SybilGuard: Defending Against Sybil Attacks via Social Networks

Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, *Member, IEEE*, and Abraham D. Flaxman

Abstract—**Peer-to-peer and other decentralized, distributed systems are known to be particularly vulnerable to *sybil attacks*. In a sybil attack, a malicious user obtains multiple fake identities and pretends to be multiple, distinct nodes in the system. By controlling a large fraction of the nodes in the system, the malicious user is able to "out vote" the honest users in collaborative tasks such as Byzantine failure defenses. This paper presents *SybilGuard*, a novel protocol for limiting the corruptive influences of sybil attacks. Our protocol is based on the "social network" among user identities, where an edge between two identities indicates a human-established trust relationship. Malicious users can create many identities but few trust relationships. Thus, there is a disproportionately small "cut" in the graph between the sybil nodes and the honest nodes. SybilGuard exploits this property to bound the number of identities a malicious user can create. We show the effectiveness of SybilGuard both analytically and experimentally.**

Index Terms—**Social networks, sybil attack, SybilGuard, sybil identity.**

I. Introduction

As THE SCALE of a decentralized distributed system increases, the presence of malicious behavior (e.g., Byzantine failures) becomes the norm rather than the exception. Most designs against such malicious behavior rely on the assumption that a certain fraction of the nodes in the system are honest. For example, virtually all protocols for tolerating Byzantine failures assume that at least 2/3 of the nodes are honest. This makes these protocols vulnerable to *sybil attacks* [1], in which a malicious user takes on multiple identities and pretends to be multiple, distinct nodes (called *sybil nodes* or *sybil identities*) in the system. With sybil nodes comprising a large fraction (e.g., more than 1/3) of the nodes in the system, the malicious user is able to "out vote" the honest users, effectively breaking previous defenses against malicious behaviors. Thus, an effective defense against sybil attacks would remove a primary practical obstacle to collaborative tasks on peer-to-peer (p2p) and other decentralized systems. Such tasks include not only Byzantine failure defenses, but also voting schemes in file sharing, DHT routing, and identifying worm signatures or spam.

Problems With Using a Central Authority. A trusted central authority that issues and verifies credentials unique to an actual human being can control sybil attacks easily. For example, if the system requires users to register with government-issued social security numbers or driver's license numbers, then the barrier for launching a sybil attack becomes much higher. The central authority may also instead require a payment for each identity. Unfortunately, there are many scenarios where such designs are not desirable. For example, it may be difficult to select/establish a single entity that every user worldwide is willing to trust. Furthermore, the central authority can easily be a single point of failure, a single target for denial-of-service attacks, and also a bottleneck for performance, unless its functionality is itself widely distributed. Finally, requiring sensitive information or payment in order to use a system may scare away many potential users.

Challenges in Decentralized Approaches. Defending against sybil attacks without a trusted central authority is much harder. Many decentralized systems today try to combat sybil attacks by binding an identity to an IP address. However, malicious users can readily harvest (steal) IP addresses. Note that these IP addresses may have little similarity to each other, thereby thwarting attempts to filter based on simple characterizations such as common IP prefix. Spammers, for example, are known to harvest a wide variety of IP addresses to hide the source of their messages, by advertising BGP routes for unused blocks of IP addresses [2]. Beyond just IP harvesting, a malicious user can *co-opt* a large number of end-user machines, creating a *botnet* of thousands of compromised machines spread throughout the Internet. Botnets are particularly hard to defend against because nodes in botnets are indeed distributed end users' computers.

The first investigation into sybil attacks [1] proved a series of negative results, showing that they cannot be prevented unless special assumptions are made. The difficulty stems from the fact that resource-challenge approaches, such as computation puzzles, require the challenges to be posed/validated simultaneously. Moreover, the adversary can potentially have significantly more resources than a typical user. Even puzzles that require human efforts, such as CAPTCHAs [3], can be reposted on the adversary's web site to be solved by other users seeking access to the site. Furthermore, these challenges must be performed directly instead of trusting someone else's challenge results, because sybil nodes can vouch for each other. A more recent proposal [4] suggests the use of network coordinates [5] to determine whether multiple identities belong to the same user (i.e., have similar network coordinates). Despite its elegance, a malicious user controlling just a moderate number of network

Manuscript received January 31, 2007; revised October 31, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Yau. This work was supported in part by NUS Grant R-252-050-284-101 and Grant R-252-050-284-133. A preliminary version of this paper appeared in the Proceedings of the ACM SIGCOMM 2006 Conference, Pisa, Italy.

H. Yu is with the Computer Science Department, National University of Singapore, Singapore 117543 (e-mail: haifeng@comp.nus.edu.sg).

M. Kaminsky and P. B. Gibbons are with Intel Research Pittsburgh, Pittsburgh, PA 15213 USA (e-mail: michael.e.kaminsky@intel.com; phillip.b.gibbons@intel.com).

A. D. Flaxman was with Carnegie Mellon University, Pittsburgh, PA 15213 USA. He is now with Microsoft Research, Redmond, WA 98052 USA (e-mail: abie@microsoft.com).

Digital Object Identifier 10.1109/TNET.2008.923723

positions (e.g., tens in practice) can fabricate network coordinates and thus break the defense. Finally, reputation systems based on historical behaviors of nodes are not sufficient either, because the sybil nodes can behave initially, and later launch an attack. Typically, the damage from such an attack can be much larger than the initial contribution (e.g., the damage caused by throwing away another user's backup data is much larger than the contribution of storing the data). In summary, there has been only limited progress on how to defend against sybil attacks without a trusted central authority, and the problem is widely considered to be quite challenging.

This paper presents *SybilGuard*, a novel decentralized protocol that limits the corruptive influence of sybil attacks, including sybil attacks exploiting IP harvesting and even some sybil attacks launched from botnets outside the system. Our design is based on a unique insight regarding *social networks* (Fig. 1), where identities are nodes in the graph and (undirected) edges are human-established trust relations (e.g., friend relations). The edges connecting the honest region (i.e., the region containing all the honest nodes) and the sybil region (i.e., the region containing all the sybil identities created by malicious users) are called *attack edges*. Our protocol ensures that the number of attack edges is independent of the number of sybil identities, and is limited by the number of trust relation pairs between malicious users and honest users.

SybilGuard: A New Defense Against Sybil Attacks. The basic insight is that if malicious users create too many sybil identities, the graph becomes "strange" in the sense that it has a small *quotient cut*, i.e., a small set of edges (the attack edges) whose removal disconnects a large number of nodes (all the sybil identities) from the rest of the graph. On the other hand, we will show that social networks do not tend to have such cuts. Directly searching for such cuts is not practical, because we would need to obtain the global topology and verify each edge with its two endpoints. Even if we did know the global topology, the problem of finding cuts with the smallest quotient (the *Minimum Quotient Cut* problem) is known to be NP-hard.

Instead, SybilGuard relies on a special kind of verifiable random walk in the graph and intersections between such walks. These walks are designed so that the small quotient cut between the sybil region and the honest region can be used against the malicious users, to bound the number of sybil identities that they can create. We will show the effectiveness of SybilGuard both analytically and experimentally.

Section II more precisely defines our system model and the sybil attack. Section III presents the SybilGuard design. Sections IV and V provide further details, including discussing SybilGuard's guarantees and how it handles dynamic social networks. The effectiveness of SybilGuard is shown experimentally in Section VI. Finally, Section VII discusses related work and Section VIII draws conclusions.

II. Model and Problem Formulation

This section formalizes the desirable properties and functions of a defense system against sybil attacks. We begin by defining our system model. The system has n honest human beings as *honest users*, and one or more malicious human beings as *malicious users*. By definition, a user is distinct. Each honest user has a single (honest) *identity*, while each malicious user has one or more (malicious) *identities*. To unify terminology, we simply refer to all the identities created by the malicious users as *sybil identities*. Identities are also called *nodes*, and we will use "identity" and "node" interchangeably. Honest users obey the defense system protocol. All malicious users may collude, and we say that they are all under the control of an *adversary*. The adversary may eavesdrop on any message sent between users over the computer network (Internet).

Nodes participate in the system to receive and provide service (e.g., file backup service) as peers. Because a node in the system may be honest or sybil, a defense system against sybil attacks aims to provide a mechanism for any node V (called a *verifier*) to decide whether or not to *accept* or *reject* another node S (called the *suspect*). Accepting S means that V is willing to receive service from and provide service to S.

Desirable Guarantees. Ideally, the defense system should guarantee that V accepts only honest nodes. But because such an idealized guarantee is challenging to achieve, we aim at bounding the number of sybil nodes that are accepted. This weaker guarantee is still sufficiently strong to be useful in most application scenarios for the following reason. The application already needs to tolerate malicious users even without sybil attacks. A sybil attack simply enables the malicious users to create an unlimited number of sybil nodes to exceed the "tolerance" threshold of the application's defense system (e.g., 1/3 in byzantine consensus), regardless of how high the "tolerance" threshold is. Thus bounding the number of sybil nodes will prevent the adversary from doing so, and then the application can rely on existing techniques to effectively tolerate the malicious users.

As a concrete example, let us consider maintaining replicas of file blocks on a DHT-based storage system. DHT-based systems (such as those based on Chord [8]) place replicas on a random set of nodes in the system, without knowledge of which nodes are honest and which are sybil. Our goal here is to ensure that a *majority* of the replicas are placed on honest nodes, so that we can use majority voting to retrieve the correct file block. If the number of accepted sybil nodes is smaller than the number of honest nodes n, then from Chernoff bounds [9], the probability of having a majority of the replicas on honest nodes approaches 1.0 exponentially fast with the number of replicas.

