Limiting Sybil Attacks in Structured Peer-to-Peer 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— **Structured peer-to-peer networks are highly scalable, efficient, and reliable. These characteristics are achieved by deterministically replicating and recalling content within a widely distributed and decentralized network. One practical limitation of these 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 admission control nodes. Implemented by the peer-to-peer nodes, 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. In this way, we exploit the structure of hierarchy to distribute load and increase resilience to targeted attacks on the admission control system. We evaluate the security, fairness, and efficiency of our scheme analytically and via simulation. Centrally, we show that an adversary must perform days or weeks of effort to obtain even a small percentage of nodes in small peer-to-peer 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 an IDs continued validity. Finally, we show that participation in the admission control system does not interfere with a node's use of the peer-to-peer system: the loads placed on the nodes participating in admission control are vanishingly small.**

I. Introduction

Structured peer-to-peer networks provide a cooperative, stable, and robust mechanism for storing and retrieving arbitrary content. Recent deployments of networks such as Chord [19], CAN [11], Pastry [12] and Tapestry [21] have reached a massive scale, where millions of user share content over global networks. These networks have been used to successfully construct large-scale applications such as file-sharing, distributed files-systems, and object stores.

User identifiers (IDs) uniquely identify participant endpoints ($nodes$) in peer-to-peer 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 [5]. 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 peer-to-peer 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 validates the uniqueness of the IP address of the joining node. Such measures are ineffective because of the relative ease with which a 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 universal issued credentials is equally intractable [6].

This work presents an admission control system for structured peer-to-peer networks resilient to Sybil attacks. The system creates and maintains a self-organized singly rooted hierarchy of participating peers. Joining nodes appeal to a leaf node of the hierarchy for admission. The node is redirected to the leaf's parent after solving a provided cryptographic puzzle [9]. 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. The node's public key is used in subsequent operations to prove its identity.

We explore further extensions that help reduce the effectiveness of adversarial behavior. These extensions tightly regulate the ID acquisition process by requiring re-assertion of an IDs validity through the *cut-off window*, and employ startup procedures that limit an adversaries ability to control the network while it is forming.

The admission control process limits the rate at which a node can obtain IDs by controlling the amount of effort needed to acquire them. The use of the hierarchy offers several advantages. First, the load of the admission control system is broadly distributed among the peers. This is consistent with the egalitarian nature of peer-to-peer systems: all participants share the load of system maintenance. Second, the structure of the hierarchy makes targeted attacks more difficult to mount. An adversary must appeal to a many node to obtain a new ID. Hence, collusion with singular entities is largely ineffective.

Third, the hierarchy localizes attacks on the admission control process itself. Adversarial nodes who fraudulently redirect many shadow nodes will quickly be found out because of the increased volume of requests arriving at its parent. Finally, the hierarchy affords an efficient countermeasure for removing malicious nodes. The adversary node and its suspect children are removed by simply severing the association between the child and parent.

We evaluate the security, fairness, and efficiency of our scheme analytically and via simulation. The security of this scheme is measured in the adversaries' ability to acquire IDs. Our analysis shows that an adversary must perform days or weeks of effort to obtain even a small percentage of nodes in small peer-to-peer networks, and that this effort increases linearly with the size of the network. It take an adversary just over 3 days to obtain 10% of the IDs in a network of only 8,000 nodes. Moreover, the simulations show a cut-off window of 4 hours prevents a committed adversary from obtaining less than 1% of the IDs in a small network. This analysis indicates that we have achieved the desired level of security: *hundreds or thousands of committed adversaries would be needed to mount a Sybil attack on the contemporary peer-to-peer networks*.

Fairness is measured by the variance in effort required to obtain an ID experienced by nodes joining the network. This is principally measured by range of work required to solve the puzzles a joining receives as part of the admission process. We show analytically that the expected variance can be controlled by requiring larger numbers of (easer) puzzles. In essence, the solution controls fairness by trading off increased puzzle costs, e.g., generating, distributing, and validating puzzles.

The burden of self-organization and admission control is placed on the peer-to-peer nodes themselves. For this reason, the computational load of these activities must be low. Our analysis shows that these costs are vanishingly small for all nodes in the network. For MD5 based HMACS, puzzle overheads are as small as 15 microseconds.

The remainder of this paper is organized as follows. The following section discuss important related work. Section III discusses existing vulnerabilities and our assumptions and system goals. Section V details out basic admission control scheme. Section VI outlines several extensions to the basic scheme. Section VII provides an analysis of the efficiency and effectiveness of the proposed approach. Section VIII concludes.

II. Related Work

Exploitation resulting from mapping multiple identities to one entity is a known problem [4]. Unfortunately, solving such attacks in a distributed environment is nontrivial. Douceur [5] was the first to consider the multiple identity problem in the context of structured peer-to-peer networks. Dubbed the "Sybil" attack, the registration of many new nodes to take control of a system plagues more than just peer-to-peer networks. Any distributed system, including sensor networks [10], where an entity can arbitrarily establish identities, is subject to its effects.

The designers of the original structured peer-to-peer overlays paid little attention to the severity of Sybil attacks; most schemes either neglect to consider it or include limited defenses. In Chord [19], Pastry [12] and Tapestry [21], the authors assumed that a node's ID was the hash of its IP address. However, this assumption is weak and will not hold against IP address spoofing. Addionally, using hashed IP address limits access to the network from machines behind NAT boxes. In CAN [11], the authors assumed that nodes pick a random ID when it enters 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 [5] showed that without the use of a centralized authority [17] that certifies all nodes, it is impossible to prevent this attack. Srivatsa and Liu [18] suggested the use of certificates with limited lifetime issued by the bootstrap entry point that bind 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. Decentralized mechanisms for limiting Sybil attacks are therefore more palatable.

Douceur [5] suggested using node validation by storage, communication and computational challenges but did not specify how this can be done. Cryptographic puzzles were first proposed by Merkle in his seminal work [9], but he only considered using them in key agreement and not for access control. Since Merkle's proposal, researchers have found many uses for the puzzles [1], [7], [3]. Wallach [20] has considered using cryptographic puzzles and multi-part bit commitment but they are open to an attacker rejoining the network.

In a different sort of computational puzzle, Castro et al. [2] consider the possibility of having a distributed node ID generation scheme that would limit the rate in which an attacker can obtain IDs. This is done by requiring prospective nodes to generate a private/public key pair such that the hash of the public key has the first p bits equal to zero. They also suggest binding the IP address of the node with its ID. To over come the possibility of an attacker to accumulate node IDs they suggest periodically invalidating node IDs and using different setting for the hash initialization. However, this would require legitimate nodes to obtain new IDs every time this happens.

