Limiting Sybil Attacks in Structured P2P Networks

Hosam Rowaihy, William Enck, Patrick McDaniel, and Thomas La Porta
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
Email: {rowaihy, enck, mcdaniel, tlp}@cse.psu.edu

Abstract—**One practical limitation of structured peer-to-peer (P2P) networks is that they are frequently subject to Sybil attacks: malicious parties can compromise the network by generating and controlling large numbers of shadow identities. In this paper, we propose an admission control system that mitigates Sybil attacks by adaptively constructing a hierarchy of cooperative peers. The admission control system vets joining nodes via client puzzles. A node wishing to join the network is serially challenged by the nodes from a leaf to the root of the hierarchy. Nodes completing the puzzles of all nodes in the chain are provided a cryptographic proof of the vetted identity. We evaluate our solution and show that an adversary must perform days or weeks of effort to obtain even a small percentage of nodes in small P2P networks, and that this effort increases linearly with the size of the network. We further show that we can place a ceiling on the number of IDs any adversary may obtain by requiring periodic reassertion of the IDs continued validity.**

I. Introduction

Structured peer-to-peer (P2P) networks provide a cooperative, stable, and robust mechanism for storing and retrieving arbitrary content. Deployments of networks such as Chord [9], CAN [4] and Pastry [5] can reach a massive scale, where millions of users share content over global networks. These networks can be used to successfully construct large-scale applications such as file-sharing and distributed filesystems.

User identifiers (IDs) uniquely identify participant endpoints ($nodes$) in P2P networks. Structured networks reduce search times by mapping content directly onto nodes based on IDs. For this reason, the assignment and use of IDs is essential to correct operation of the network. In particular, it has been shown that an adversary that is able to generate many shadow identities can arbitrarily subvert content storage and acquisition [2]. To simplify, these *Sybil attacks* insert malicious entities into the network such that any (or most) content operations are in some way dependent on them.

Existing P2P networks provide little or no defenses against Sybil attacks. One must limit the acquisition of multiple identities to prevent the adversary from exploiting the system. However, the absence of universal facilities for user authentication makes such prevention difficult. For example, one popular countermeasure to mitigate Sybil attacks is to validate the uniqueness of the IP address of the joining node. Such measures are ineffective because of the relative ease with which IP addresses can be spoofed. The realities are generalizable: any solution based on weak authentication of the global and largely anonymous user community is doomed to failure. Moreover, strongly authenticating that community based on universally issued credentials is equally intractable.

This work presents an admission control system for structured P2P networks resilient to Sybil attacks. The system creates and maintains a self-organized hierarchy of participating peers. A joining node appeals to a leaf node of the hierarchy for admission which provides it with a cryptographic puzzle [3]. After solving the provided puzzle, the joining node is redirected to the leaf's parent. This puzzle challenge/solution process is recursively repeated with the parent until the joining node reaches the root. The root node issues the joining node a cryptographic proof of completion of the admission process. This globally verifiable proof, called a *token*, encodes the public key of the joining node and its ID. This token is then used in subsequent operations to prove a node's identity.

The admission control process limits the rate at which a node can obtain IDs by controlling the amount of effort needed to acquire them. While this effort is not overly burdensome on a single node, it makes it difficult for an adversary to acquire a large percentage of IDs. Our analysis shows that an adversary must perform days or weeks of effort to obtain a small percentage of nodes in small P2P networks, and that this effort increases linearly with the size of the network. It takes an adversary just over 3 days to obtain 10% of the IDs in a network of only 8,000 nodes.

II. Related Work

Douceur [2] was the first to consider multiple identity problems in the context of P2P networks. Dubbed the "Sybil" attack, the registration of many new nodes to take control of a system plagues more than just P2P networks. Any distributed system in which an entity can arbitrarily establish identities, is subject to its effects.

The designers of the original structured P2P overlays paid little attention to the severity of Sybil attacks; most schemes either neglect to consider it or include limited defenses. For example, in Chord [9] and Pastry [5], the authors assumed that a node's ID was the hash of its IP address. However, an adversary can simultaneously spoof many IP addresses to quickly obtain a multitude of identities. Additionally, using hashed IP addresses limits access to the network from machines behind NAT boxes. In CAN [4], the authors assumed that nodes pick random IDs when they enter the network. This places trust on all nodes in the system and easily allows an adversary to create many IDs.