Summary of SybilGuard Guarantees. SybilGuard is completely decentralized and all functions are with respect to a given node. SybilGuard guarantees that with high probability, an honest node accepts at most $g \cdot w$ sybil nodes, where g is the number of attack edges in the system and w is the length of the protocol's random walks. Conceptually, in SybilGuard, there is an equivalence relation that partitions all accepted nodes into equivalence classes (called *equivalence groups*). Nodes that are rejected do not belong to any equivalence groups. An equivalence group that includes one or more sybil nodes is called a *sybil group*. SybilGuard achieves its $g \cdot w$ guarantee by (i) bounding the number of *sybil groups* within g, and (ii) bounding the size of each *sybil group* within w. SybilGuard bounds the number and the size of sybil groups without necessarily knowing which groups are sybil. Also, the concept of sybil groups does not need to be visible to the application.

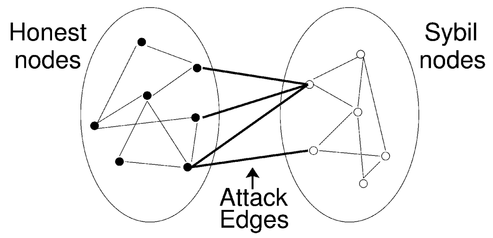

Fig. 1. The social network with honest nodes and sybil nodes. Note that regardless of which nodes in the social network are sybil nodes, we can always "pull" these nodes to the right side to form the logical network in the figure.

As a side effect of bounding the number and size of sybil groups, SybilGuard may (mistakenly) reject some honest nodes. SybilGuard guarantees that an honest node accepts, and also is accepted by, most other honest nodes (except a few percent in our later simulation) with high probability. Thus, an honest node can successfully obtain service from, and provide service to, most other honest nodes. Notice that because SybilGuard is decentralized, the set of accepted nodes by node V_1 can be different from those accepted by node V_2. However, the difference should be small since both V_1 and V_2 should accept most honest nodes with high probability.

III. SybilGuard Design

In this section, we present our SybilGuard design. We will assume a static social network where all nodes are online; we will discuss user and node dynamics in Section V.

A. Social Network and Attack Edges

SybilGuard leverages the existing human-established trust relationships among users to bound both the number and size of sybil groups. All honest nodes and sybil nodes in the system form a *social network* (see Fig. 1). An undirected edge exists between two nodes if the two corresponding users have strong social connections (e.g., colleagues or relatives) and trust each other not to launch a sybil attack. If two nodes are connected by an edge, we say the two users are *friends*. Notice that here the edge indicates strong trust, and the notion of friends is quite different from friends in other systems such as online chat rooms. An edge may exist between a sybil node and an honest node if a malicious user (Malory) successfully fools an honest user (Alice) into trusting her. Such an edge is called an *attack edge* and we use g to denote the total number of attack edges. The authentication mechanism in SybilGuard ensures that regardless of the number of sybil nodes Malory creates, Alice will share an edge with at most one of them (as in the real social network). Thus, the number of attack edges is limited by the number of trust relation pairs that the adversary can establish between honest users and malicious users. While the adversary has only limited influence over the social network, we do assume it may have full knowledge of the social network.

The degree of the nodes in the social network tends to be much smaller than n, so the system would be of little practical use if nodes only accepted their friends. Instead, SybilGuard bootstraps from the given social network a protocol that enables honest nodes to accept a large fraction of the other honest nodes.

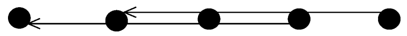

Fig. 2. Two routes of length 3. Sharing an edge necessarily means that one route starts after the other.

It is important to note that SybilGuard does not increase or decrease the number of edges in the social network as a result of its execution.

B. Random Routes

SybilGuard uses a special kind of random walks, called *random routes*, in the social network. In a standard random walk, at each hop, the current node flips a coin on the fly and selects a (uniformly) random edge to direct the walk. In random routes, each node uses a pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges. Specifically, each node uses a randomized routing table to choose the next hop. A node A with d neighbors uniformly randomly chooses a permutation "$x_1, x_2, \ldots, x_d$" among all permutations of $1, 2, \ldots, d$. If a random route comes from the ith edge, A uses edge x_i as the next hop. It is possible that $i = x_i$ for some i. The routing table of A, once chosen, will never change (unless A's degree changes—see Section V).

For random routes in the honest region, these routing tables give us the following properties. First, once two routes traverse the same edge along the same direction, they will merge and stay merged (called the *convergence property*). Furthermore, an outgoing edge uniquely determines an incoming edge as well; thus the random routes can be back-traced (called the *back-traceable property*). In other words, it is impossible for two routes to enter the same node along different edges but exit along the same direction.

With these two properties, if we know that a random route of a certain length w traverses a certain edge e along a certain direction in its ith hop, the entire route is uniquely determined. In other words, there can be only one route with length w that traverses e along the given direction at its ith hop. In addition, if two random routes ever share an edge in the same direction, then one of them must start in the middle of the other (Fig. 2).

Of course, these properties can be guaranteed only for the portions of a route that do not contain sybil nodes. Sybil nodes may deviate from any aspect of the protocol.

C. Route Intersection as the Basis for Acceptance

In SybilGuard, a node with degree d performs d random routes (starting from itself) of a certain length w (e.g., w is roughly 2000 for the one-million node network in our later experiments), one along each of its edges. These random routes form the basis of SybilGuard whereby an honest node (the *verifier* V) decides whether or not to accept another node (the *suspect* S). In particular, a verifier route *accepts* S if and only if at least one route from S intersects that route from V (see Fig. 3). V *accepts* S if and only if at least a threshold t of V's routes accept S.

Because of the limited number of attack edges, if one chooses w appropriately, most of the verifier's routes will remain entirely within the honest region with high probability. To intersect with a verifier's random route that remains entirely within the

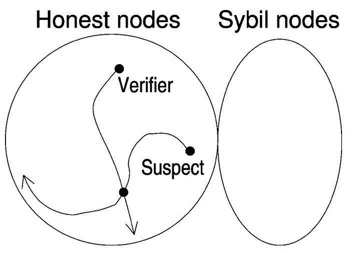

Fig. 3. The verifier's random route accepts the suspect because the random routes intersect. SybilGuard leverages the facts that 1) the average honest node's random route is highly likely to stay within the honest region and 2) two random routes from honest nodes are highly likely to intersect within w steps.

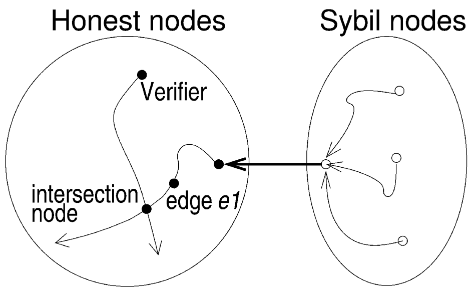

Fig. 4. All random routes traversing the same edge merge.

honest region, a sybil node's random route must traverse one of the attack edges (whether or not the sybil nodes follow the protocol). Suppose there were only a single attack edge (as in Fig. 4). Based on the convergence property, the random routes from sybil nodes must merge completely once they traverse the attack edge. Thus, all of these routes that intersect the verifier's route will have the same intersection node; furthermore, they enter the intersection node along the same edge (edge $e1$ in the figure). The verifier thus considers all of these nodes to be in the same equivalence group, and hence there is only a single sybil group. In the more general case of g attack edges, the number of sybil groups is bounded by g.

SybilGuard further bounds the size of the equivalence groups (and hence of the sybil groups) to be at most w, the length of the random routes. From the back-traceable property, we know there can be at most w distinct routes that i) intersect with the verifier's random route at a given node, and ii) enter the intersection node along a given edge (e.g., along edge $e1$ in Fig. 4). Specifically, there is one such route that traverses the given edge in its ith hop, for $i = 1, \ldots, w$. Thus, the verifier accepts exactly one node for each of the w hop numbers at a given intersection point and a given edge adjacent to the intersection point. In summary, there are many equivalence groups, but only g are sybil and each has at most w nodes.

For honest nodes, we will show that with appropriate w, (i) an honest node's random route intersects with the verifier's route with high probability, and (ii) such an honest node will never compete for the same hop number with any other node (including sybil nodes). Thus, the average honest node will be accepted with high probability.

Our SybilGuard design leverages the following three important facts to bound the number of sybil nodes: (i) social networks tend to be *fast mixing* (defined in Section IV), which necessarily means that subsets of honest nodes have good connectivity to the rest of the social network, (ii) too many sybil nodes (compared to the number of attack edges) disrupts the fast mixing property, and (iii) the verifier is itself an honest node, which breaks symmetry. We will elaborate on these aspects later.

D. Secure and Decentralized Design for Random Routes and Their Verification

The previous sections explained the basics of random routes. In the actual SybilGuard protocol, these routes are performed in a completely decentralized way. The two local data structures (registry tables and witness tables) described in this section are the only data structures that each node needs to maintain. Also, propagating these tables to direct neighbors is the only action each node needs to take in order to determine random routes.