Finally, Saxena, Tsudik, and Yi [14] studied several ways of admission control in peer-to-peer networks based on threshold cryptography. However, their method does not limit the number of IDs a malicious node can obtain because there is no cost associated with obtaining an ID.

III. Background

In this section, we first give a brief background on structured P2P network taking Chord as an example to look at. We then discuss the vulnerabilities of P2P networks that an attacker can take advantage of by launching a Sybil attack.We also define the attacker model.

A. Structured P2P Networks

Structured P2P networks were designed to solve the scalability problem and uncertainty when performing searches that face today's unstructured networks. The contents in structured P2P networks are not placed on random nodes as in unstructured networks, but are assigned to particular nodes which will allow for effective lookup. To achieve this, structured P2P networks implement an abstraction of Distributed Hash Table (DHT) to map objects to nodes in a deterministic way. Each object will have a unique *key* with an associated *value* and each node will have a unique *ID*; both the key and the node ID are picked from the same ID space. The IDs and keys are chosen randomly with a uniform distribution from the ID space to ensure balanced load among nodes. A strong one-way function (like MD5 [8] or SHA-1 [15]) is used to obtain a node's ID from its IP address. For an object the key can be the hash of the content of the object. Each key is mapped by the overlay structure to a single node according to its ID. The value associated with a key can either be the object itself or just a pointer to another node that has the object. To increase availability keys are replicated and stored in other nodes according to a replica function. Each node will maintain a *routing table* that hold information about other nodes in the network including their IDs and IP addresses. Using this table, requests to objects can be forwarded to the nodes that holds the corresponding keys.

Chord [19] is an example of a structured P2P network is that assigns both object keys and node IDs from the same one-dimensional space (160-bits). This space wraps around and forms a ring; a node's ID will indicate its location on this ring. Each key is stored in its *successor* which is the node with the smallest ID greater than the or equal to the key. Each node need only to know its successor for the lookup scheme to work correctly. A query for a key will be forwarded clock-wise until the the key's successor is found. Of course, this would require $O(N)$ hops in a network with N nodes. To improve this the routing table (in Chord it is called *finger table*) of each node will contain the addresses of nodes other than just the successor. The *i*th finger of node n will contain the address of the node with smallest ID larger than $n+2^{i-1}$. The first entry of this table will be n's successor, the next one will be double the distance away and so on. This will allow the lookup to be performed in $O(logN)$. Each node will also know the address of its predecessor. The predecessor is used to initialize the finger table when a new node joins the network.

When node n is ready to join the Chord ring, it contacts a bootstrap node which helps it find a node already in the network. Then, n will ask that node to find node k which is successor of n's ID. When node k is found, n will contact it and gets its predecessor information which n then uses to initialize its finger table. Now, n will become predecessor of k. To ensure correct operation even with node failures, each node will maintain a *successor list* which will contain addresses of the closest r successors so that if the one hop successor failed the next successor can be used. Finally, replication in Chord is done by assigning a replica of a key to each of its root's r closest successors.

B. Vulnerabilities

Because structured P2P networks such as Chord take very limited measures against a Sybil attack, an attacker can obtain many IDs and hence many nodes in the network. This will allow an attacker to take advantage of two major vulnerabilities from which such networks suffer, the *routing mechanism* and the *object serving mechanism*.

In structured P2P networks, the routing mechanism follows a specific path to its destination and require the cooperation of the peers on that path. Hence, a single malicious node in that path can cause the whole lookup process to fail. For example, in [2] the authors show that in a network with 100,000 nodes if an attacker was able to compromise 10% of the nodes then the success rate of a the routing process will drop to 65%. Also, when an insertion of an object is done without secure routing, the application cannot be sure that the object was placed on a legitimate node (on its successor for example) and it cannot know if the replicas for that object are all placed on correct nodes. Even if techniques like secure P2P routing [2], which uses alternative lookup paths, is used, an attacker can still reduce the success rate of the routing process by obtaining more nodes. For example, with secure P2P routing an attacker can cause notable failures in the lookup process the network by compromising 25% of the nodes instead of 10% when normal routing is used. Another vulnerability related to the routing mechanism is populating victim nodes' routing tables with compromised nodes. This would allows an attacker to control the communication from the victim node and can isolate that node from the network. This type of attack is called the Eclipse Attack and is discussed in [16]. To take advantage of this vulnerability an attacker would first obtain many nodes in the network by launching a Sybil attack then use these nodes to populate the routing tables of correct nodes.

The second main vulnerability is the object serving mechanism. Since objects in structured P2P networks are mapped to different nodes in the network, an attacker who controls the node to which an object is mapped can deny serving it or serve an incorrect one. By doing this the attacker is able to hide that object from other legitimate users. Of course replication will make it more difficult for an attacker to do this hiding but it is still possible. For example in Chord, to be able to hide an object that is not replicated, an attacker must obtain the ID which is the closest to the object key. If replication is used and r replicas exist then the attacker will need to obtain the r IDs closest to the replicas. Because an attacker cannot choose its own ID which would require inverting a one-way hash function, the only way to hide object would be to launch a Sybil attack and obtain many IDs hoping that the required IDs are obtained. Although this is not easy to achieve, it is still possible as shown in [18].

C. Attacker Model

The attacker's purpose is to disrupt the correct operation of the network by misleading other non-malicious nodes and by not following the protocol specifications. The underlying IP network in our model can be viewed as an open broadcast medium in which every packet can be heard by the attacker. An attacker is also assumed to have the ability to spoof any IP address and to sniff traffic destined to any other node. With regard to an attacker's computational capability, we assume that it can have slightly more computational power compared to other nodes but not infinite so that it will encounter some puzzle solving cost when joining the network. Finally, we assume that an attacker can be aware of other attackers and hence can collude with them against the system.

The network in our model consists of N nodes and an attacker can be controlling fraction f $(0 \leq f < 1)$ of the nodes. As this fraction increases the attacker will be in control of more nodes and hence becomes capable of doing more damage.