Many different types of cryptographic solutions to the Sybil attack have been proposed. While the application of cryptography potentially provides a solution, no current method efficiently mitigates the attacks. Because Sybil attacks result from entities misidentifying themselves, requiring all nodes to authenticate with public keys is a one approach to securing these networks. Douceur [2] showed that without the use of a centralized authority [7] that certifies all nodes, it is impossible to prevent this attack. Srivatsa and Liu [8] suggested the use of certificates with limited lifetime issued by the bootstrap entry point that binds a node with a unique ID. This would limit the number of IDs an adversary can obtain during a time period and will depend on the lifetime of the ticket. However, requiring all nodes to obtain a certificate that will bind it with a unique ID is not only expensive but will require either releasing private information or paying an amount of money for the service. Douceur [2] suggested using node validation by storage, communication and computational challenges. He also found theoretical bounds on the number of IDs an attacker can accumulate if such challenges are used. However, he did not specify how this can be done in a practical system.

III. Admission Control System

In this section, we describe an Admission Control System (ACS) for structured P2P systems. ACS defends against Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. A bootstrap node, located at the root, allows users to join. This creates a tree structure as in Fig. 1. It is important that the upper layer nodes should be both static and trustworthy, particularly for large and long-standing networks. The bootstrap node in this system can be a dedicated server and does not need to be one of the peers.

In Fig. 1, X_0 is the bootstrap node and A is a joining node. Before joining, A must gain admission from a sequence of nodes, starting with leaf node X_n and ending with root X_0. At each stage i, A is required to successfully solve a puzzle presented by X_i.

The remainder of this section uses the following notation:

K_A^+, K_A^-	Node A's public and private keys
ID_A	Node A's ID
R_j	Random value where j is a session ID
TS_i	Time Stamp
X_i	Node at level i
X_{i-1}	Parent of node X_i
K_{X_i}	Secret key known only to node X_i
$K_{X_i,X_{i-1}}$	Shared key between X_i and its parent

Note that $\cdot$ denotes concatenation, $MAC(x,k)$ denotes the keyed message authentication code of data x and key k, and $sig(x,k)$ denotes the signature of x using the private key k.

A. Join Protocol

Before joining the network, node A must generate a public/private key pair K_A^+/K_A^-. When node, A, wishes to join the network, it must first find a leaf node X_n. This is accomplished by consulting a bootstrap node which will randomly select one of the leaves in the system. Next, to gain admission from X_n, A requests a puzzle. After A solves X_n's puzzle, it is given a token. This token is used to prove admission by X_n to X_n's parent. At tree height i, the protocol message flow proceeds as follows:

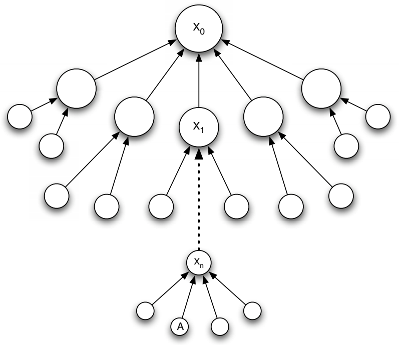


Fig. 1. Example ACS node organization. X_0 is a bootstrap node of the ACS tree of depth n; A is a joining node.

$$\begin{array}{ll}
A \longrightarrow X_i : K_A^+ & \text{(request)} \\
X_i \longrightarrow A : TS_1, h(K_A^+ \cdot TS_1 \cdot R_1), & \text{(puzzle)} \\
\qquad MAC(K_A^+ \cdot TS_1 \cdot R_1, K_{X_i}) & \\
A \longrightarrow X_i : K_A^+, R_1, TS_1, MAC(K_A^+ \cdot TS_1 \cdot R_1, K_{X_i}) & \text{(solution)} \\
X_i \longrightarrow A : ID_{X_i}, TS_1, MAC(K_A^+ \cdot TS_1, K_{X_i,X_{i-1}}) & \text{(token)}
\end{array}$$