Edge keys. Each pair of friends in the social network shares a unique symmetric secret key (e.g., a shared password) called the *edge key*. The edge key is used to authenticate messages between the two friends (e.g., with a Message Authentication Code). Because only the two friends need to know the edge key, key distribution is easily done out-of-band (e.g., via phone calls). Because of the nature of the social network and the strong trust associated with the notion of friends in SybilGuard, we expect node degrees to be relatively small and will tend not to increase significantly as n grows. As a result, a user only needs to invoke out-of-band communication a small number of times. In order to prevent the adversary from increasing the number of attack edges (g) dramatically by compromising high-degree honest nodes, each honest node (before compromised) voluntarily constrains its degree within some constant (e.g., 30). Doing so will not affect the guarantees of SybilGuard as long as the social network remains fast mixing. On the other hand, researchers have shown that even with rather small constant node degrees, social networks (or more precisely, small-world topologies) are fast mixing [10], [11].

A node informs its friends of its IP address whenever its IP address changes, to allow continued communication via the Internet. This IP address is used only as a hint. It does not result in a vulnerability even if the IP address is wrong, because authentication based on the edge key will always be performed. If DNS and DNS names are available, nodes may also provide DNS names and only update the DNS record when the IP address changes.

Registration. In SybilGuard, each node S with degree d must perform d random routes of w hops each and remember these routes. To prevent S from "lying" about its routes, SybilGuard requires S to *register* with all w nodes along each of its routes. A node Q along the route permits S to register only if S is one of the nodes that are within w hops "upstream" (details below). When the verifier V wants to verify S, V will ask the intersection point (between S's route and V's route) whether S is indeed registered.

In this registration process, each node needs to use a "token" that cannot be easily forged by other nodes. Note that the availability of such tokens does not solve the sybil attack problem by itself, because a malicious user may have many such tokens. A node will be accepted based on its token. The token must be unforgeable to prevent the adversary from stealing the token

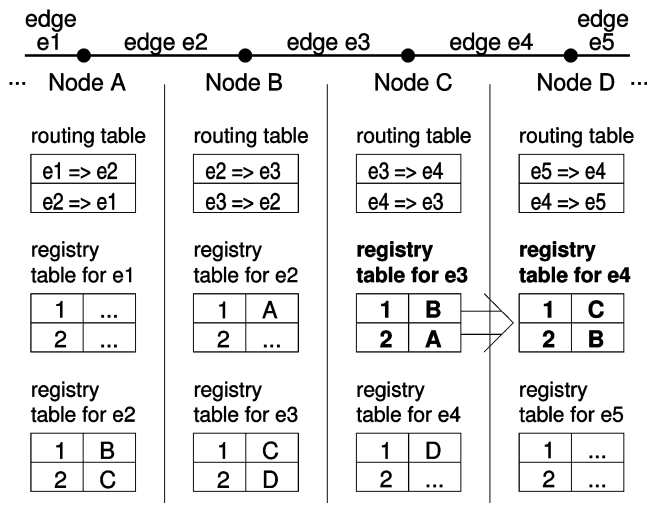

Fig. 5. Maintaining the registry tables. In order to simplify this example, $w = 2$, each node has exactly two edges, and the routing tables are carefully chosen. The node names in the registry tables stand for the nodes' public keys.

of an honest node (unless the node is compromised). If users have static or slowly changing IP addresses, and there is no IP spoofing, then a node's IP address could be used as its token.

To address a more general scenario, including frequently changing IP addresses and IP spoofing, we can instead use public key cryptography for the tokens. Each honest node has a locally generated public/private key pair. Notice that these public and private keys have no connection with the edge keys (which are secret symmetric keys). Malicious nodes may create as many public/private key pairs as they wish. We use the private key of each node as the unforgeable token, while the public key is registered along the random routes as a proof of owning the token. Note that we do not need to solve the public key distribution problem, because we are not concerned with associating public keys to, for example, human beings or computers. The only property SybilGuard relies on is that the private key is unforgeable and its possession can be verified. To perform the registration in a secure and completely decentralized manner, SybilGuard uses registry tables and witness tables, as described next.

Registry tables. Each node I maintains a *registry table* for each of its edges (Fig. 5). The ith entry in the registry table for edge e lists the public key of the node whose random route enters I along e at its ith hop. For example, consider the registry table on C for edge e_3 in Fig. 5. Here, one of B's random routes is $B \rightarrow$ (via edge e_3) $C \rightarrow$ (via edge e_4) D. In other words, in the first hop of this random route, B enters C via edge e_3. Thus the first entry in the registry table is B's public key. Similarly, the second entry is A's public key. As a result, the registry table has w entries that are the public keys of the w "upstream" nodes along the direction of edge e_3 from C.

Suppose that according to C's routing table, e_4 is the outgoing direction corresponding to e_3 (as in Fig. 5). C will forward its registry table for e_3 to its neighbor D along e_4, via an authenticated channel established using the edge key for e_4. D then populates its registry table for e_4 by shifting the registry table from C downward by one entry and adding C's public key as the new first entry.

As shown in Fig. 5, this simple design will ultimately register each node's public key with all nodes on its d random routes. The protocol does not have to proceed in synchronous rounds, and nodes in the system may start with empty registry tables. The overhead of the protocol is small as well. Even with one million nodes, if we were to use $w = 2000$ (already pessimistic given our simulation results), then a registry table is roughly 256 KB when using 1024-bit public keys. For a node with ten neighbors, the total data sent is 2.56 MB. A further optimization is to store cryptographically secure hashes of the public keys in the registry table instead of the actual public keys. With each hashed key being 160-bit, the total data sent by each node would be roughly 400 KB. Finally, it is important to notice that registry table updates are needed only when social trust relationships change (Section V). Thus, we expect the bandwidth consumption to be quite acceptable.

Witness tables. Registry tables ensure that each node registers with the nodes on its random routes. Each node, on the other hand, also needs to know the set of nodes that are on its random routes. This is achieved by each node maintaining a *witness table* for each of its edges. The ith entry in the table contains the public key (or its hash, if we use the above optimization) and the IP address of the node encountered at the ith hop of the random route along the edge. The public key will later be used for intersection and authentication purposes, while the IP address will be used as a hint to find the node. If the IP address is stale or wrong, it will have the same effect as the intersection node being offline. (Offline nodes are addressed in Section V-A.)

The witness table is propagated and updated in a similar fashion as the registry table, except that it propagates "backward" (using the reverse of the routing table). In this way, a node will know the w "downstream" nodes along the direction of each of its edges, which is exactly the set of nodes that are on its random routes. Different from registry tables, witness tables should be updated when a node's IP address changes (even with a static social network). But this updating can be done lazily, given the optimizations described below in the verification process.

Verification process. Fig. 6 depicts the process for a node V to verify a node S. V needs to perform an intersection between each of its random routes and all of S's random routes. To do this, S sends all of its witness tables to V, together with S's public key. The communication overhead in this step can be reduced using standard optimizations such as Bloom filters [9] to summarize the nodes in witness tables.

For each of V's witness tables, V performs an intersection with all of S's tables, and determines the (hashed) public key of the first intersection point X (if any) on V's route. V then contacts X using the recorded IP address in the witness table as a hint. V authenticates X by requiring X to sign each message sent, using its private key. If hashed keys are used, X also sends its public key, which V hashes and compares with the stored hash, before authenticating X. If X cannot be found using the recorded IP address, V will try to obtain X's IP address from nearby nodes in the witness table. They will likely have X's more up-to-date IP address because they are near X. Because V will always authenticate X based on X's public key, this does not introduce a vulnerability.

```
1. $S$ sends its witness tables and public key to $V$
2. For each of $V$'s $d$ witness tables, $T_V$, do
     // Find the first intersection point for this route, if any
3.   Find the first public key, $X$, in $T_V$ that occurs in
       at least one of $S$'s witness tables
4.   If no such $X$ exists, the route rejects $S$ and we skip
       to the next loop iteration
     // Verify the intersection point
5.   $V$ contacts $X$ using the IP address in $T_V$, and
       authenticates $X$ by requiring $X$ to sign each
       message sent
6.   If $X$ cannot be found using the IP address in $T_V$,
       $V$ tries to obtain $X$'s IP address from nearby
       nodes in $T_V$ and then repeats step 5
     // Verify that a route from $S$ goes through $X$
7.   $V$ checks with $X$ whether $S$'s public key is
       present in one of $X$'s registry tables
8.   If it is present, then the route accepts $S$,
       otherwise the route rejects $S$
   // Done with $V$'s witness tables, determine outcome
9. If at least $d/2$ of $V$'s routes accept $S$, $V$ accepts $S$
```

Fig. 6. Protocol for a node V of degree d to verify a node S.

V then checks with X whether S's public key is indeed present in one of X's registry tables. The entry number is not relevant. If it is present, then that route from V accepts S. If at least a threshold t of V's routes accept S, V accepts S (i.e., S's public key). Although we have set $t = d/2$ in the figure, the asymptotic guarantees of the protocol do not depend on this particular choice for t. Finally, when interacting with S, V always authenticates S by requiring S to sign every message sent, using its private key.

In all, only a constant number of messages are required for one node to verify another.

Key revocation. A node can easily revoke its old public/private key pair by unilaterally switching to a new public/private key pair, and then using the new public key in its registry table and witness table propagation. The old public key in registry and witness tables will be overwritten by the new public key. As for edge keys, a node can revoke an edge key unilaterally simply by discontinuing use of the key and discarding it.

Sybil nodes. We described the protocol for the case where all nodes behave honestly. A sybil node may not follow the protocol and may arbitrarily manipulate the registry tables and witness tables. SybilGuard is still secure against such attacks. To understand why and obtain intuition, it helps to consider the set of all registry table entries on all honest nodes in the system. For simplicity, assume that all honest nodes have the same degree d. Thus, altogether the honest nodes contain $n \cdot d \cdot w$ registry table entries.