IV. Design Goals

In this section we discuss our design goals for the admission control system in P2P networks which are ensuring end user's privacy, scalability, efficiency, response to attacks and verifiability.
Security: An admission control system should provide resiliency against attacks that work by creating shadow nodes, such as the Sybil attack we are considering here. It should make such attacks hard to mount and reduce their effectiveness.
Efficiency: An admission control system should be simple and does not require a lot of overhead on participating nodes. This makes techniques that depend heavily on public key cryptography not desirable.
Fairness: An admission control system should be fair in terms of the joining cost. Nodes should do an equal amount of work in order to join the network. This goal prevents an attacker from taking advantage of a weak point in the systems.
Response to attack: An admission control system should provide techniques to response to a Sybil attack that would make the attack more difficult while not affecting other legitimate nodes.
Privacy: An admission control system should not require end users to reveal any private information in order to join the network. This means providing a node with a certificate that requires reveling to an authority some private information such as the user's name or a credit card number is not desired.
Scalability: An admission control system must scale to very large networks which means that it should distribut the work among nodes.
Verifiability: An admission control system should allow any node in the system to verify that a it is communicating with a node that was legitimately admitted to the network.

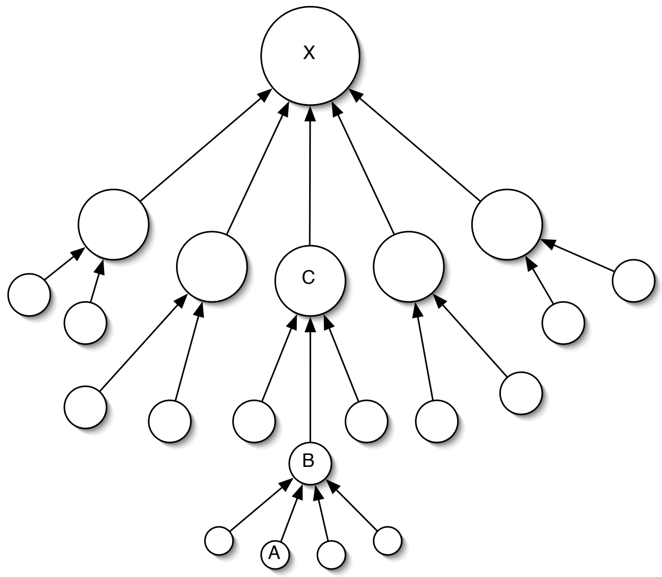

Fig. 1. Example ACS node organization. X is a bootstrap node; A is an joining node

V. Solution Details

In this section we describe an Admission Control System (ACS) for structured peer-to-peer systems. ACS defends against Sybil attacks by adaptively constructing a hierarchy of cooperative admission control nodes. A bootstrap node, located at the root, allows trusted nodes, such as major ISPs, to join the network. These nodes allow smaller providers or companies to join as immediate children, which in turn allow more users to join. This continues, creating a tree structure as illustrated in Figure 1. It is important that the upper layer nodes are both static and trustworthy, particularly for large and long-standing networks such as Gnutella.

In Figure 1, X is the bootstrap node and A is an joining node. Before joining, A must gain admission from a sequence of nodes, starting with leaf node B and ending with root X. At each stage, A is required to successfully solve a puzzle presented by B. As supported by the analysis presented in the latter sections of this paper, it is infeasible for an adversary to obtain a sufficient number of identities to mount a Sybil attack in this decentralized, multi-puzzle scheme.

We use the following notation throughout the remainder of this section:

K_A^+, K_A^-	node A's public and private key
ID_A	h(K_A^+), for hash function h
R_i	Random value where i is a session ID
TS_i	Time Stamp
$P(ID_n)$	Parent of node with ID_n
B	Leaf node
K_{ID_n}	Secret key known only to node ID_n's
$K_{ID_n,P(ID_n)}$	Shared key between ID_n 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 Setup

Before joining the network, a node A must generate a public/private key pair K_A^+/K_A^-. The node's identifier is the cryptographic hash of the node's public key, e.g., using MD5 [8] or SHA-1 [15]. This prevents a node from choosing its own ID, uniformly distributes IDs, and ultimately provides a balanced distribution of content objects. All nodes are configured with the public key of the root node.

B. Join Protocol

When a node, A, wishes to join the network, it must first discover a leaf node B. This is accomplished either by consulting a bootstrap node or using an automatic service discovery mechanism. Next, to gain admission from B, A requests a puzzle. After A solves B's puzzle, it is given a token. This token is used to prove to B's parent admission by B. The protocol message flow follows:

$$\begin{array}{lr} A \longrightarrow B : K_A^+ & \text{(request)} \\ B \longrightarrow A : TS_1, h(ID_A \cdot TS_1 \cdot R_1), & \text{(puzzle)} \\ \qquad\qquad MAC(ID_A \cdot TS_1 \cdot R_1, K_{ID_B}) & \\ A \longrightarrow B : ID_A, R_1, TS_1, MAC(ID_A \cdot TS_1 \cdot R_1, K_{ID_B}) & \text{(solution)} \\ B \longrightarrow A : ID_l, TS_1, MAC(K_A^+ \cdot TS_1, K_{ID_B, P(ID_l)}) & \text{(token)} \end{array}$$

In the request phase, A sends its public key K_A^+. Upon receiving a request, the challenger, B, 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 the value R_1 from its hash, e.g., invert $h()$. As a cryptographic hash function is non-invertible, A must use a brute force method taking, on average, 2^{x-1} attempts. An extension to the above message flow, would indicate the dynamic length x, thus making hardness malleable as circumstances dictate. We use this extension to provide protocol enhancements discuss in Section VI.

In order to provide stateless verification of puzzles, B couples the puzzle with a MAC of A's ID, a time stamp, and the puzzle solution R_1. When A replies with the solution, it includes the MAC included with the puzzle. The verifier calculates the MAC based on the received values to verify the puzzle solution. The adversary cannot forge a MAC because only B knows its secret key.

The ID data and timestamp are included in the MAC to avoid replay. Consider the case without either of these values. Failing this, an eavesdropping adversary could replay an overhead puzzles solution and MAC pair. As sent MAC is keyed with B's secret key, the solution will be accepted. Adding the identifier ID_A to the MAC ensures only A can use the puzzle solution, but it does not prevent all forms of replay. Without a time stamp, an adversary only needs to complete the puzzle path once. After becoming disconnected, possibly due to malicious behavior, the adversary needs only to resubmit old puzzle solutions. In doing so, it quickly returns to its connected state. However, puzzle solutions are only valid for a given time in the proposed approach. Hence, the scheme is resilient replay.

Once B 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 B and its parent. Again, to prevent replay, a time stamp and identification data are included in the MAC. When the parent receives the token, it can verify A has been admitted by B. This proof of admittance by children, used for all subsequent requests, generalized as:

$$A \longrightarrow P(ID_n) : K_A^+, ID_n, TS_1, MAC(K_A^+ \cdot TS_i, K_{ID_n, P(ID_n)})$$

At each stage, A is given a token to be used as proof of previous puzzle solution. When A reaches the root, a final token format is issued by X:

$$X \longrightarrow A : Sig(K_A^+ \cdot TS_j, K_X^-)$$

This final token proves A as successfully traversed the admission sequence and hence is verifiably valid. Using the original MAC based token is no longer appropriate, because it would require all nodes to know the secret key. Hence, a public key signature is the necessary, albeit computationally expensive, choice.

