and periodically change node identifiers to mitigate the effect of malicious nodes infiltrating the routing tables of correct nodes. Unlike our design, this approach requires that node identifiers be changed periodically, which limits its applicability to systems that can deal with the resulting churn.

Other works achieve fault-tolerance through specially designed overlay structures. Saia et al. [34] and Naor and Wieder [29] also use ideas related to wide paths and recursive routing. Fiat and Saia [15] consider a butterfly network of virtual nodes, where fault-tolerance is achieved by having more than one starting point for each message.

VIII. Conclusions

This paper has shown that Eclipse attacks on overlays are a real threat: attackers can disrupt overlay communication by controlling a large fraction of the neighbors of correct nodes even when they control only a small fraction of overlay nodes. Therefore, it is important to defend against Eclipse attacks. We have proposed a novel defense that prevents Eclipse attacks using anonymous auditing to bound the degree of overlay nodes. This defense can be used in homogeneous structured overlays with moderate churn and, unlike previous defenses based on a constrained routing table, it permits important optimizations like proximity neighbor selection. Experimental results show that the defense can prevent attacks effectively in a structured overlay. Moreover, for typical systems and for all but very low application traffic, our defense is more efficient than previously proposed techniques.

IX. Acknowledgements

This work originated during an internship of the first author at Microsoft Research, Cambridge. We wish to thank Miguel Castro and Antony Rowstron for their ideas, advice, and support. This research was supported by Texas ATP (003604-0079-2001), by NSF (CNS-0509297 and ANI-0225660), and by Microsoft Research. We thank the anonymous reviewers for their helpful comments.

References

[1] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, June 2003.

[2] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *Proceedings of 2th International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003.

[3] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of ACM SIGMETRICS*, June 2000.

[4] M. Castro, M. Costa, and A. Rowstron. Performance and dependability of structured peer-to-peer overlays. In *Proceedings of International Conference on Dependable Systems and Networks (DSN 2004)*, Florence, Italy, June 2004.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of USENIX Operating System Design and Implementation(OSDI)*, Boston, MA, Dec. 2002.

[6] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, June 2003.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of USENIX Operating System Design and Implementation(OSDI)*, New Orleans, Louisiana, Feb. 1999.

[8] Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an Internet broadcast system based on overlay multicast. In *Proceedings of USENIX Annual Technical Conference*, Boston, MA, June 2004.

[9] B.-G. Chun, B. Y. Zhao, and J. D. Kubiatowicz. Impact of neighbor selection on performance and resilience of structured p2p networks. In *Proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, Ithaca, NY, Feb. 2005.

[10] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[11] T. Condie, V. Kacholia, S. Sankararaman, J. Hellerstein, and P. Maniatis. Induced Churn as Shelter from Routing-Table Poisoning. In *Proceedings of Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2006.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, 2nd edition edition, 2001.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of 13th USENIX Security Symposium*, San Diego, CA, Aug. 2004.

[14] J. R. Douceur. The Sybil Attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar. 2002.

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

[16] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *Proceedings of Networked System Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004.

[17] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar. 2002.

[18] The Gnutella protocol specification. `http://dss.clip2.com/GnutellaProtocol04.pdf`.

[19] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of ACM Internet Measurement Workshop*, Marseille, France, Nov. 2002.

[20] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[21] K. Hildrum and J. Kubiatowicz. Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks. In *Proceedings of 17th International Symposium on Distributed Computing*, Sorrento, Italy, Oct. 2003.

[22] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of USENIX Operating System Design and Implementation (OSDI)*, San Diego, CA, 2000.

[23] KaZaA. `http://www.kazaa.com/`.

[24] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Proceedings of Annual ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, May 1997.

[25] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar. 2002.

[26] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of Symposium on Operating System Principles (SOSP)*, Charleston, SC, Dec. 1999.

[27] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. AP3: Anonymization of group communication. In *Proceedings of ACM SIGOPS European Workshop*, Leuven, Belgium, Sept. 2004.

[28] MSPastry. `http://research.microsoft.com/~antr/Pastry/`.

[29] M. Naor and U. Wieder. A simple fault tolerant distributed hash table. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb. 2003.

[30] OverNet. `http://www.overnet.com/`.

[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[32] M. K. Reiter and A. D. Rubin. Anonymous Web transactions with Crowds. *Communications of the ACM*, 42(2):32–48, Feb. 1999.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware*, Heidelberg, Germany, Nov. 2001.

[34] J. Saia, A. Fiat, S. Gribble, A. Karlin, and S. Saroiu. Dynamically fault-tolerant content addressable networks. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar. 2002.

[35] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against Eclipse attacks in overlay networks. In *Proceedings of SIGOPS European Workshop*, Leuven, Belgium, Sept. 2004.

[36] A. Singh, T.-W. J. Ngan, P. Druschel, and D. S. Wallach. Implementation and evaluation of secure routing primitives. Technical Report TR05-459, Rice University, Jan. 2006.

[37] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, Massachusetts, Mar. 2002.

[38] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[39] Trackerless in BitTorrent. `http://www.bittorrent.com/trackerless.html`.

[40] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proceedings of USENIX Annual Technical Conference*, Boston, MA, June 2004.

[41] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, Mar. 1996.

[42] X. Zhang, J. Liu, B. Li, and P. Yum. DONet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE INFOCOM*, Miami, FL, Mar. 2005.

[43] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB-CSD-01-1141, U. C. Berkeley, Apr. 2001.

[44] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien. A first look at peer-to-peer worms: Threats and defenses. In *Proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS)*, Cornell, NY, Feb. 2005.

[45] L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron. Cashmere: Resilient anonymous routing. In *Proceedings of Networked System Design and Implementation (NSDI)*, Boston, MA, May 2005.