Expanded Technical Report. The original version appears
In *Proceedings of the ACM SIGCOMM Conference on Computer Communications (SIGCOMM 2006)*, Pisa, Italy, September 2006

SybilGuard: Defending Against Sybil Attacks via Social Networks

Haifeng Yu[†], Michael Kaminsky[†], Phillip B. Gibbons[†], Abraham Flaxman[*]
[†]*Intel Research Pittsburgh* [*]*Microsoft Research*

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.

1 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* [10], 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 [21]. 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 [10] 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

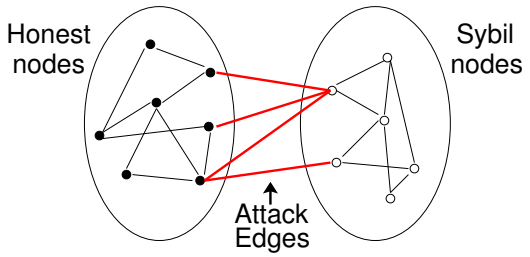

Figure 1: The social network with honest nodes and sybil nodes. Note that regardless of which nodes in the social network are sybil nodes, we can always "pull" these nodes to the right side to form the logical network in the figure.

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 [25], 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 [19] 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 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 nicely 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.

SybilGuard: A new defense against sybil attacks. 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* (Figure 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.

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.

The next section more precisely defines our system model and the sybil attack. Section 3 provides an overview of SybilGuard. Sections 4 and 5 elaborate on SybilGuard in depth. The effectiveness of SybilGuard is shown experimentally in Section 6. Finally, Section 7 discusses related work and Section 8 draws conclusions.

2 Model & 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 from now on use "identity" and "node" interchangeably. All malicious users may collude, and we say that they are all under the control of an *adversary*.

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

guarantee that V accepts only honest nodes. Because such an idealized guarantee is challenging to achieve, we aim at providing the following guarantees that, while weaker, are still sufficiently strong to be useful.

Bounding the number of sybil groups. The first guarantee is based on defining an *equivalence relation* among accepted nodes. The equivalence relation partitions all accepted nodes into equivalence classes, called *equivalence groups*. Notice that 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*. The defense system provides a guaranteed bound on the *number* of sybil groups, without necessarily knowing which groups are sybil.

Such notion of equivalence groups was also implicitly used by Bazzi and Konjevod [4], where they define (implicit) equivalence classes according to network coordinates. In their scheme, all nodes are accepted, and those nodes with similar network coordinates (e.g., nodes within the same university campus) are considered equivalent. Thus, the number of sybil groups is simply the number of distinct network locations that the adversary controls.

To understand why bounding the number of sybil groups is sufficient in some scenarios, imagine that we are maintaining replicas of a file that has been digitally signed for authenticity. Our goal is to ensure that not all replicas are placed on sybil nodes. If the defense system guarantees that the number of sybil groups is at most some value g, then placing the file on nodes from $g + 1$ different equivalence groups will ensure at least one good copy of the file. Another example is replicating a file that is not signed. As long as we obtain the file from $2g+1$ nodes from $2g+1$ different equivalence groups, the majority is guaranteed to have the correct file.

Bounding the size of sybil groups. In some other scenarios, only bounding the number of sybil groups is not effective. Unavoidably, the bound on the number of sybil groups depends on how "powerful" the adversary is. For example, the adversary can always "bribe" or even threaten honest users to act maliciously and thus force the defense system to accept more sybil groups. As a result, one may want a pessimistic estimation of the number of sybil groups g. On the other hand, even when g is only moderately large (e.g., 100), maintaining $g + 1$ replicas is wasteful.