After receiving an admission token from the root, A attempts to connect to the network in the location defined by the P2P protocol[1]. In order to connect, A must prove to its prospective neighbors that it has been admitted by the root node X. As signature verifications require significantly more overhead than a MAC calculation, the neighboring nodes each require A to solve one more puzzle before validating the root's signature. These puzzle challenges protect neighbors from a DoS attack.

C. Node Upgrade

A does not immediately become part of the admission process after joining the network. A must prove its stability before inclusion in the ACS. Initially, A joins the ACS as a leaf node. The leaf evaluates A's stability in one of two ways – storing the token timestamp in a child table, or periodically probing A. Once the leaf is convinced A is stable and has been part of the network for a long time, e.g., several hours, it will be considered for upgrade.

In order to maintain a balanced tree, a leaf node only upgrades nodes when its number of children has reached the degree of the tree, i.e., for a k-ary tree, a leaf will upgrade its children only after it has k stable nodes. After the upgrade, the children are promoted to ACS leaf nodes and can join in the admission process. At any point when a node associates with a new parent, an authenticated Diffie-Hellman exchange is used establish the shared key used to validate MAC based tokens.

[1]Node organization is commonly based on node identifiers

Note that the ACS may wish to more aggressively promote nodes. This likely occurs during startup or during periods when local or global outages cause many nodes to be dropped from the tree. Conversely, the tree may at other times choose not to add nodes when it is deemed sufficiently deep to support the join load and achieve the proper security guarantees. These deep issues are the subject of policy and system configuration, and are explicitly deferred to future work.

D. Node Departure

A node can leave the network in two ways – gracefully or due to a failure. When a node leaves the network and it is not a member of the ACS, no change in the system structure is required[2]. If the node leaving the network is a member of ACS and has children, the tree must be restructured.

An ACS node that leaves gracefully will inform its children. The oldest child (in terms of time in the network) is chosen to replace the departing node. To ensure that every node has the same degree, the replacement occurs consecutively, i.e., the node that replaced the departing node will itself be replaced by the oldest of its children. This continues until a leaf node is reached.

In the case of failure, the children must rejoin the network to reestablish their connection to the tree. The rejoining process is as follows: a node n that discovers the absence of its parent will try to contact its grandparent. If this fails, it tries to find another node in the ACS and provide it with its token. The node can either accept n as a child or ask it to reinitiate the join process.

E. Security

The ACS is designed to limit Sybil attacks, not to prevent them. Sybil attacks are still possible but, as shown in Section VII, 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 the token requests. If this rate surpasses a predefined threshold, the node is detected and severed from the tree, causing the entire subtree to rejoin. The entire subtree is dropped, because it is impossible to determine which nodes are legitimate.

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. By guaranteeing that the joining nodes all solve the same number of puzzles with the same difficulty, there will be no advantage point for an attacker.

[2]Nodes not part of the ACS still exist in the child table of ACS leaf nodes. Removing a node from this table has negligible effect

Regardless of an attacker's affiliation with the ACS, there is the potential for DoS. As mentioned in the protocol description, signature verification of the root provided token is the most intensive task. In order to prevent DoS through bogus tokens, neighbors require the requester to solve a puzzle. Verifying this puzzle is reduced to a simple MAC. Preventing a DoS attack exploiting MAC computations is orthogonal to the problem discussed in this paper.

Finally, the root of the tree is a single point of failure. Any attack that affects the fidelity of the bootstrap node will negatively affect the operation of the ACS. Therefore, bootstrap nodes must be fault-tolerant, e.g., hot swappable backups. In the occurrence of the bootstrap and all backup nodes failing, our system can rely on the level one nodes, which as discussed are service providers.

VI. Improvements Over the Basic Protocol

In this section we propose two improvements over the basic protocol. First, we propose a cut-off window mechanism to limit the maximum number of IDs an attacker can accumulate over time. Second, we propose a start-up algorithm to protect against attacks that are launched during network initialization, when due to the small size of the network, it is particularly susceptible to attack.

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 the perform same amount of work again after time W from their initial join time to maintain their mebership. 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. We calculate how many IDs an attacker may maintain using this impartment in the analysis section.

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. We determine this value of W in the analysis section.

B. Startup Up Window

The basic protocol provides minimum protection of the network during the startup process when it has small number of nodes. This is because an attacker can obtain a large percentage of nodes in a shorter time. For example, if the network has 36 nodes, an attacker needs to obtain 4 nodes to be in control of 10% of all the nodes. If we assume that it takes 5 minutes to get an ID, the 10% target can be achieved in less than 20 minutes.

A naive solution is to make the puzzles at the starting phase very difficult, and then decrease the difficulty linearly as nodes join. For example, if the initial puzzle takes an average of two hours to be solved, then after one node joins the puzzle difficulty is reduced to 1 hour and 50 minutes. The drawback of this scheme is that network initialization time will be high.

A better scheme is to define a start up window that impacts the joining process for a finite time. Puzzle difficulty in this scheme decay over time as opposed to the naive scheme which reduces the puzzle difficult as the number of nodes grow. For example, nodes joining the network at its inception are given puzzles that take two hours to solve. Nodes that join five minutes after inception are given puzzles that take 1 hour and 55 minutes to solve. This continues until we reach the puzzle difficulty targeted for the normal join process. In this example, after the two hour start-up window finishes, the network will see a sharp increase in its membership; the number of nodes that an attacker can obtain during this window will be very small compared to the remaining nodes. The number of node IDs an attacker may obtain during this start up window depends on the arrival rate of the nodes and how much more powerful the attacker is compared to the average user.

This scheme results in a much shorter network initialization time compared naive scheme we mentioned above. The analysis of this scheme is shown in the next section.

VII. 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. In the first subsection we present analytical results for fairness, steady-state operation, and the impact of the cut-off window. In the following subsection we present simulation results for these cases and show the impact of the start-up window. We conclude this section with a discussion on expected performance of this protocol in large peer-to-peer networks.

A. Analysis

We assume that legitimate nodes arrive at the network according to a Poisson distribution with an arrival rate of λ_g. The lifetime of each node is exponentially distributed with a mean of μ_g. We The difficulty of a puzzle is measured by the time it takes to solve it.

In the following analysis 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.