In the request phase, A sends its public key K_A^+ which is used to identify A during the joining process. Upon receiving a request, the challenger, X_i, creates a cryptographic puzzle based on a hash function. The hash puzzle contains two parts—a known and unknown part. The unknown part consists of an x-bit random number R_1, where x is an exponentially increasing *hardness* metric for the puzzle. The goal of the solver is to determine R_1, i.e., invert $h()$. As a cryptographic hash function is non-invertible, this requires A to brute force the solution, taking 2^{x-1} attempts on average. This may be extended by making the length, x, dynamic, thereby allowing malleable hardness as circumstances dictate. Note that environments concerned with the computational diversity of nodes can substitute alternative puzzles, e.g. memory-based puzzles [1] which rely on memory-bound computations and not on the actual computational power of a systems.

In order to provide stateless verification of puzzles, X_i couples the puzzle with a MAC of A's public key, a timestamp, and the puzzle solution R_1. When A replies with the solution, it bundles the MAC included with the puzzle. X_i then calculates the MAC based on the received values to verify the puzzle solution. The adversary cannot forge a MAC, because only X_i knows its secret key. The public key data and timestamp are included in the MAC to avoid replay. Adding K_A^+ to the MAC ensures only A can use the puzzle solution.

Once X_i has verified the puzzle solution, a token is given to A. This token is sent to the next level admission node along with a puzzle request. The token largely consists of a MAC keyed with a secret known only by X_i and its parent X_{i-1}. Again, to prevent replay, a timestamp and public key are included in the MAC. Upon receiving the token, X_{i-1} can verify A has been admitted by X_i. This proof of admittance by children is used for all subsequent requests:

$$A \longrightarrow X_{i-1} : K_A^+, ID_{X_i}, TS_1, MAC(K_A^+ \cdot TS_i, K_{X_i,X_{i-1}})$$

When A reaches the root, a final token format is issued by X_0 and an ID is assigned:

$$X_0 \longrightarrow A : ID_A, TS_j, Sig(ID_A \cdot K_A^+ \cdot TS_j, K_{X_0}^-)$$

where ID_A is $h(K_A^+ \cdot R_A)$. The node's identifier is generated from the cryptographic hash of the node's public key and a random value chosen by X_0. The hash of the public key of A and the random value is used instead of the hash of the public key alone to prevent an attacker from generating enough key pairs off-line until a desired ID is found. The hash also uniformly distributes IDs and ultimately provides a balanced distribution of content objects. The final token proves that A successfully traversed the admission sequence and hence is verifiably valid. All nodes are configured with the public key of the root node and therefore can verify that A has ID_A.

B. *Security*

The ACS is designed to limit Sybil attacks, not to prevent them. Sybil attacks are still possible but, as shown in Section V, are very expensive or intractable to mount. There are two attack scenarios of interest: when the attacker is a member of the ACS, and when it is not.

If the attacker is member of the ACS, it can take advantage of its position. Instead of requiring new identities to traverse the entire tree, the attacker can hand out tokens, reducing the number of puzzles that must be solved. Such an attack can be easily detected by the parent of the attacker by observing the rate of token requests. If this rate surpasses a predefined threshold, the node is detected and severed from the tree, causing the entire subtree to rejoin. Because joining happens at a random leaf, the average number of join requests seen by a node depends on the overall average join rate and the node's hight in the tree. Knowing this information helps every node in the system to determine the value of this threshold. We drop the entire subtree because it is impossible to determine which nodes are legitimate. After dropping the nodes from the tree, the next task is to eject them from the P2P network. During the join process, the intermediate tokens stores the path that a joining node has traversed. This includes a series of IDs representing the nodes in the path from the leaf to the root. The path of a node can be stored in the final token provided by the root. Using this, ejecting a full subtree from the P2P network becomes easy; the root simply needs to broadcast a revocation message containing the prefix of the subtree. After receiving this message, nodes remove from their routing tables all nodes with such a prefix in their path.

An attacker who is not a member of the ACS can slowly obtain identities. Each time it will be required to traverse the tree from the bottom up. The cost of solving the puzzles is such that acquiring a significant fraction of nodes, especially if the size of the network is large, is infeasible. An attacker who is not member of the ACS may also choose to acquire many nodes from one location. This attack is limited by ensuring only a small number of tokens are released during a period of time which limits the attack on that location without affecting other parts of the network.