To be more effective, a defense system may further bound the number of nodes accepted into each of the g sybil groups. If the number of nodes in each sybil group (or the *size* of the sybil group) is at most w, then a node will accept at most $g \cdot w$ sybil nodes. To see the benefits of bounding both the number and size of the sybil groups, consider our running example of replicating unsigned and signed files. Suppose we use a simple assignment that maps replicas to random nodes. If $g \cdot w$ is smaller than the number of honest nodes n, then from Chernoff bounds [16], the probability of having a majority of the replicas on honest nodes (as required for unsigned files) approaches 1.0 exponentially fast with the number of replicas. Similarly, as long as $g \cdot w$ is not much larger than n, the probability of having at least one replica on an honest node (as required for signed files) also approaches 1.0 exponentially fast.

Choosing roughly uniformly random nodes as replicas is not difficult in most decentralized distributed systems. For example, DHT-based systems (such as those based on Chord [24]) typically place replicas on a random set of nodes. It may appear that instead of choosing uniformly random nodes, we could avoid the need for bounding sybil group sizes by instead choosing uniformly random equivalence groups (and then picking a random node from each chosen group). However, such a design would cause severe load imbalance under heterogeneous group sizes, which is the case, for example, in the network coordinates approach. Moreover, for DHT-based systems, the design would completely disrupt DHT routing.

Side-effects on honest nodes. As side-effects of bounding the number and size of sybil groups, the defense system may both (mistakenly) reject some honest nodes and (mistakenly) consider two or more distinct honest nodes as equivalent. For example, as noted above, all honest nodes in the same university campus may be considered equivalent in the network coordinates approach.

Summary of SybilGuard functionalities. SybilGuard is completely decentralized and all functionalities are with respect to a given node. 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. SybilGuard also guarantees that with high probability, an honest node only accepts a bounded number of sybil nodes. Notice that since 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.

SybilGuard further enables a node V to partition the accepted nodes (by V) into equivalence groups such that only a certain number of those groups contain sybil nodes. Notice that if the application only wants to bound the number of sybil nodes accepted, the notion of equivalence groups does not need to be visible to the application. It is possible for two distinct honest users to be mistakenly considered by SybilGuard to belong to the same equivalence group. This does not affect their ability to receive service. As for providing

service, the application may prevent them from, for example, both storing replicas of the same file. As argued in [4], as long as there are a sufficiently large number of equivalence groups, this will not likely result in wasted resource capacity.

3 SybilGuard Overview

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 Figure 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. 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.

Random routes and route intersection. 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. As a result, two random routes entering an honest node along the same edge will always exit along the same edge (called the *convergence property*). Furthermore, the outgoing edge uniquely determines the incoming edge as well; thus the random routes can be back-traced (called the *back-traceable property*). 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.

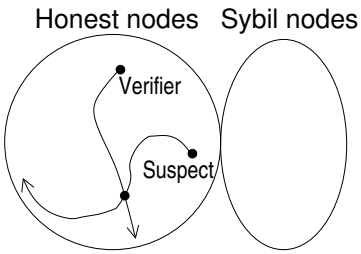

Figure 2: Verifier accepts the suspect because their random routes intersect. SybilGuard leverages the facts that (1) the average honest node's random route is highly likely to stay within the honest region and (2) two random routes from honest nodes are highly likely to intersect within w steps.

In the simplest form of SybilGuard, each node performs a random route (starting from itself)[1] of a certain length w (e.g., w is roughly 2000 for the one-million node topology in our later experiments). These random routes form the basis of SybilGuard whereby an honest node (called the *verifier*) decides whether or not to accept another node (called the *suspect*). In particular, the verifier only accepts a suspect whose random route intersects with the verifier's random route (see Figure 2). Because of the limited number of attack edges, with appropriate w, the verifier's route will remain entirely within the honest region with high probability. (An exception is a verifier with a nearby attack edge; our redundancy techniques discussed in Section 4.4 will address such nodes.)

Bounding the number and size of sybil groups. To intersect with the verifier's random route, 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 Figure 3). 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 will have the same intersection node with the verifier's route; 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.

[1] In the full protocol, each node performs multiple random routes.