1) 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. 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 time is l. By dividing the large puzzle to several smaller puzzles we can decrease the variance of the total puzzle solving time. We know that the variance σ^2 of a uniformly distributed puzzle of length l is

$$\sigma^2 = \frac{l^2}{12} \tag{1}$$

If n puzzles of length l/n are used, the variance becomes:

$$\sigma_n{}^2 = \frac{l^2}{12n} \tag{2}$$

For example, if we desire a puzzle that takes 5 minutes (300 seconds) to solve with a maximum standard deviation of 30 seconds (standard deviation = σ), then n is equal to 8.33. So, the puzzle should be divided into 9 puzzles each takes 33.3 seconds.

We use n to be the minumum 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.

2) 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.

$$N = \lambda_g \times \mu_g \tag{3}$$

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 Sybil attack becomes:

$$T_{attack} = \frac{fN}{(1-f)\lambda_a} \tag{4}$$

For example, if $\lambda_g = 1$ node/sec, and $\mu_g = 2.3$ hours, the steady state number of nodes is 8280 (we discuss the derivation of these parameters in the following section). For the attacker to control 10% of the total nodes in the network it is required to obtain 920 IDs. If the joining process takes on average 5 minutes, a successful attack would take 76 hours which is more than 3 days.

3) Cut-off window: The cut-off window optimization is proposed to defend against a silent attacker that may accumulate many IDs over a long period of time. It requires each node to reacquire a fresh token after time W. As mentioned above, the choice of this cut-off window time is critical because if it is not chosen properly many legitimate users will be required to do this extra work which is clearly undesirable. 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. In effect, this limits the number of node IDs an attacker may obtain to the number at which the work required to maintain the nodes IDs by reacquiring tokens occupies 100% of the attackers time, i.e., an attacker has no time left to solve puzzles for new node IDs.

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:

$$P = 1 - \int_0^W f(x)dx \tag{5}$$

where $f(x)$ is the *pdf* of the lifetime of legitimate nodes. Since we are using an exponential distribution to model node lifetime:

$$P = 1 - \frac{1}{\mu_g}\int_0^W e^{\frac{-x}{\mu_g}} dx \tag{6}$$

For example, if μ_g = 2.3 hours and W = 4 hours, the percentage of Legitimate nodes that will be cut off the network and asked to rejoin is 17.5%. This value turns out to be conservative as shown in our simulation results. Increasing W results in fewer nodes being required to do extra work with the tradeoff that an attacker can accumulate a larger number of nodes.

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

$$N_{attacker} = \frac{n \times W}{l} \tag{7}$$

For example, if the average join time is 5 minutes and $W = 4$ hours, the maximum number of nodes an attacker can accumulate is 48 nodes.

B. Simulation Results

In this section we show our simulation results in which we study how resilient is our protocol against Sybil attacks. We assume that nodes arrive at the network according to a Poisson distribution. This is a common assumption used to model requests on different servers. Node lifetimes are exponentially distributed which is heavy-tailed; 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.

We also 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 amount of time a node spends on the network, i.e. its lifetime.

We developed an ACS simulator using Java. We assume that the degree of the tree is 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. This results in a stable tree of 73 nodes. These nodes are assumed to have an infinite lifetime and hence will not fail or leave the system. The arrival rate of legitimate nodes is set at $\lambda_g = 1$ node/second (unless specified otherwise). The average lifetime of a legitimate node in the network, μ_g is expontentially distributed with a mean of 2.3 hours. This is consistent with a study performed on the Gnutella P2P network [13]; the study shows that over a period of 60 hours, the average session time was 2.3 hours. We do not consider any time of day effect in this simulation.

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. From a usability point of view, we believe that this time is tolerable by most users. Having a longer time will discourage the users from participating in the network while having a shorter time will decrease the limiting capabilities of the protocol. This value is used during all experiment except when evaluating the start-up window, in which case it varies during the start-up period.

We experiment with scenarios that include one, four and eight attackers. Multiple attackers can be thought of either as colluding attackers or a single attacker that is many times faster than the average user. Attacker nodes are assumed to have infinite lifetimes so they do not leave the network nor do they fail. The main goal of an attacker is to obtain as many IDs as possible, so we assume that as soon as one ID is obtained a new joining process is started. An attack is considered to be successful if the attacker is able to obtain 10% of the total nodes (i.e attacker nodes plus remaining user nodes). As previously discussed, by controlling 10% of the nodes an attacker can easily disrupt the routing process. This is a conservative choice since with the use of techniques such as secure P2P routing an attacker must control 25% of the nodes to significantly impair a network.

1) 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

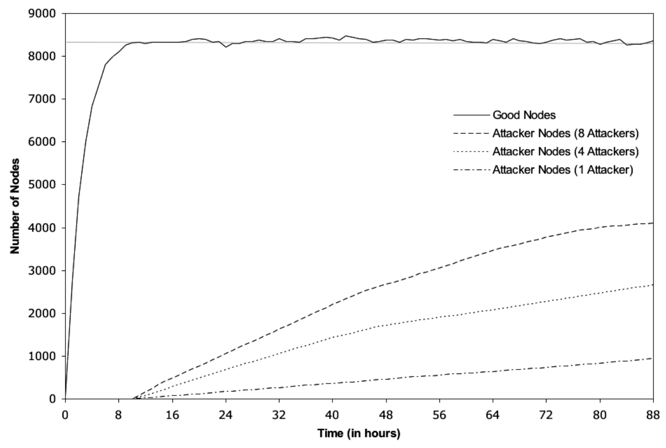

Fig. 2. Number of nodes vs. time (attack start at steady state T = 10 hours)

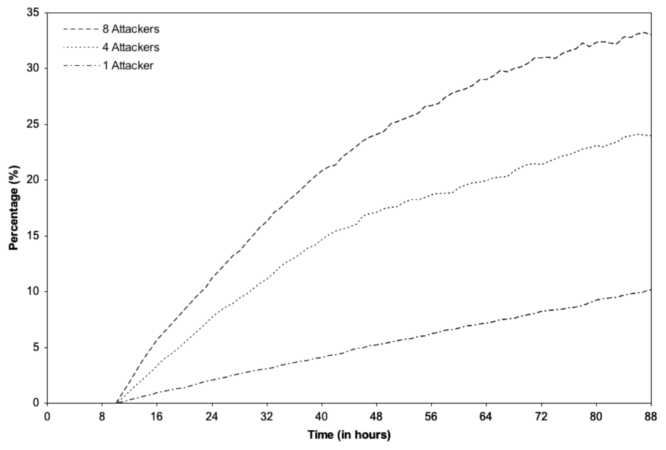

Fig. 3. Percentage of attacker nodes (attack start at steady state T = 10 hours)