Consider a malicious node M and a single attack edge connecting an honest node A with M. Clearly, M can propagate to A an arbitrary registry table, thus polluting the w entries in A's registry table. Suppose A next forwards the registry table to B, who shifts the table downward and adds A as the first entry. Thus $w - 1$ entries in B's registry table are polluted. Continuing this argument, we see that a single attack edge enables M to control $w + (w-1) + \cdots + 1 \approx w^2/2$ entries system-wide. With g attack edges and even when $g \cdot w$ approaches n, the total number of polluted entries $(g \cdot w^2/2)$ is still a factor of $2d$ smaller than the total number of entries $(n \cdot d \cdot w)$. This provides some intuition why the number of accepted sybil nodes is properly bounded even though the adversary may not follow the SybilGuard protocol.

IV. SybilGuard Guarantees

A. *Limiting the Number of Attack Edges*

The effectiveness of SybilGuard relies on there being a limited number of attack edges (g). There are several ways the adversary might attempt to increase g:

- The malicious users establish social trust and convince more honest users in the system to "be their friends" in real life. But this is quite difficult to do on a large scale.
- A malicious user (Malory) who managed to convince an honest user (Alice) to be her friend creates many sybil nodes, and then Malory forwards to these sybil nodes her edge key with Alice. Notice, however, that Alice only has a single edge key corresponding to the edge between Alice and Malory. As a result, all messages authenticated using that edge key will be considered by Alice to come from the same edge. Thus the number of attack edges remains unchanged.
- The adversary compromises a single honest node with degree d, making it a sybil node. Because d was already constrained (before the node is compromised) within some constant by the user, g can be increased by at most some constant. On the other hand, the adversary will not be able to create further attack edges from the node because adding an edge to another honest user requires out-of-band verification by that user. When a user drops and then makes new friends, it is possible for the adversary with access to the old edge keys to "resurrect" dropped edges and hence further increase g. However, we expect such effect to be negligible in practice and if necessary, can be prevented by requiring out-of-band confirmation when deleting edges.
- The adversary compromises a small fraction of the nodes in the system. This will not likely increase g excessively due to the reasons above.
- The adversary compromises a large fraction of the nodes in the system. Here the system has already been subverted, and the adversary does not even need to launch a sybil attack. SybilGuard will not help here.
- The adversary compromises a large number of computers (i.e., creates a botnet), only some of which belong to the system. The increase in g is upper bounded by some constant times the number of compromised computers which already belong to the system. The increase is not affected by the total size of the botnet. Although acquiring a botnet with many nodes may be relatively easy (e.g., in the black market), acquiring a botnet containing many nodes that are already in the system is more challenging.

In summary, SybilGuard is quite effective in limiting the number of attack edges, as long as not too many honest users are compromised. Relatively speaking, SybilGuard is more effective defending against malicious users than defending against compromised honest users that belong to the system. This is because a malicious user must make real friends in order to increase the number of attack edges, while compromised honest users already have friends.

B. Designing the Length of Random Routes in Order to Achieve SybilGuard's Guarantees

A critical design choice in SybilGuard is w, the length of the random routes. The value of w must be sufficiently small to ensure that (i) a verifier's random route remains entirely within the honest region with high probability, and (ii) the size of sybil groups is not excessively large. On the other hand, w must be sufficiently large to ensure that routes in the honest region will intersect with high probability.

In the following, we provide some analytical assurance that having $w = \Theta(\sqrt{n}\log n)$ will likely satisfy the above requirements simultaneously. Our results are for random walks instead of the random routes used in SybilGuard—considering random walks allows us to leverage the well-established theory on such walks. At the end of this section, we will explain how these results likely apply to random routes, which will be further confirmed in our later experiments.

Guarantees on honest nodes. The first property we would like to show is that $w = \Theta(\sqrt{n}\log n)$ is likely to be sufficiently large for routes from an honest verifier and an honest suspect to intersect with high probability. Such a property for random walks has been proven [12], [13] in several other contexts, and thus we give only a high-level review. First, we need to provide some informal background. With a length-w random walk, clearly the distribution of the ending point of the walk depends on the starting point. However, for connected and non-bipartite graphs, the ending point distribution becomes independent of the starting point when $w \to \infty$. This distribution is called the *stationary distribution* of the graph. The *mixing time* T of a graph quantifies how fast the ending point of a random walk approaches the stationary distribution. In other words, after $\Theta(T)$ steps, the node on the random walk becomes roughly independent of the starting point. If $T = \Theta(\log n)$, the graph is called *fast mixing*.

Many randomly-grown topologies are fast mixing, including social networks (or more specifically, small-world topologies) [10], [11]. Thus, a walk of $\Theta(\sqrt{n}\log n)$ steps contains $\Theta(\sqrt{n})$ independent samples drawn roughly from the stationary distribution. When a verifier's walk and a suspect's walk remain in the honest region (which we show below occurs with high probability), both walks draw $\Theta(\sqrt{n})$ independent samples from roughly the same distribution. It follows from the generalized Birthday Paradox [12], [13] that they intersect with probability $1-o(1)$. Because this claim holds for each of the verifier's walks and each of the suspect's walks, an honest verifier accepts an honest suspect with high probability.

Guarantees on the number of Sybil nodes accepted. Recall from Section III-C that for a verifier V's route entirely in the honest region, SybilGuard limits the number of sybil groups to g and the size of each sybil group to w, for a total of $g \cdot w$ sybil nodes accepted. On the other hand, a verifier V's route that enters the sybil region falls under the control of the adversary, and V may not be able to bound the number of sybil nodes intersecting that route. The following theorem bounds the probability that a random walk starting from a random honest node enters the sybil region, showing that such problematic routes are rare (given an upper bound on g and our choice for w).

Theorem 1: For any connected social network, the probability that a length-w random walk starting from a uniformly random honest node will ever traverse any of the g attack edges is upper bounded by $g \cdot w/n$. In particular, when $w = \Theta(\sqrt{n}\log n)$ and $g = o(\sqrt{n}/\log n)$, this probability is $o(1)$.

We leave the proof to our full technical report [14]. The actual likelihood, as shown in our later experiments, is much better than the above pessimistic theoretical bound of $g \cdot w/n$. We should point out that the above theorem provides only an "average" guarantee for all honest nodes. Honest nodes that are closer to attack edges are likely to have a larger probability of walking into the sybil region. Recall, however, that V performs a random route starting from each of its edges and accepts a suspect S only if at least a threshold t of these routes accept S. This serves to mask the misleading effects of routes extending into the sybil region. The parameter t involves the following trade-off: if t is too small, then V may have a large probability of having more than t routes entering the sybil region; if t is too large, then V may have trouble accepting other honest nodes if more than $(d-t)$ routes from V enter the sybil region and if the sybil nodes prevent intersection from happening (d is the degree of V). In other words, to avoid both of the above two problematic scenarios, the number of routes entering the sybil region must be smaller than $\min(t, d-t)$. Thus, obviously, setting t to $d/2$ will maximize the probability of avoiding the two problematic scenarios, and our approach effectively becomes majority voting. Our later simulation results show that using majority voting gives most nodes a high probability of success. Thus, an honest node accepts at most $g \cdot w$ sybil nodes with high probability.

Random Routes Versus Random Walks. SybilGuard uses random routes, while the above derivations are for random walks. If a random route enters a node A for the first time, then the next hop is indeed uniformly randomly chosen from all of A's neighbors, which is exactly the same as in random walks. In some sense, we can imagine that A simply pre-flipped all the coins it needed to flip. On the other hand, a random route differs from a random walk when the random route intersects with itself.

Consider a random route that previously entered node A via edge i and was directed to edge x_i. Imagine that now the route enters A for a second time via edge j. We consider the following two cases and explain the behavior of random routes, as compared to random walks.

If $j = i$, then we have a *repeated edge loop* and the random route will traverse this loop repeatedly, which clearly deviates significantly from the behavior of a random walk. We now provide an intuition as to why such loops are rare. Notice that the first edge in the route must be the first edge that is traversed twice. In other words, repeated edge loops can only form at the starting node (Fig. 7). If a loop is formed, the random route must have come back to the starting point, and the starting point must have decided to forward the route along the first edge. Also notice that the smallest loop has three hops, otherwise it is impossible for the route to traverse the same edge (in the same di-

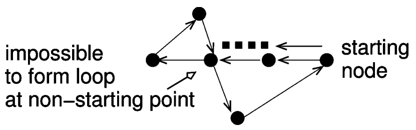

Fig. 7. A loop can form only at the starting point of a route.

rection) twice. More concretely, consider a simplified scenario where all nodes have the same degree d. At the second hop, the route will return to the starting point with probability $1/d$. At the third hop, if a loop is formed, the starting point must have decided to forward the route along the same edge as the first hop. Thus, a repeated edge loop is formed at the third hop with probability $1/d^2$. As the route proceeds, the chance of repeating the first hop edge at the given hop will usually become smaller and smaller. In fact, in a fast mixing graph, after a small number of hops a random walk is equally likely to be traversing any edge in a given hop. This provides an intuition as to why loops are unlikely. Moreover, routes with loops can still be used, because they do not compromise security—they simply have fewer than w distinct nodes and hence are less likely to intersect with other routes.