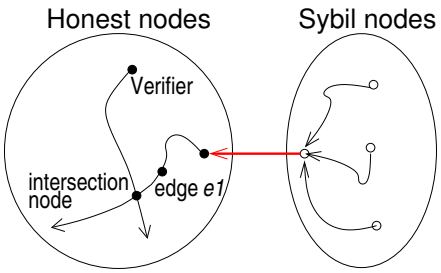

Figure 3: All random routes traversing the same edge merge.

SybilGuard further bounds the size of equivalence groups (and hence of sybil groups) within the length of the random routes w. 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 Figure 3). Specifically, the ith such route, $i = 1, \ldots, w$, traverses the given edge in its ith hop. 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.

Guarantees on honest 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.

SybilGuard partitions the honest nodes in the system into at most z different equivalence groups, where z is the sum of the degrees of the w nodes on the verifier's route. While z can still be far from n, note that z can easily be much larger than the number of different equivalence groups needed in practice (e.g., when choosing $g + 1$ different equivalence groups for placing replicas).

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 the next section), 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.

4 SybilGuard Design

With the preceding high-level sketch in mind, this section provides the detailed design of SybilGuard, explains the insights, and also formally argues about its properties.

4.1 Social Network

Consider the social network defined in the previous section. Each pair of friends 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). A node can also revoke an edge key unilaterally simply by discontinuing use of the key and discarding it.

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 [7, 12] (also see Section B of Appendix).

A node informs its friends of its IP address whenever its IP address changes, to allow continued communication via the network. 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.

4.2 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 tries to convince Alice to also be friends with these sybil nodes. But 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. 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.

Figure 4: Two routes of length 3. Sharing an edge necessarily means that one route starts after the other.

4.3 Random Routes

Starting from here, the rest of Section 4 assumes a static social network where all nodes are online—we will discuss user and node dynamics in Section 5. SybilGuard relies on the convergence and back-traceable properties in random routes to bound the number and size of sybil groups. Here, we elaborate on how to achieve these properties and their implications.

For random routes, 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 5). Using such a randomized routing table introduces some correlation in the random choices if a random route visits the same node multiple times. It is possible that random routes become repeated loops due to this; however, later we will explain intuitively and also demonstrate experimentally why this is unlikely.

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 (i.e., the convergence property). Using a permutation as the routing table further guarantees that the random routes are back-traceable. In other words, it is impossible for two routes to enter the same node along different edges but exit along the same direction. With the above 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 (Figure 4).

4.4 Problematic Routes and Redundancy

A random route is *problematic* if either (i) it traverses some **edge** in the same direction more than once (i.e., a *loop*), or (ii) it enters the sybil region. Note that a route traversing the same **node** more than once may or may not be a loop. Because of the use of routing tables, loops will repeatedly visit many nodes, reducing the "effective" length of the route

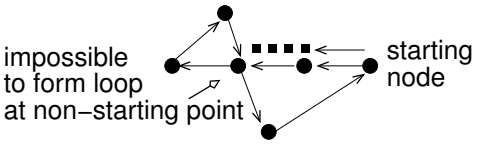

Figure 5: A loop can form only at the starting point of a route.

and the probability of route intersection. On the other hand, random routes that go into the sybil region fall under the control of the adversary. If a verifier uses such a route, it may accept an unbounded number of sybil nodes.

Because the routing table is a permutation, if a random route ever traverses the same edge twice in the same direction, the first edge in the route must be the first edge that is traversed twice. In other words, loops can only form at the starting node (Figure 5). 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 (via the same direction) 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 loop is formed at the third hop with probability $1/d^2$. As the route proceeds, the chance of repeating the first hop edge 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. As for the probability of a random route extending to the sybil region, we will later formally argue (Theorem 1) why this probability is also likely to be small. Finally, Section 6 will provide concrete experimental results demonstrating that problematic random routes are relatively rare.

An effective way to further avoid problematic random routes is to use redundancy. In SybilGuard, a node with degree d performs d random routes, one along each of its edges. Now imagine that a verifier V tries to decide whether to accept a suspect S. Those routes that are loops can still be used, because they do not compromise security—they are simply less "effective." We can also safely use all routes from S regardless of whether they extend to the sybil region: If S is an honest node, then using all routes simply increases the probability of some route intersecting with V's routes. On the other hand, if S is a sybil node, then all of S's routes still need to cross the attack edges before intersecting with V's routes that are in the honest region. Because of the convergence property, we can easily see that this will not compromise SybilGuard's guarantees (Section 3).