IV. Improvements Over the Basic Protocol

A. *Cut-off Window*

The basic protocol is designed to make obtaining enough IDs to disrupt the normal operation of the network take a long enough time so that it is likely an attacker will be discovered. However, if an attacker is patient and silently accumulates node IDs over a long period of time, it can achieve the required number of IDs to launch a massive attack. To resolve this weakness we propose the enforcement of a cut-off window.

This technique works as follows. In addition to requiring a node to solve puzzles and obtain a token during the joining process, a node is required to perform the same amount of work again after time W from their initial join time to maintain their membership. To do this, we define a token expiration time. A node can anticipate when its token will expire and reacquire a fresh one beforehand. This will allow for uninterrupted operation of the node. However, an attacker with n IDs will have to acquire n new tokens. This will prohibit an attacker from accumulating many IDs.

The main drawback of this approach is that even legitimate users may be asked to do the extra work of reacquiring tokens. By setting the cut-off time, W, properly we can limit the number of good users that must execute the rejoin process to a small percentage who stay in the network for a very long time. Finally, note that IDs are valid over multiple sessions, therefore nodes reconnecting within W do not need to reacquire a token.

B. *Multiple Roots*

The single root represents a single point of failure. Hence outages, DoS attacks, or other events that effect the availability of root could cripple the network's ability to add new identities. Hence, it is highly undesirable to employ a solution that requires a single host to mediate every identity admission.

A straight forward solution is to replicate the root across multiple hosts. In the simplest scheme, multiple instances of the root (holding the same public/private key pair) are placed at strategic locations in the network. Using DNS redirection a joining member would be directed to closest root to complete its admission to the P2P network. This not only will help avoid the single point of failure, but may be necessary for load balancing expensive cryptographic operations, i.e., identity signing.

An alternate solution operates in a similar manner, except that rather than having multiple instances of the same root, the system allow for multiple roots. This would be accomplished by designating a *master root*, which serves as a trusted CA that issues certificates for roots that mediate independent admission tress. Of course, one would have to be careful to balance the size of all the root trees—perhaps by having the admission process itself randomly select or balance root sub-trees.

V. Performance Evaluation

In the following subsections, we evaluate the performance of the protocol and its enhancements in terms of fairness, the difficulty of an attacker obtaining 10% of the nodes in a network, and work required by normal nodes.

We assume that legitimate nodes arrive at the network according to a Poisson distribution with an arrival rate of λ_g. This is a common assumption used to model requests on different servers. Node lifetimes are exponentially distributed with a mean of μ_g. This is a heavy-tailed distribution meaning that in our model a large fraction of nodes stay for a small amount of time. This models actual user behavior because most users will only be in the network for the time it takes to download a file or two and then leave, whereas there will be fewer server nodes which will be part of the network for a long time. Finally, the difficulty of a puzzle is measured by the time it takes to solve it.

In the following, we assume that an attacker is equal in computational power to the average user. To analyze a more powerful attacker we use the notion of multiple colluding attackers. For example, if an attacker is twice as fast as the average user then we consider that there are two colluding attackers and so on. An attacker retains the node IDs it obtains for an infinite time; whenever it obtains a node ID, the attacker will immediately try to obtain another one. In this way, an attacker may accumulate many node IDs over time.

A. Analysis

Puzzles and Fairness: The cost of joining the network for any legitimate node will depend on the time it requires to traverse the tree starting from a leaf up to the root and solving a puzzle for each level. To make this process fair we need to fix the time it takes the average user to join the network. To do this we first set the joining difficulty (measured in average time) to l. We note that, if a node must only solve a single puzzle of average time l, it is possible that it will "get lucky" and solve the puzzle on its first guess. In fact, because the distribution of the time to solve the puzzle is uniform, the variance for the time taken to solve it is high, and hence unfair.

To solve this problem, we divide the puzzle into n smaller puzzles each of difficulty l/n such that the combined average time is l. By dividing the large puzzle to several smaller puzzles we can decrease the variance of the total puzzle solving time from $l^2/12$ to $l^2/12n$.