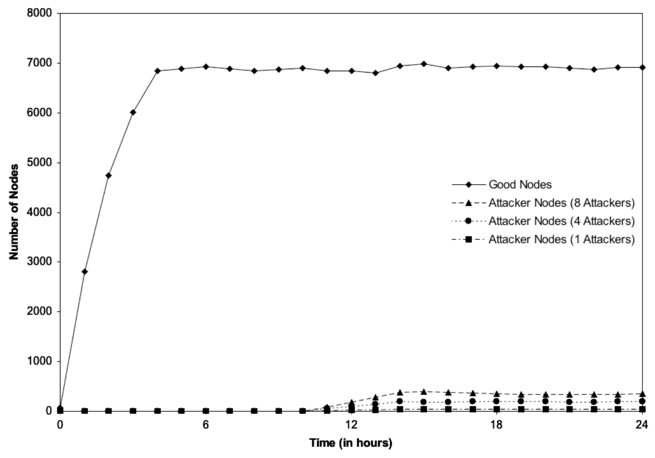

Fig. 4. Number of attacker nodes when a cut-off window of four hours is used (attack starts at t = 10 hours)

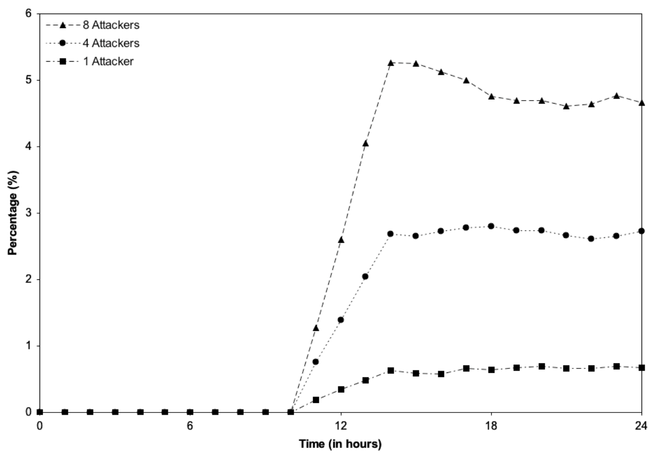

Fig. 5. Percentage of attacker nodes when a cut-off window of four hours is used (attack starts at t = 10 hours)

an attack is lauunched. As shown in figure 2, the number of legitimate nodes in the network stabilizes around 8280 (shown as unbroken line), which is consistent with our analysis.

The attack starts at $t = 10$ hours. Our results show that a single attacker can obtain 10% of the 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. Figure 3 shows these percentages of as time progresses.

2) Cutoff 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 $W = 8$ and determined how many legitimate users are required to reaquire fresh tokens and the number of IDs an attacker or multiple attackers can maintain. Figure 4 shows the number of good nodes in the network compared to the number of node IDs what one, four and eight attackers can maintain. As in the previous experiment, the attack was launched after the network reached steady state.

The analysis in the previous section shows that with the simulated settings the percentage of nodes required to refresh tokens is equal to 17.5% (equation 6). However, in our simulation the average number of nodes that were required to refresh tokens was around 640 nodes/hour which is less than 10% of the nodes. The difference between the simulation result and the analytical result is apparent because in our simulation we do not count the nodes that are required to reacquire fresh tokens more than once during their lifetime. We notice that as soon as the network starts the cut-off process the number of nodes stabalizes.

The number of nodes that an attacker can maintain perfectly matches the analytical results we obtain by using equation 7. A single attacker is only able to maintain around 48 nodes, four attackers can maintain around 192 nodes and eight attackers

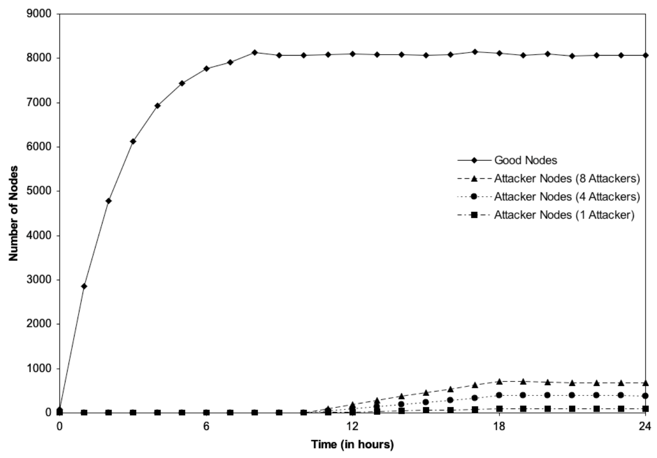

Fig. 6. Number of attacker nodes when a cut-off window of four eight is used (attack starts at t = 10 hours)

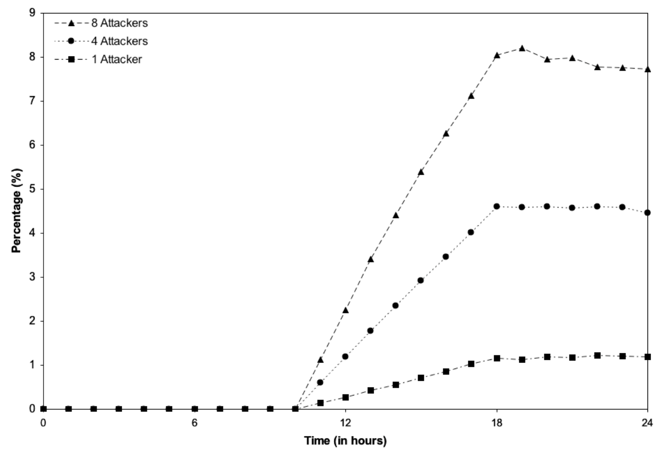

Fig. 7. Percentage of attacker nodes when a cut-off window of eight hours is used (attack starts at t = 10 hours)