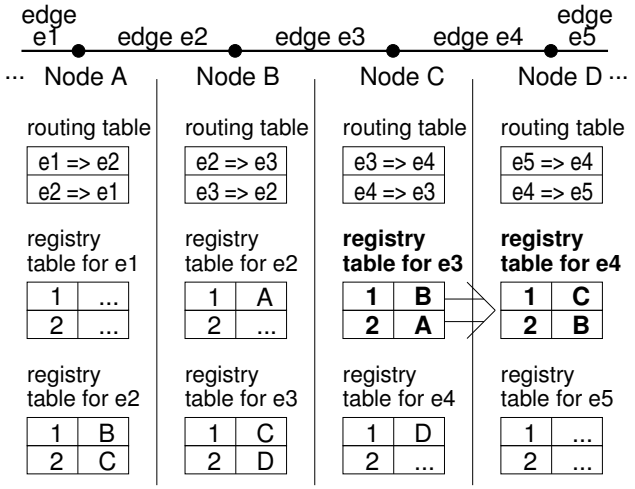

Figure 6: Maintaining the registry tables. In order to simplify this example, $w = 2$, each node has exactly two edges, and the routing tables are carefully chosen. The node names in the registry tables stand for the nodes' public keys.

On the other hand, if a route from V extends to the sybil region, V will not be able to bound the number of sybil nodes using that route. V uses the following technique to mask the misleading effects of routes extending to the sybil region. For each of V's routes, as long as at least one route from S intersects that route from V, that route from V *accepts* S. V *accepts* S if and only if at least a threshold t of V's routes accept S. The parameter t involves the following tradeoff: 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. 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.

4.5 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 perform random routes.

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". 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 of an honest node (unless the node is compromised). Our initial design of SybilGuard used a node's IP address as its token and the node simply registered its IP address. This design assumed no IP spoofing, and was mainly suited for users with static or slowly changing IP addresses.

Our current design of SybilGuard uses public key cryptography for the tokens. This improved design does not rely on the stability of IP addresses, and is secure even under IP spoofing. 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 intend or 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 (Figure 6). 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 Figure 6. 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 Figure 6). C will forward its registry table for e_3 to its neighbor D along e_4, via a secure 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 Figure 6, 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 256KB when using 1024-bit public keys. For a node with 10 neighbors, the total data sent is 2.56MB. 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 400KB. Finally, it is important to notice that registry table updates are needed only when social trust relationships change (Section 5). 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 5.1.)

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. Figure 7 depicts the process for a node

1. S sends its witness tables and public key to V
2. For each of V's 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 half of V's routes accept S, V accepts S

Figure 7: Protocol for a node V to verify a node S

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 [16] 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.

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 half of V's routes accept S, V accepts S (i.e., S's public key). 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.

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 there are altogether, $n \cdot d \cdot w$ registry table entries in the system.

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 $w+(w-1)+\ldots+1 \approx w^2/2$ entries system-wide. With g attack edges and even when gw approaches n, the total number of polluted entries ($gw^2/2$) is still less than half of 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.

4.6 Designing the Length of Random Routes

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 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 enables 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.

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 approach the stationary distribution. In other words, after $\Theta(T)$ hops, 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) [7, 12] (also see Section B of Appendix).

Our first theorem considers the probability that a random walk starting from a random honest node enters the sybil region of the topology.