If $j \neq i$, then the random route will not have formed a loop and A will pick x_j ($x_j \neq x_i$) as the next hop. Since the routing table is a permutation, x_j will be a uniformly random edge except that it cannot be x_i. In other words, A has already eliminated x_i as a choice for the next hop. This introduces some small correlation between A's next hop choice for the second time and for the first time. Thus strictly speaking, a random route is different from a random walk unless the random route does not intersect itself. Intuitively, however, such correlation is small, because only x_i is eliminated (out of A's edges) as a choice for x_j, and also because a random route does not tend to encounter the same node many times.

C. Locally Determining the Appropriate Length of Random Routes

Because SybilGuard is decentralized, each node needs to locally determine w. Directly setting $w = \Theta(\sqrt{n}\log n)$ requires the knowledge of n. This is challenging because we must exclude sybil nodes when estimating n, which requires running SybilGuard with an appropriate w.

Instead, to locally determine w, a node A first performs a short random walk (e.g., 10 hops), ending at some node B. Because the random walk is short, with high probability, it stays in the honest region and B is an honest node. Next A and B conceptually both perform random routes to determine how long the two routes need to be to intersect. In practice, A and B should have already performed random routes along all directions, thus B simply needs to hand over one of its witness tables to A. It is important here to use a standard random walk (instead of a random route) to choose B, otherwise A's random route will always intersect with B within a small number of hops. Also, our later simulation will show that even a walk as short as 3 hops suffices to obtain good estimates of w in a million-node social network.

The intuition behind the above design is that in fast mixing graphs, a random walk of short length is sufficient to approach the stationary distribution. Thus, B is just a random node drawn from the stationary distribution, and the procedure yields a random sampling of w. The sampling, however, is biased because the stationary distribution is not necessarily a uniform distribution and B is more likely to be a higher-degree node than a lower-degree node. On the other hand, notice that if we start a random walk from a uniformly random node C, then after $\Theta(T)$ hops (T being the mixing time), the walk will be at a node roughly drawn from the stationary distribution. Thus, the needed route length for two routes (starting from A and C, respectively) to intersect is at most $\Theta(T) + w$. Because $w = \Theta(\sqrt{n}\log n)$ and $T = \Theta(\log n)$, we can safely ignore the term of $\Theta(T)$, which will be further confirmed in our later experiments.

Finally, node A obtains multiple such samples using the above procedure, and calculates the median m of the samples (see Section VI for the number of samples needed). It then sets $w = 2.1\,m$, where the constant 2.1 is derived from our analysis of Birthday Paradox distributions [14]. The analysis proves that multiplying the median by 2.1 is sufficient to ensure a collision probability of 95%, regardless of n. Note that when B is itself a sybil node or the random route from either A or B enters the sybil region, the adversary controls that particular sample. Thus, using the median sample to estimate w is much more robust than directly using the 95th percentile.

V. SybilGuard Under Dynamics

Our protocol so far assumes that the social network is static. In decentralized distributed systems, a typical user first downloads and installs the software (i.e., the user is *created*). The node corresponding to the user may then freely *join* or *leave* the system (i.e., become *online* and *offline*) many times. Finally, the user may decide to uninstall the software and never use it again (i.e., the user is *deleted*). Node join/leave tends to be much more frequent than user creation/deletion. For example, dealing with frequent node join/leave (or "churn") is often a critical problem faced by DHTs.

SybilGuard is designed such that it needs to respond only to user creation/deletion, and *not* to node churn (i.e., not to nodes going offline and coming online in possibly unpredictable ways). The social network definition in this paper always includes all users/nodes that have been created and not yet deleted, regardless of whether they are currently online or offline.

A. Dealing With Offline Nodes

In SybilGuard, a node communicates with other nodes only when (i) it tries to verify another node, and hence needs to contact the intersection nodes of the random routes, and (ii) it propagates its registry and witness tables to its neighbors.

For the first scenario, because both the verifier V and the suspect S perform multiple random routes, there will likely be multiple intersections. In fact, even a single route from V and a single route from S may have multiple intersections. The verification can be done as long as a majority of V's routes have at least one intersection point online.

For propagating registry and witness tables, note that this occurs when a random route changes, due to user creation/deletion or edge creation/deletion in the social network. Witness table

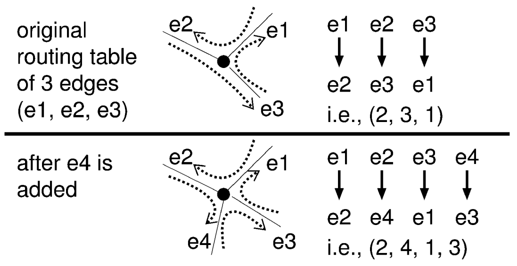

Fig. 8. Incremental maintenance of routing tables. The example assumes that $d = 3$ and $k = 2$. Note that after edge $e4$ is added, only routes entering via edge $e2$ need to be redirected.

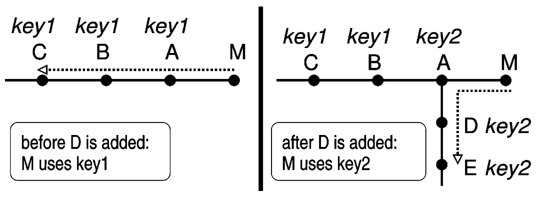

Fig. 9. A potential attack by M during node dynamics.

propagation may also be needed when IP addresses change, but such updating can be performed lazily. Previous studies [15] on p2p systems show that despite high node churn rate, user creation/deletion occurs only infrequently and the average user lifetime is roughly a year. Similarly, people make and lose social trust relations in real life over months-long time horizons. Thus, the system can afford to take days to completely propagate a new registry or witness table, waiting for nodes to come online. In the case of a new user, prior to becoming a full participant, she can always use the system via a friend as a proxy. As an optimization, SybilGuard also has a mechanism that allows a node to bypass offline nodes when propagating registry and witness tables. We leave the details of such mechanism to [14].

In the process of propagating/updating registry and witness tables, the social network may change again. Thus, it is helpful to consider it as a decentralized, background stabilization process. This means that if the topology were to stop changing, then the registry and witness tables would eventually stabilize to a consistent state for this (now static) topology.

B. Incremental Routing Table Maintenance

When users and edges are added or deleted in the social network, the routing tables must be updated as well. Adding a new node can be considered as first adding a node with no edges and then successively adding its edges one by one. Deleting a node can be considered similarly. Thus, we need to discuss only edge creation and deletion.

We first explain how A updates its routing table when a new edge is added between A and B. Suppose A's original degree is d and its original routing table is the permutation "$x_1, x_2, \ldots, x_d$". A trivial way to update A's routing table would be to pick a new random permutation of "$1, 2, \ldots, d, d+1$" that is unrelated to "$x_1, x_2, \ldots, x_d$". Doing so, however, would affect/redirect many routes, and incur unnecessary overhead in updating registry and witness tables.

Instead, SybilGuard uses an incremental maintenance algorithm where only routes entering A along a specific edge may be affected (Fig. 8). This reduces the expected overhead on the network by a factor of almost d. In this algorithm, when a new edge is added to A, A chooses a uniformly random integer k between 1 and $d+1$, inclusive. If $k = d+1$, then A's new routing table will be "$x_1, x_2, \ldots, x_d, d+1$". If $1 \le k \le d$, A's new routing table will be "$x_1, x_2, \ldots, x_{k-1}, d+1, x_{k+1}, \ldots, x_d, x_k$". In other words, we replace x_k with $d+1$, and then append x_k to the end of the permutation. Similarly, for edge deletion, suppose A's original routing table is "$x_1, x_2, \ldots, x_d, x_{d+1}$". Without loss of generality, assume that we are deleting edge $d+1$, and let k be such that $x_k = d+1$. If $k = d+1$, then A's new routing table is trivially "$x_1, x_2, \ldots, x_d$". Otherwise the new routing table will be "$x_1, x_2, \ldots, x_{k-1}, x_{d+1}, x_{k+1}, \ldots, x_d$". In other words, we simply substitute x_k with x_{d+1}. For both insertion and deletion, only routes entering A via edge k are affected, and one can prove [14] that the resulting routing table is indeed a uniformly random permutation.

C. Attacks Exploiting Node Dynamics

This section shows that having a node perform a random route along each of its edges is necessary for security and provides a defense against potential attacks under node dynamics. We first explain the potential attack scenario. Suppose each node were to perform only a single random route, and consider the example in Fig. 9, where $w = 3$. Here M is malicious and the other nodes are honest. M's random route is $M \rightarrow A \rightarrow B \rightarrow C$. Thus, A, B, and C record M's public key *key1* in their registry tables. Now another honest node D joins, and establishes edges with A and E. A updates its routing table, and suppose that routes from M now go to D instead of B. Being malicious, M launches the attack by changing its public key to *key2*. Now A, D, and E will record *key2* in their registry tables. At this point, *key1* is registered on $w-1$ nodes, while *key2* is registered on w nodes. Both of them are likely to be successfully verified with good probability.

The source of the above vulnerability is that when the routing table on A changes, the system needs to "revoke" the stale entry of *key1* from the registry tables on B and C, because M's random route no longer passes through these nodes. Explicitly revoking stale entries would introduce considerable complexity because B and C may be offline. An alternative design would be to associate TTLs with table entries, which unavoidably introduces a trade-off between security and overheads to refresh expired entries.