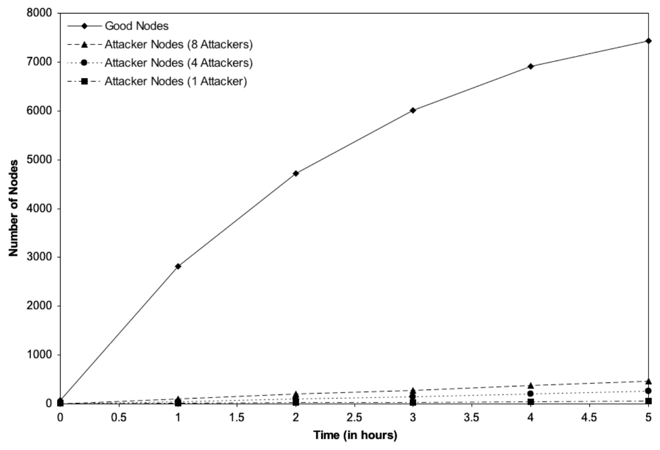

Fig. 8. Number of nodest vs. time with attack starting at time 0

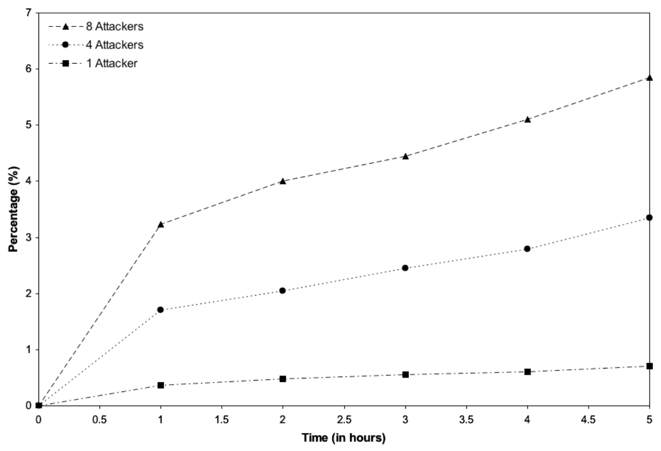

Fig. 9. Percentage of attacker nodes vs. time with attack starting at time 0

can maintain around 384 nodes; all are well under the 10% target. The percentage of nodes attackers can maintain are shown in figure 5.

To decrease the percentage of good nodes that are required to reacquire fresh tokens during their lifetimes we experimented with a cut-off window of 8 hours (see figures 6 and 7). The results show that the percentage of good nodes that need to do the extra work dropped to less than 2% while even 8 attackers combined can only maintain around 7% of the nodes, still under the 10% target.

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 with no 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.

3) Start-Up Window: As previously discussed, a network can be vulnerable during its startup phase because there are only few nodes which makes obtaining the target percentage of node IDs easy. To verify this we ran the following experiment; we start the network and instead of waiting until it reaches steady state we let the attacker start acquiring nodes from time $t = 0$. In figures 8 and 9 we can see the number of nodes that an attacker can obtain during this time and what percentage of the total nodes this constitutes. We note that by the fifth hour a single attacker is able to obtain only 0.7% of the nodes whereas 4 attackers are able to obtain about 3.4%. However, the vulnerability is clear when a more powerful attacker is present as 8 attackers combined can get close to 6% of the nodes in just 5 hours.

To limit the attack on a starting network we proposed using a startup window in which joining nodes are provided with difficult puzzles that decay in difficulty over the window time. In our simulation we set the difficulty of puzzles that are provided to nodes at the beginning of the window to an average of two hours. Average puzzle difficulty is then decreased every

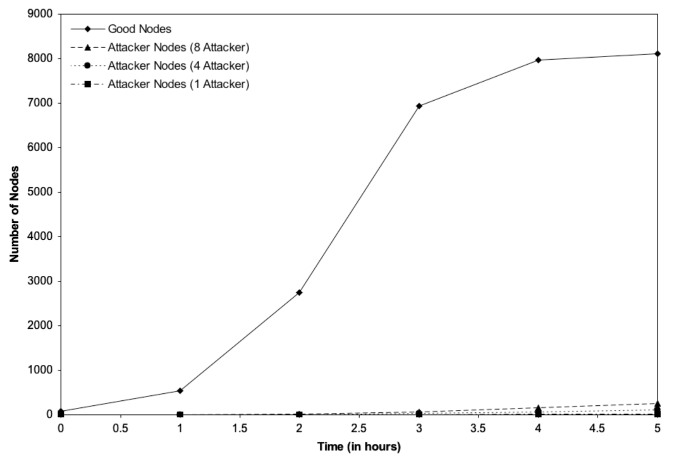

Fig. 10. Number of nodes when a startup window of two hours is used (λ_g = 1 node/sec)

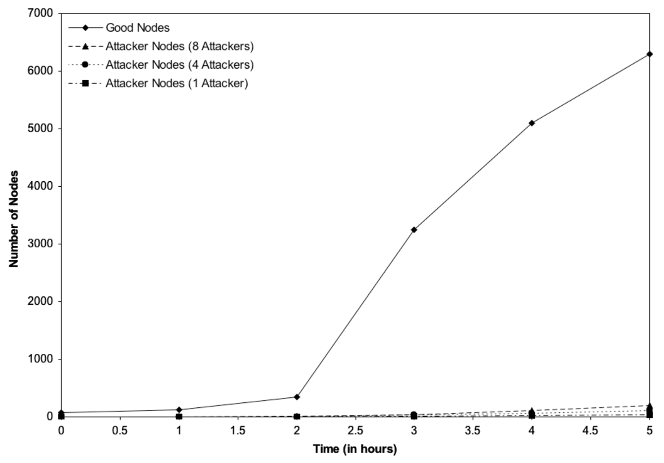

Fig. 12. Number of nodes when a startup window of two hours is used (λ_g = 0.1 node/sec for the first two hours then becomes 1 node/sec)

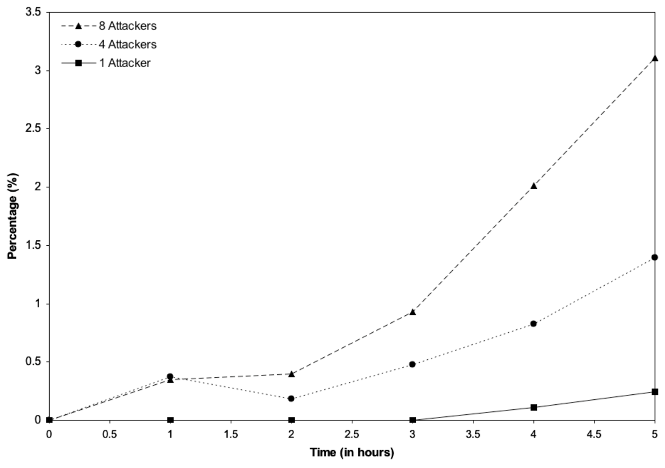

Fig. 11. Percentage of attacker nodes when a startup window of two hours is used (λ_g = 1 node/sec)