We use n to be the minimum number of puzzles a node must solve to join the network. If a node is joining on a branch of the tree that has depth $k \geq n$, the puzzle is divided into k pieces, each of average duration l/k. In this case, the variance will be tighter than the minimum requirement. If a node is joining on a branch of the tree that is depth $k < n$, the puzzle is divided into n pieces, and some nodes on the branch will pose more than one puzzle.

Steady state: In the steady state the number of nodes in the network, N, is found by considering the arrival rate of legitimate nodes, λ_g, and mean lifetime μ_g.

$$N = \lambda_g \times \mu_g \tag{1}$$

To be able control fraction f of the nodes, an attacker will be required to obtain $\frac{fN}{(1-f)}$ IDs. If the average joining difficulty is l and there are n attackers, the arrival rate of attacker nodes will be $\lambda_a = \frac{n}{l}$ and the time to launch a successful attack:

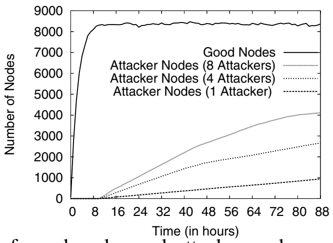


Fig. 2. Number of good nodes and attacker nodes vs. time in a network (attack start at steady state t = 10 hours).

$$T_{attack} = \frac{fN}{(1-f)\lambda_a} \tag{2}$$

Cut-off window: The cut-off window optimization requires each node to reacquire a fresh token after time W. We choose W such that most legitimate users will not be required to reacquire new tokens during their lifetime in the network, but so that attackers will have to relinquish node IDs they have accumulated and perform work to reclaim each one.

Following our assumptions on the arrival rate and node lifetime, the percentage, P, of legitimate nodes that will be required to reacquire fresh tokens can be found as follows:

$$P = 1 - \frac{1}{\mu_g}\int_0^W e^{\frac{-x}{\mu_g}}\,dx \tag{3}$$

If there are n attackers, the combined number of nodes they can accumulate ($N_{attacker}$) is found as follows, assuming that a cut-off window of W is used and the average join time is l.

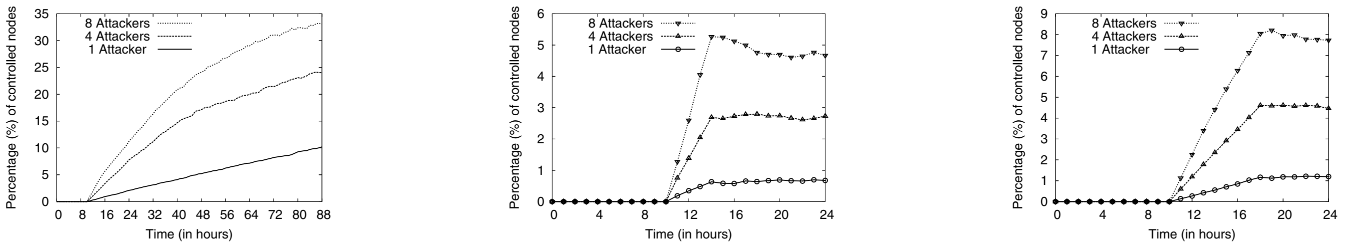
$$N_{attacker} = \frac{n \times W}{l} \tag{4}$$

From (4), for a 10,000 node network, a 5 minute puzzle provides ample protection. To protect against the same number of attackers in a 100,000 node network, the puzzles can be as small as 30 seconds; in a 1,000,000 node network, the puzzle strength required is only 3 seconds. For our simulations in the next section, we choose 5 minute puzzles, because we simulate a small network.

B. Simulation Results

Here we show our simulation results in which we study the resiliency of our protocol against Sybil attacks. We assume that nodes join at random ACS leaves with uniform distribution. Because nodes join through other nodes that are close to them, there could be hot spots where the tree will increase in height faster than other places; in our simulation, we do not consider such cases. The height of the tree is then determined by the order of the tree, the arrival rate and the average node lifetime.