SybilGuard prevents the above attack by having all nodes perform random routes along all directions. In particular, if D (with *key3*) has a random route of $D \rightarrow A \rightarrow B \rightarrow C$, then *key3* will overwrite M's *key1*. It is also possible that D's route may not be $D \rightarrow A \rightarrow B \rightarrow C$. However, it is easy to show that the stale entries will always be overwritten by some node. To understand why, suppose that an entry in B's registry table indicates that B is the ith hop in the random route of M. If this entry is stale, it means that B is no longer the ith hop in M's route. From the back-traceable property of random routes, we can always backtrace from B for i hops and reach some node F. One of F's routes must visit B at the ith hop and thus F's public key will overwrite the stale entry on B. In other words, the back-traceable property ensures that for any registry table entry, there is one and exactly one "owner" of the entry. Under node dynamics, ownership may change and there may be temporary periods where a malicious user "owns" more entries than it should. However, after the system stabilizes, all entries will be "owned" by the right owner. Based on such observations, we can easily see that other similar attacks under node dynamics will be prevented by SybilGuard as well.

VI. Evaluation

This section uses simulation to evaluate the guarantees of SybilGuard. We choose to use simulation because it enables us to study large-scale systems. Because social networks tend to contain private information, there are only a limited number of publicly available social network datasets. Those that are publicly available [16], [17] are quite small, which prevents a thorough evaluation of probabilistic guarantees. Thus we use the widely accepted Kleinberg's synthetic social network model [18] in our evaluation, which generalizes from the Watts–Strogatz model [19]. We use the model to instantiate three different graphs: a million-node graph with average node degree of 24, a ten-thousand-node graph with average degree of 24, and a hundred-node graph with average degree of 12.

A. Model for Social Network

Kleinberg's social network model [18] successfully explains the principle of "six degrees of separation" in social networks. The model uses a two-dimensional grid as the base structure. The *grid distance* between two nodes is defined to be the minimum number of hops needed to go from one node along the grid edges to the other. The small-world topology constructed contains all nodes in the two-dimensional grid. The grid edges may or may not be in the small-world topology depending on the parameters.

To construct the small-world topology, each node A in the topology establishes (undirected) edges to p *local friends/nodes* and q *remote friends/nodes*. The p local friends are the p nodes (among all nodes) that are the closest to A in terms of grid distance. The q remote friends are chosen using q independent random trials. In each trial, a node B has a probability of $\rho \cdot dist^{-r}$ being chosen. Here $dist$ is the grid distance between A and B, and ρ is a constant normalization factor that makes the sum of all probabilities equal to 1. The parameter r is tunable between 0 and ∞. When $r = 0$, the remote friends are simply chosen uniformly randomly out of all nodes in the graph. As r increases, the remote friends tend closer and closer to A. We have experimented with various p, q, and r values. The following results use $r = 1.9$. For the million-node and 10000-node graph, we set $p = q = 8$. We use $p = q = 4$ for the 100-node graph. Results for other p, q, and r values we experimented with are qualitatively similar.

B. Results With No Malicious Users

We start by studying the basic behavior of SybilGuard when there are no malicious users. Without malicious users, the only property we are concerned with is whether an honest verifier accepts an honest suspect. This is affected by: (i) whether the random routes from the two nodes are loops; (ii) whether the random routes from the two nodes intersect; (iii) whether there is at least one intersection node online; and (iv) whether the needed length of random routes is properly estimated.

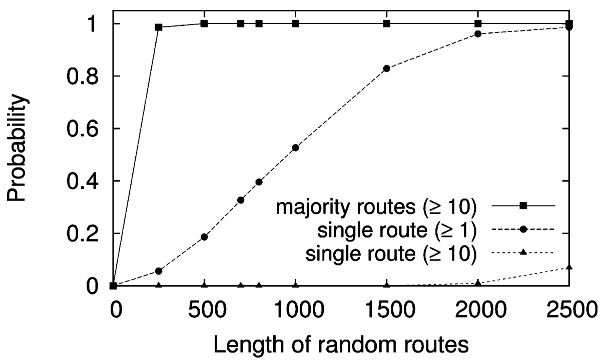

Fig. 10. Probability of having the given number of intersections. The legend "majority routes" means that each node performs random routes along all directions (and uses majority voting), while "single route" means performing a single random route. The legend "$(\geq x)$" means that we are considering the probability of having at least x distinct intersections. SybilGuard corresponds to "majority routes (≥ 10)".

Probability of random routes being loops. As discussed in Section IV-B, if a random route becomes a loop, then its effective length is reduced. Our simulation shows that 99.3% of the routes in the million-node graph do not form loops in their first 2500 hops (while later we will show that the needed length of the routes is below 2000). Furthermore, all the nodes in our simulation have at least one of their routes that is not a loop within their first 2500 hops. For the ten-thousand-node graph, 99.7% of the routes do not form loops in their first 200 hops, which is above the needed route length. For the hundred-node graph, 90% of the routes do not form loops in the first 50 hops, which is again above the needed route length.

As the results show that loops are quite rare, and also because they only impact effectiveness rather than security, we will not investigate them further. In all our results below, we do not distinguish loops from non-loops, and thus all the results will already capture the impact of random routes being loops.

Probability of an honest node being successfully accepted. We move on to study the probability of the verifier V accepting the suspect S. For V to accept S, their routes must intersect and at least one intersection must be online. We do not directly model nodes being online or offline. Rather, we assume that as long as there are at least 10 intersections, the verification succeeds. Note that even when nodes are online only 20% of the time, the probability that at least one out of 10 intersections is online is already roughly 90%.

Fig. 10 plots the probability of V successfully accepting S, as a function of w (length of the random routes). For better understanding, we also include in Fig. 10 two other curves for the cases where each node performs only a *single* random route, and seeks either at least 1 or 10 intersections. The results show that in a million-node social network, even having a w as small as 300 yields a 99.96% probability of having at least 10 intersections. On the other hand, if we do not route along all directions, the needed length will be much larger. For our ten-thousand-node

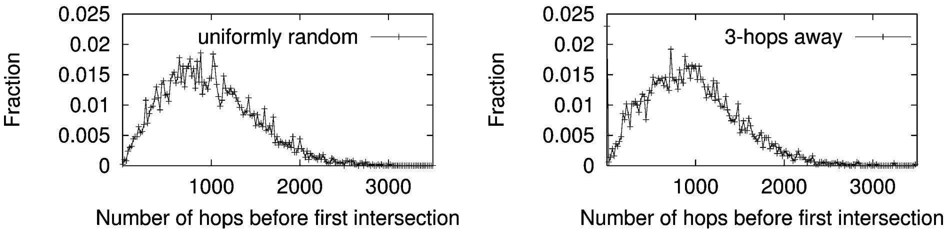

Fig. 11. Probability distribution histogram for the number of hops needed before the first intersection.

graph, $w = 30$ yields a 99.29% probability of having at least 10 intersections. For the hundred-node graph, $w = 15$ gives us a probability of 99.97%.

Estimating the needed length of the routes w. In SybilGuard, each node infers the needed length of the routes using the sampling technique described in Section IV-C. Using this technique, a node A first performs a short random walk ending at some node B. Then A and B both perform random routes to determine how long the routes need to be in order to intersect. Such estimation would be entirely accurate if (i) B were chosen uniformly randomly from all nodes in the system; and (ii) the number of samples were infinite. In practice, however, neither condition holds.

To gain insight into the impact of B not actually being a uniformly random node, Fig. 11 depicts the distribution of the number of hops before intersection, comparing the case when B is chosen uniformly at random to the case when B is chosen using a 3-hop random walk from A. As the figure shows, the two distributions are quite similar. This will help to explain later the small impact of B not being uniformly random. Based on the distribution when B is chosen uniformly at random, we obtain an accurate w of 1906 needed for 95% of the pairs to intersect. This value of 1906 will be used as a comparison with SybilGuard's estimated w.

To understand the error introduced by having only a finite number of samples, we study how the estimated w fluctuates and approaches 1906 as a node takes more and more samples. This experiment is repeated from multiple nodes. In all cases, we observe that the estimated w always falls within 1906 ± 300 after 30 samples. While after 100 samples, the estimated w always falls within 1906 ± 150. These results show that the estimated w is accurate enough even after a small number of samples. Even with only 30 samples and a worst case estimated w of 1606, Fig. 10 still shows a close-to-100% intersection probability when using majority routes. On the other hand, because taking each sample only involves a 3-hop random walk and the transfer of a witness table, the overhead is quite small. Finally, because the number of users n changes slowly and w changes roughly proportionally to $\sqrt{n}\log n$, we do not expect w to change rapidly. Thus a node needs only to re-estimate w, for example, on a daily basis. For our ten-thousand-node graph, the accurate w is 197, and the estimated w falls within 197 ± 30 after 35 samples. For the hundred-node graph, the accurate w is 24, and the estimated w falls within 24 ± 7 after 40 samples.

C. Results With Sybil Attackers

Next we study the behavior of SybilGuard when there are malicious users. We will use the term "sybil attacker" to refer to any such user, in order to distinguish the attacker from the potentially *unlimited* number of malicious nodes he creates. Sybil attackers influence the system by creating attack edges. There are clearly many possibilities regarding where the attack edges are in the graph, and we consider two extremes in our experiments. In `random`, we repeatedly pick uniformly random nodes in the graph as sybil attackers, until the total number of attack edges reaches a certain value. In `cluster`, we start from a random "seed" node and perform a breadth-first search from the seed. Nodes encountered are marked as sybil attackers, until the total number of attack edges reaches a certain value. All our results below are based on `random` placement, unless explicitly mentioned. We have obtained all corresponding results for `cluster` as well, which are always slightly better but the difference is usually negligible. Namely, under `cluster` the probability of accepting more than $g \cdot w$ sybil nodes is lower, the probability of an honest node being accepted is higher, and the estimates of w are more accurate, than under `random`. The reason for these better results under `cluster` is that the random routes are more likely to cross attack edges under `random`.