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 gw/n. In particular, when $w = \Theta(\sqrt{n}\log n)$ and $g = o(\sqrt{n}/\log n)$, this probability is $o(1)$.*

Proof: The entire social network $\mathcal{G}$ contains both honest nodes and sybil nodes. To prove the theorem, we first transform $\mathcal{G}$ to a new graph $\mathcal{G}'$ which enables us to later talk about the stationary distribution of $\mathcal{G}'$. The graph $\mathcal{G}'$ contains all honest nodes in $\mathcal{G}$ and also all the edges between honest nodes. In addition, $\mathcal{G}'$ contains a *sink node* that represents all sybil nodes in $\mathcal{G}$. For each edge in $\mathcal{G}$ that connects an honest node with a sybil node, we add an edge in $\mathcal{G}'$ to connect that honest node with the sink node. Notice that it is possible for a single honest node to have multiple edges to the sink node. Obviously, if $\mathcal{G}$ is connected, then $\mathcal{G}'$ must be connected as well. Finally, we add a self-loop at the sink node (i.e., both end points of the edge is the sink node). This self-loop serves to make $\mathcal{G}'$ non-bipartite, which is needed to ensure that the stationary distribution is well-defined on $\mathcal{G}'$. In $\mathcal{G}'$, we say that any edge connecting an honest node with the sink node is an *attack edge*. Clearly, the number of attack edges is the same in $\mathcal{G}'$ as in $\mathcal{G}$.

In the following, we first study in $\mathcal{G}'$ the probability that a length-w random walk starting from a uniformly random node will ever traverse any of the g attack edges along the direction of entering the sink node. Notice here that the walk starts from a random node instead of a random honest node, and also we consider only traversing attack edges in one direction. After studying such probability, we will draw the connection between such probability in $\mathcal{G}'$ and the probability in $\mathcal{G}$, and complete the proof of the theorem.

In $\mathcal{G}'$, we number the n honest nodes from 1 through n in an arbitrary way. The sink node is numbered $n+1$. In $\mathcal{G}'$, let p'_i $(1 \le i \le n+1)$ be the probability that a random walk of length w starting from node i ever traverses any attack edge along the direction of entering the sink node. Let q'_i $(1 \le i \le n+1)$ be the probability of starting the random walk from node i. Clearly, the probability in $\mathcal{G}'$ that a length-w random walk starting from a uniformly random node will ever traverse any of the g attack edges along the direction of entering the sink node should be $\sum_{i=1}^{n+1} p'_i \cdot q'_i$ where $q'_i = 1/(n+1)$ for $1 \le i \le n+1$. Lemma 2 will prove that $\sum_{i=1}^{n+1} p'_i/(n+1) \le g \cdot w/(n+1)$.

Next, we draw the connection between the probability of walks traversing any attack edge along the direction of entering the sink node in $\mathcal{G}'$ and the probability of walks traversing any attack edge in $\mathcal{G}$. For a given honest node i $(1 \le i \le n)$ in $\mathcal{G}$, we define p_i to be the probability that a random walk of length w starting from that honest node ever traverses any attack edge (along any direction). We will show that $p_i = p'_i$ for $1 \le i \le n$ as follows. In graph $\mathcal{G}$, we define $\mathcal{W}$ to be the finite set of all length-w random walks starting from honest node i that do not traverse any attack edge. Similarly in graph $\mathcal{G}'$, we define $\mathcal{W}'$ to be the finite set of all length-w random walks starting from honest node i that do not traverse any attack edge along the direction of entering the sink node. Notice that in $\mathcal{G}'$, it is impossible for a random walk starting from an honest node, to traverse any attack edge along the direction of leaving the sink node without first traversing some attack edge to enter the sink node. Thus it is easy to see that $\mathcal{W} \subseteq \mathcal{W}'$ and $\mathcal{W}' \subseteq \mathcal{W}$, and thus $\mathcal{W} = \mathcal{W}'$. Next notice that an honest node always has the same degree in $\mathcal{G}'$ as in $\mathcal{G}$. Thus for each random walk in $\mathcal{W}$, the probability of it happening is exactly the same in $\mathcal{G}$ as in $\mathcal{G}'$. This implies that $1 - p'_i = 1 - p_i$ and thus $p'_i = p_i$ $(1 \le i \le n)$.