We developed an ACS simulator using Java. The degree of the tree is set to 8 meaning that no node has more than 8 children. The tree initially includes the bootstrap node as the root and two levels of children nodes. These nodes are assumed to have an infinite lifetime. The arrival rate of legitimate nodes is set at 1 node/second. The average lifetime of a legitimate node in the network is exponentially distributed



(a) No cut-off window (b) Cut-off window of four hours (c) Cut-off window of eight hours

Fig. 3. Attack on a steady network (attack starts at t = 10 hours).

with a mean of 2.3 hours. This is consistent with a study performed on the Gnutella P2P network [6].

The average joining time, l, which is the time to traverse the tree and solve the required puzzles was chosen to be 300 seconds which is uniformly distributed. We choose 300 seconds because of the network size; for bigger networks (100,000 and 1,000,000 nodes) puzzles would take only 30 and 3 seconds respectively. Also, note that nodes need not reacquire tokens for multiple sessions within W. We experiment with scenarios that include one, four and eight attackers.

Steady State: In the first experiment, we evaluate our solution when the network is in the steady state. The simulation is run until the number of nodes stabilizes, and then an attack is launched. As shown in Fig. 2, the number of legitimate nodes stabilizes around 8280, which is consistent with our analysis.

The attack starts at $t = 10$ hours. Our results show that one attacker can obtain 10% of total nodes in 77 hours (more than 3 days) whereas four attackers can achieve the same percentage in about 20 hours. We also found that a collusion of eight attackers can get 10% of the nodes in less than 10 hours. Fig. 3(a) shows these percentages as time progresses.

Cut-off Window: From the results of the basic protocol, we can see that although our admission control system is able to greatly limit a single attacker, it does not do a good job when more attackers are involved. The cut-off window is designed to solve this problem.

We simulated scenarios with $W = 4$ and 8 hours and determined how many legitimate users are required to reacquire fresh tokens and the number of IDs an attacker or multiple attackers can maintain. As in the previous experiment, the attack was launched after the network reached steady state. We assume that good nodes refresh their tokens before they expire so they are not cut off the network. The number of good nodes in these two cases remain the same as with the steady state experiment, i.e. around 8280 nodes.

The number of nodes that an attacker can maintain perfectly matches the analytical results we obtain from Equation 4. When $W = 4$, a single attacker is only able to maintain around 48 nodes, four attackers can maintain around 192 nodes and eight attackers can maintain around 384 nodes; all are well under the 10% target. The percentages of nodes attackers can maintain are shown in Fig. 3(b).

To decrease the percentage of legitimate nodes that are required to reacquire fresh tokens during their lifetimes we experimented with a cut-off window of 8 hours. The results show that the percentage of good nodes that need to do the extra work drops to less than 2% while even 8 attackers combined can only maintain around 8% of the nodes, still under the 10% target. Fig. 3(c) shows the percentages of attacker nodes in this case.

Comparing this with the steady state results we can clearly see that the cut-off window optimization greatly improves the limiting capability of our protocol. We see that instead of letting the number of attacker nodes grow without bounds as in the basic protocol, the cut-off window places a limit on this number and prevents it from growing larger. This comes at the cost of requiring some legitimate nodes to reacquire their tokens after some time.

VI. Conclusion

In this paper, we proposed an admission control system that mitigates Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. A node wishing to join the network is serially challenged by the nodes from a leaf to the root of the hierarchy. In this way, we exploit the structure of hierarchy to distribute load and increase resilience to targeted attacks on the admission control system. We also define a cut-off window that provides a provable ceiling to the number of node IDs a computationally bounded adversary can obtain independent of the life and size of the network. This is the first practical method that provides such a hard bound for limiting Sybil attacks.

References

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *Transactions on Internet Technology*, 5(2):299–327, 2005.

[2] J. Douceur. The sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems 200, Cambridge, MA, March 2002.*

[3] R. Merkle. Secure communications over insecure channels. In *Communications of the ACM, 21(8):294–299, April 1978.*

[4] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *SIGCOMM 2001.*

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

[6] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *University of Washington Department of Computer Science and Engineering Tech Report UW-CSE-01-06-02.*

[7] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash table. In *1st International Workshop on Peer-to-Peer Systems, Cambridge, MA, March 2002.*

[8] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *ACSAC 2004.*

[9] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM 2001, August 2001.*