For our experiments based on the million-node graph, we vary the number of attack edges g from 0 to 2500. When $g = 2500$, there are roughly 100 nodes marked as sybil attackers. It is crucial to understand that just having 100 sybil attackers in the system will not necessarily result in 2500 attack edges—on average, each attacker must be able to convince 25 real human beings to be his friend. The hardness of creating these social links is what SybilGuard relies on.

In the presence of sybil attackers, we are concerned with several measures of "goodness": (i) the probability that an honest node accepts more than $g \cdot w$ sybil nodes; (ii) the probability that an honest node accepts another honest node; and (iii) the impact of sybil nodes on estimating w.

Probability of an honest node accepting more than $g \cdot w$ sybil nodes. Routes from an honest verifier V may enter the sybil region, and the adversary can then direct the routes to intersect with the routes of many sybil nodes. SybilGuard uses majority voting over all of V's routes to limit the influence of such problematic routes. The curve labeled "majority routes" in Fig. 12 shows the probability that the majority of an honest node's routes remain entirely in the honest region. Here we use $w = 1906$ as obtained before (the same is true for all the following experiments). If a majority of the routes are in the honest region, then the remaining routes will not constitute a majority, and the adversary will not be able to fool the node into accepting more than $g \cdot w$ sybil nodes. As shown in the figure, the probability is always almost 100% before $g = 2000$. Moreover, it is still 99.8% when $g = 2500$. This means that

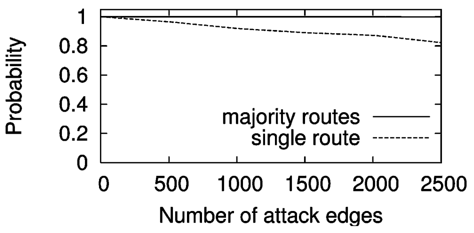

Fig. 12. Probability of routes remaining entirely within the honest region.

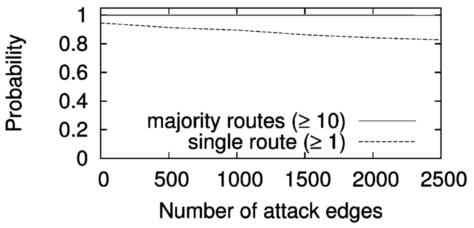

Fig. 13. Probability of an honest node accepting another honest node (i.e., having at least a target number of intersections). The legends are the same as in Fig. 10, and SybilGuard corresponds to “majority routes (≥ 10)”.

even with 2500 attack edges, only 0.2% of the nodes are not protected by SybilGuard. These are mostly nodes adjacent to multiple attack edges. In some sense, these nodes are “paying the price” for being friends of sybil attackers. For the ten-thousand-node graph and the hundred-node graph, $g = 204$ and $g = 11$ will result in 0.4% and 5.1% nodes unprotected, respectively. For better understanding, Fig. 12 also includes a second curve showing the probability of a single route remaining entirely in the honest region.

Probability of an honest node being successfully accepted. In the presence of sybil nodes, the probability that an honest verifier V accepts another honest suspect S decreases. First, the routes from S may enter the sybil region, and the adversary can prevent these routes from intersecting with V's routes. The same is true for V's routes. Second, the presence of sybil nodes necessitates the technique of using multiple routes. We use majority voting: among the d routes from V, at least $d/2$ routes need to successfully accept S before V can accept S.

To capture the worst case scenario, here we will assume that after a route (from V or S) enters the sybil region, the rest of the route can no longer be used for verification/intersection. In some sense, the presence of sybil nodes “truncates” the routes. As in Section VI-A, we assume that a (possibly truncated) route from V accepts S if it has at least 10 distinct intersections with S's (possibly truncated) routes. Finally, V successfully accepts S if a majority of V's routes accept S. At each trial, we select a random honest V and a random honest S.

Fig. 13 presents the probability of V accepting S, as a function of the number of attack edges g. This probability is still 99.8% with 2500 attack edges, which is quite satisfactory. The case with a single route is much worse (even if we seek only a single intersection), demonstrating that exploiting multiple routes is necessary. For the ten-thousand-node graph and the hundred-node graph, $g = 204$ and $g = 11$ give probabilities of 99.6% and 87.7%, respectively. Notice that a 87.7% probability does not mean that 12.3% of the nodes will not be accepted by the system. It only means that (i) given a random verifier, 12.3% of the nodes will not be accepted by that verifier, and (ii) a random honest node will not be accepted by 12.3% of the honest nodes (verifiers).

Estimating the needed length of the routes w. The final set of experiments seeks to quantify the impact of sybil nodes on the estimated w. Recall from Section IV-C that to estimate w, a node A performs a short (3-hop in our experiments) random walk ending at some node B. A and B then both perform random routes to determine when the two routes intersect, which is used as a sample. The sample taken is *bad* (i.e., potentially influenced by the adversary) if any of the two routes or the short random walk enters the sybil region. Our simulation shows that the probability of obtaining bad samples increases roughly linearly with the number of attack edges g. Even when g reaches 2500, the fraction of bad samples is still below 20%. Since our estimation uses the median of the samples, these 20% bad samples will have only limited influence on the estimate for w. For the ten-thousand-node graph and the hundred-node graph, the fraction of bad samples is always below 20% when $g \leq 204$ and $g \leq 11$, respectively.

VII. Related Work

The sybil attack [1] is a powerful threat faced by any decentralized distributed system (such as a p2p system) that has no central, trusted authority to vouch for a one-to-one correspondence between users and identities. As mentioned in Section I, the first investigation [1] into sybil attacks already proved a series of negative results.

Bazzi and Konjevod [4] proposed using network coordinates [5] to foil sybil attacks, and a similar idea has also been explored for sensor networks [20]. The scheme relies on the assumption that a malicious user can have only one network position, defined in terms of its minimum latency to a set of beacons. However, with network coordinates in a d-dimensional space, an adversary controlling more than d malicious nodes at d different network positions can fabricate an arbitrary number of network coordinates, and thus break the defense in [4]. This is problematic because d is usually a small number (e.g., < 10) in practice. Moreover, a solution based on network coordinates fundamentally can only bound the number of sybil groups and not the size of the sybil groups.

Danezis *et al.* [21] proposed a scheme for making DHT lookups more resilient to sybil attacks. The scheme leverages the bootstrap tree of the DHT, where two nodes share an edge if one node introduced the other into the DHT. The insight is that sybil nodes will attach to the rest of the tree only at a limited number of nodes (or attack edges in our terminology). One can imagine defining a similar notion of equivalence groups here, which correspond to subtrees. The scheme can then properly bound the number of sybil groups. In comparison, SybilGuard exploits the graph property in social networks instead of the bootstrap tree. This helps SybilGuard to further bound the size of sybil groups, which is not possible based on bootstrap trees. As a result, even with a single attack edge, the effectiveness of the scheme based on bootstrap tree deteriorates [21] as the adversary creates more and more sybil nodes. Furthermore, compromising even a single node in the bootstrap tree will disconnect the tree, breaking the assumption of the scheme.

Sybil attacks in sensor networks. Sybil attacks have also been studied for sensor networks [22]. The solutions there, such as radio resource testing and random key predistribution, unfortunately do not apply to distributed systems in the wide-area. A sybil-related attack in sensor networks is the *node replication attack* [23], where a single compromised sensor is replicated indefinitely, by loading the node's cryptographic information into multiple generic sensor nodes. All these replicated nodes have the same ID (e.g., they all have to use the same secret key issued to the compromised sensor). The solution [23], which is based on simple random walk intersection, does not extend to sybil attacks because the sybil nodes do not necessarily share a single, verifiable ID.

Sybil attacks have also been studied for sensor networks [22]. The solutions there, such as radio resource testing and random key predistribution, unfortunately do not apply to distributed systems in the wide-area. A sybil-related attack in sensor networks is the *node replication attack* [23], where a single compromised sensor is replicated indefinitely, by loading the node's cryptographic information into multiple generic sensor nodes. All these replicated nodes have the same ID (e.g., they all have to use the same secret key issued to the compromised sensor). The solution [23], which is based on simple random walk intersection, does not extend to sybil attacks because the sybil nodes do not necessarily share a single, verifiable ID.

Sybil attacks in reputation systems. In a reputation system, each user has a rating describing how well the user behaves. For example, eBay ratings are based on users' previous transactions with other users. Sybil attacks can create a large number of sybil nodes that collude to artificially increase a user's rating. Known defenses [24]–[26] against such attacks aim at preventing the sybil nodes from boosting a malicious user's rating (and attracting buyers, in the case of eBay). All of the sybil nodes are able to obtain the same rating/reputation as the malicious user. Unlike SybilGuard, these defenses cannot and do not aim to control the number of sybil nodes.

In some other reputation systems such as Credence [27], users cast votes regarding the validity of shared files. The votes are then combined using a weighted average based on the ratings of the user. Sybil nodes are able to dramatically influence the average (even when applying the techniques from [24]), and thus Credence relies on a central authority to limit sybil nodes [27].