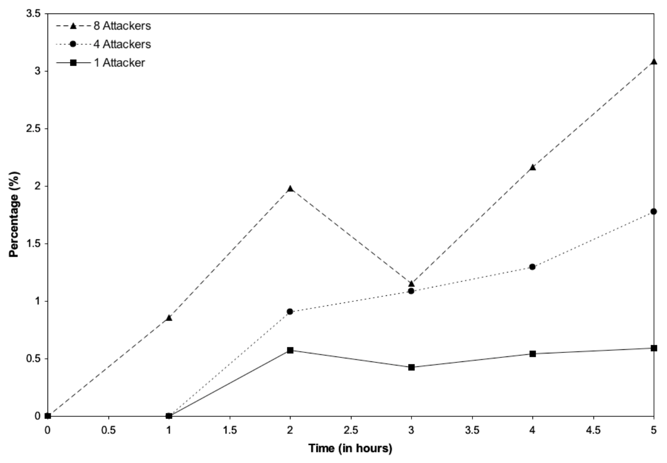

Fig. 13. Percentage of attacker nodes when a startup window of two hours is used (λ_g = 0.1 node/sec for the first two hours then becomes 1 node/sec)

5 minutes such that it takes 5 minutes less time to solve. Figure 10 shows the effect of having a two hour startup window on the number of good nodes and attacker nodes; figure 11 shows the respective percentages of attacker nodes to total nodes. Comparing this to the case where no startup window is used we note that for 8 attackers the percentage of obtained node IDs after five hours decreased from around 6% to around 3% which means that using this optimization cuts the attacker effectiveness in half.

We understand that a long joining time may discourge users from joining the network at its startup phase so we experiment with a lower arrival rate of legitimate users during the startup window. We set λ_g to 0.1 node/second for the first two hours then increase it to 1 node/second afterwards. The results are shown in figures 12 and 13. The main observation is that even under these conditions using a startup window is better compared to the basic case since it reduces the percentage of nodes an attacker can obtain.

C. Discussion on Large Networks

In this subsection we apply our analysis to large peer-to-peer networks. We find that the protocols proposed here increase in resiliency as the size of the network grows. Intuitively, the reason is that as networks grow, attackers must compromise more nodes to gain control of the network.

Consider a large network at steady state. If λ_g, the arrival rate of legitimate users is 10/sec, using equation 3, the steady state number of nodes in the network will be 82,800. In this case an attacker must obtain 9,200 node IDs to control 10% of the network. A single attacker solving five minute puzzles requires 760 hours, or more than one month to obtain this many node IDs when the basic protocol is running.

If the cut-off optimization is used, a single attacker cannot achieve this value if W is set properly. This is because the number of IDs a node can obtain is only a function of the puzzle difficulty, l, and W, so as the network grows, this value will become an increasingly small percentage. Likewise, the inconvenience cause to a legitimate user is a function of only μ_g and W, not the node arrival rate. We are able to set W to a higher value in large networks because we can tolerate a higher absolute number of compromised nodes. This in turn will mean that the cut-off protocol causes inconvenience to fewer legitimate users.

For example, in an 82,800 node network, 9,200 nodes must be compromised to gain control of 10% of the network. Using equation 7, if $l = 5$ minutes, W can be set to over 70 hours to limit a single attacker from obtaining 10% of the nodes. To be resilient against 10 attackers, $W = 7$. At this value, using equation 6, the percentage of legitimate users required to reacquire fresh tokens during their lifetime in the network is less than 5%.

D. Overhead

To evaluate the overhead imposed on nodes participating in admission control we measured the time it takes to complete the required cryptographic operations. We performed the experiment on an Intel Petium III with 800MHz CPU and found the following values: an MD5 based HMAC takes 5 microseconds, SHA-1 based HMAC takes 7.5 microseconds, and RSA-1024 private key operation takes 9.4 milliseconds.

There are three basic operations that an ACS node performs per join request; three SHA-1 or MD5 keyed hash operations to generate the puzzle, verify the previous token and generate the token. Which combined takes 15 microsconds for MD5 or 22.5 microseconds for SHA-1. Clearly, this is very low overhead. The root performs two MD5 or SHA-1 operations to verify the previous token and generate the puzzle. It will also perform one private key operation when generating the final token. The total cost is dominated by the cost of RSA signature generation which takes 9.4 ms per join request. This is still a small cost even with such a slow machine.

VIII. 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. Nodes completing the puzzles of all nodes in the chain are provided a cryptographic proof of the vetted identity. In this way, we exploit the structure of hierarchy to distribute load and increase resilience to targeted attacks on the admission control system.

We augment the protocol with a mechanism to limit the vulnerability of a network as it grows from a small number of nodes. Additionally, we 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 method that provides such a hard bound for limiting Sybil attacks.

References

[1] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. In *Security Protocols, 8th International Workshop, Cambridge, UK, April 3-5, 2000*, pages 170–177.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *5th Symposium on Operating System Design and Implementation (OSDI 2002), December 9-11, 2002, Boston, Massachusetts, USA.*

[3] D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the Tenth USENIX Security Symposium, August 13–17, 2001, Washington, DC, USA.*

[4] J. Donath. Identity and deception in the virtual community. In *Communities in Cyberspace: Perspectives on New Forms of Social Organization. Berkeley: University of California Press, 1997.*

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

[6] C. Ellison and B. Schneier. Ten risks of pki: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[7] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of Network and Distributed Systems Symposium, San Diego, CA, Feb. 1999.*

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

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

[10] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis and defenses. In *Proceedings of the third international symposium on Information processing in sensor networks, April 26-27, 2004, Berkeley, California, USA.*

[11] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001, August 2001.*

[12] 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, 2001.*

[13] 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 Technical Report UW-CSE-01-06-02.*

[14] N. Saxena, G. Tsudik, and J. Yi. Admission control in peer-to-peer: design and performance evaluation. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks, October 31, 2003, Fairfax, Virginia.*

[15] SHA-1. Us secure hash algorithm 1. http://www.ietf.org/rfc/rfc3174.txt, 2001.

[16] A. Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th ACM SIGOPS European Workshop. Sep 2004.*

[17] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash table. In *Proceedings of the First International Workshop on Peer-to-Peer Systems 200, Cambridge, MA, March 2002.*

[18] M. Srivatsa and L. Liu. Vulnerabilities and security threats in structured overlay networks: A quantitative analysis. In *Proceedings of the 20th IEEE Annual Computer Security Applications Conference (ACSAC 2004).*

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

[20] D. Wallach. A survey of peer-to-peer security issues. In *Proceedings of International Symposium on Software Security (Tokyo, Japan), November 2002.*

[21] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley. April 2001.*