Trust networks. The social network in SybilGuard is one kind of trust network. Many previous works [24], [25], [27] use trust networks that are based on past successful transactions or demonstrated shared interest between users. The trust associated with our social network is much stronger, which is essential to the effectiveness of SybilGuard. Such a strong-trust social network is also leveraged by LOCKSS [28], where the verifier accepts all its direct social friends, as well as a proportional number of other nodes. The total number of nodes accepted (proportional to the degree of the verifier) can be orders of magnitude smaller than the system size. Because a node can only accept and thus use a limited number of other nodes in the system, LOCKSS is more suited for specific application scenarios such as digital library maintenance. Ostra [29] leverages strong-trust social networks to prevent the adversary from sending excessive unwanted communication. In comparison, SybilGuard's functionality is more general: Since SybilGuard already bounds the number of sybil nodes, it can readily provide functionality equivalent to Ostra by allocating each node a communication quota. Furthermore, different from Ostra, SybilGuard has strong, provable end guarantees and has a complete design that is decentralized.

Trust propagation or transitive trust is a common technique for trust networks [24]–[27]. SybilGuard is more related to exploiting graph properties rather than trust propagation.

VIII. Conclusion

This paper presented SybilGuard, a novel decentralized protocol for limiting the corruptive influences of sybil attacks, by bounding both the number and size of sybil groups. SybilGuard relies on properties of the users' underlying social network, namely that (i) the honest region of the network is fast mixing, and (ii) malicious users may create many nodes but relatively few attack edges. In all our simulation experiments with one million nodes, SybilGuard ensured that (i) the number and size of sybil groups are properly bounded for 99.8% of the honest users, and (ii) an honest node can accept, and be accepted by, 99.8% of all other honest nodes.

The current SybilGuard design relies on the fast mixing property of social networks. If the social network is not fast mixing, SybilGuard will still properly bound the number of accepted sybil nodes within $g \cdot w$ (with high probability). The main drawback of a slower mixing social network is that more honest nodes will be (mistakenly) rejected. Although this paper already referred to several independent results confirming the fast mixing property based on social network models, our follow-on work [7] provides further assurance through an experimental study based on real and large-scale social networks. That work also presents a revised protocol that reduces the number of sybil nodes accepted per attack edge from $O(\sqrt{n}\log n)$ to $O(\log n)$.

Other future work includes deploying SybilGuard in real applications. Important issues include how to bootstrap the social network (can we leverage an existing social network, can we make it easy to join the social network, etc.) and what applications can best benefit from SybilGuard's fully decentralized approach.

Acknowledgment

The authors thank D. Andersen, T. Anderson, M. Freedman, P. Maniatis, A. Perrig, S. Seshan, and the anonymous SIGCOMM'06 and ToN reviewers for many helpful comments on the paper.

References

[1] J. R. Douceur, "The sybil attack," in *Proc. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar. 2002, 6 pp.

[2] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *Proc. ACM SIGCOMM 2006*, Pisa, Italy, Sep. 2006, pp. 291–302.

[3] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," in *Proc. Eurocrypt 2003*, Warsaw, Poland, May 2003, pp. 294–311.

[4] R. Bazzi and G. Konjevod, "On the establishment of distinct identities in overlay networks," in *Proc. 24th ACM Symp. Principles of Distributed Computing (PODC 2005)*, Las Vegas, NV, Jul. 2005, pp. 312–320.

[5] T. S. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *Proc. IEEE INFOCOM 2002*, New York, NY, Jun. 2002, pp. 170–179.

[6] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: Defending against sybil attacks via social networks," in *Proc. ACM SIGCOMM 2006*, Pisa, Italy, Sep. 2006, pp. 267–278.

[7] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "SybilLimit: A near-optimal social network defense against sybil attacks," in *Proc. 2008 IEEE Symp. Security and Privacy*, Oakland, CA, May 2008, pp. 3–17.

[8] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM 2001*, San Diego, CA, Aug. 2001, pp. 149–160.

[9] M. Mitzenmacher and E. Upfal, *Probability and Computing*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[10] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: Design, analysis and applications," in *Proc. IEEE INFOCOM 2005*, Miami, FL, Mar. 2005, pp. 1653–1664.

[11] A. Flaxman, "Expansion and lack thereof in randomly perturbed graphs," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2006-118, Aug. 2006 [Online]. Available: ftp://ftp.research.microsoft.com/pub/tr/TR-2006-118.pdf

[12] I. Abraham and D. Malkhi, "Probabilistic quorums for dynamic systems," in *Proc. DISC 2003*, Sorrento, Italy, Oct. 2003, pp. 60–74.

[13] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. Marsh, "Efficient lookup on unstructured topologies," in *Proc. 24th ACM Symp. Principles of Distributed Computing (PODC 2005)*, Las Vegas, NV, Jul. 2005, pp. 77–86.

[14] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "SybilGuard: Defending against sybil attacks via social networks," Intel Research Pittsburgh, Pittsburgh, PA, Tech. Rep. IRP-TR-06-01, Jun. 2006 [Online]. Available: http://www.comp.nus.edu.sg/~yuhf/sybilguard-tr.pdf

[15] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," in *Proc. ACM SIGMETRICS 2000*, Santa Clara, CA, Jun. 2000, pp. 34–43.

[16] International Network for Social Network Analysis. 2006 [Online]. Available: http://www.insna.org/INSNA/data_inf.htm

[17] Center for Computational Analysis of Social and Organizational Systems (CASOS). 2006 [Online]. Available: http://www.casos.cs.cmu.edu/computational_tools/data.php

[18] J. Kleinberg, "The small-world phenomenon: An algorithm perspective," in *Proc. ACM Symp. Theory of Computing (STOC 2000)*, Portland, OR, May 2000, pp. 163–170.

[19] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, 1998.

[20] N. Sastry, U. Shankar, and D. Wagner, "Secure verification of location claims," in *Proc. ACM Workshop on Wireless Security (WiSE'03)*, San Diego, CA, Sep. 2003, 10 pp.

[21] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson, "Sybil-resistant DHT routing," in *Proc. European Symp. Research in Computer Security (ESORICS 2005)*, Milan, Italy, Sep. 2005, pp. 305–318.

[22] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil attack in sensor networks: Analysis & defenses," in *Proc. 3rd Int. Symp. Information Processing in Sensor Networks (IPSN 2004)*, Berkeley, CA, Apr. 2004, pp. 259–268.

[23] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *Proc. IEEE Symp. Security and Privacy*, Oakland, CA, May 2005, pp. 49–63.

[24] A. Cheng and E. Friedman, "Sybilproof reputation mechanisms," in *Proc. 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON-05)*, Philadelphia, PA, Aug. 2005, pp. 128–132.

[25] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proc. ACM Electronic Commerce (EC'04)*, New York, NY, May 2004, 10 pp.

[26] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *Proc. 2nd Int. Semantic Web Conf. (ISWC2003)*, Sanibel Island, FL, Oct. 2003, pp. 351–368.

[27] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *Proc. 3rd USENIX Symp. Networked Systems Design and Implementation (NSDI 2006)*, San Jose, CA, May 2006, pp. 1–14.

[28] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, and M. Baker, "The LOCKSS peer-to-peer digital preservation system," *ACM Trans. Comput. Syst. (TOCS),*, vol. 23, no. 1, pp. 2–50, 2005.

[29] A. Mislove, A. Post, K. Gummadi, and P. Druschel, "Ostra: Leveraging trust to thwart unwanted communication," in *Proc. 5th USENIX Symp. Networked Systems Design and Implementation (NSDI 2008)*, San Francisco, CA, Apr. 2008, pp. 15–30.

Haifeng Yu received the B.E. degree from Shanghai Jiaotong University, China, in 1997, and the M.S. and Ph.D. degrees from Duke University, Durham, NC, in 1999 and 2002, respectively.

He is currently an Assistant Professor in the Department of Computer Science, National University of Singapore (NUS). Prior to joining NUS, he was a Researcher at Intel Research Pittsburgh and an Adjunct Assistant Professor in the Department of Computer Science, Carnegie Mellon University. His research interests cover the general area of distributed systems, as well as related fields such as fault tolerance, large-scale peer-to-peer systems, distributed computing, and security. More information about his research is available at http://www.comp.nus.edu.sg/yuhf.

Michael Kaminsky received the B.S. degree from the University of California at Berkeley, and the S.M. and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, MA.

He is currently a Research Scientist at Intel Research Pittsburgh and an Adjunct Research Scientist at Carnegie Mellon University, Pittsburgh, PA. He is generally interested in computer science systems research, including distributed systems, networking, operating systems and network/systems security.

Dr. Kaminsky has been a member of the ACM since 2004. More information about his research is available at http://www.pittsburgh.intel-research.net/people/kaminsky/.

Phillip B. Gibbons (M'89) received the B.A. degree in mathematics from Dartmouth College in 1983 and the Ph.D. degree in computer science from the University of California at Berkeley in 1989.

He is currently a Principal Research Scientist at Intel Research Pittsburgh. He joined Intel after 11 years at (AT&T and Lucent) Bell Laboratories. His research interests include parallel computing, databases, and sensor systems, with over 100 publications.

Dr. Gibbons has served as an Associate Editor for the *Journal of the ACM*, the IEEE TRANSACTIONS ON COMPUTERS, and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He has served on over 35 conference program committees, including serving as program chair/co-chair/vice-chair for the SPAA, SenSys, IPSN, DCOSS, and ICDE conferences. He has been an ACM Fellow since 2006. More information about his research is available at http://www.pittsburgh.intel-research.net/people/gibbons/.

Abraham D. Flaxman received the B.S. degree from the Massachusetts Institute of Technology, Cambridge, MA, and the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA.

He is currently a postdoctoral researcher with the Microsoft Research Theory Group, Redmond, WA. He is interested in measurement, modeling, algorithms, and decision making for complex networks.

Dr. Flaxman has been a member of the ACM since 2006.