Finally, Lemma 2 tells us that $\sum_{i=1}^{n+1} p'_i \le g \cdot w$. The probability in the theorem is then $\sum_{i=1}^{n} p_i/n = \sum_{i=1}^{n} p'_i/n \le \sum_{i=1}^{n+1} p'_i/n \le g \cdot w/n$. □

Lemma 2 *In $\mathcal{G}'$, the probability that a length-w random walk starting from a uniformly random node will ever traverse any of the g attack edges along the direction of entering the sink node is at most $g \cdot w/(n+1)$. In other words, $\sum_{i=1}^{n+1} p'_i \cdot q'_i \le g \cdot w/(n+1)$ when $q'_i = 1/(n+1)$ for $1 \le i \le n+1$.*

Proof: Directly reasoning about this probability is challenging. But if instead of starting the random walk from a uniformly random node, we start the walk from the stationary distribution of $\mathcal{G}'$ (i.e., q'_i follows the stationary distribution), then every hop in the walk follows the stationary distribution. A simple union bound will give us the probability of the walk traversing attack edges. Such probability can then

be translated back to the case where the starting node is a uniformly random node.

We formalize the above idea as follows. Let d_i ($1 \leq i \leq n+1$) be the degree of node i and M be the total number of edges in $\mathcal{G}'$. If we set q'_i to $d_i/(2M)$, then the starting node follows the stationary distribution of $\mathcal{G}'$. From the properties of stationary distributions, (1) every hop in the walk enters a random node that again follows the stationary distribution, and (2) every hop in the walk traverses a uniformly random edge along a uniformly random direction. The probability that a hop traverses one of the g attack edges in $\mathcal{G}'$ along the direction of entering the sink node is thus $g/(2M)$. A simple union bound tells us that the probability of a length-w random walk traversing any attack edge is upper bounded by $g \cdot w/(2M)$. This gives us $\sum_{i=1}^{n+1} p'_i \cdot d_i/(2M) \leq g \cdot w/(2M) \Rightarrow \sum_{i=1}^{n+1} p'_i \leq g \cdot w \Rightarrow \sum_{i=1}^{n+1} p'_i/(n+1) \leq g \cdot w/(n+1)$. □

We should point out that Theorem 1 provides only an "average" guarantee for all honest nodes. Those honest nodes that are closer to attack edges are likely to have a larger probability of walking into the sybil region. Our later simulation results, however, will show that using the redundancy techniques from Section 4.4 will give most nodes a high probability of success.

The next 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 rigorously proved [3, 17] in several other contexts, and thus we only give a high-level review. As noted above, many randomly-grown topologies are fast mixing (i.e., the mixing time is $\Theta(\log n)$ hops), including social networks (or more specifically, small-world topologies). Thus, a walk of $\Theta(\sqrt{n}\log n)$ hops contains $\Theta(\sqrt{n})$ independent samples drawn roughly from the stationary distribution. When the verifier's and the suspect's walks remain in the honest region, both walks draw $\Theta(\sqrt{n})$ independent samples from roughly the same distribution. It follows from the generalized Birthday Paradox [3, 17] that they intersect with probability $1 - o(1)$.

Random routes vs. 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 separate cases and explain the bahavior of random routes, as compared to random walks. If $j = i$, then we have a loop and the random route will traverse this loop repeatedly, which clearly deviates significantly from the behavior of a random walk. However, earlier we explained why loops tend to be rare. 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.

4.7 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 estimations on 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 $\Theta(T)+w$. Since $w = \Theta(\sqrt{n}\log n)$ 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 6 for the number of samples needed). It then sets $w = 2.1m$, where the constant 2.1 is derived from our analysis of Birthday Paradox distributions (see Section A of Appendix). 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.

5 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. The social network in this paper always includes all users/nodes that have been created and not yet deleted. In other words, many of the nodes in the graph can be offline at any given time.

5.1 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 (Section 4.4), there will likely be multiple intersections. In fact, even a single route from V and a single route from S may still have multiple intersections. The verification can be done as long as a majority of V's routes have at least one intersection point online.

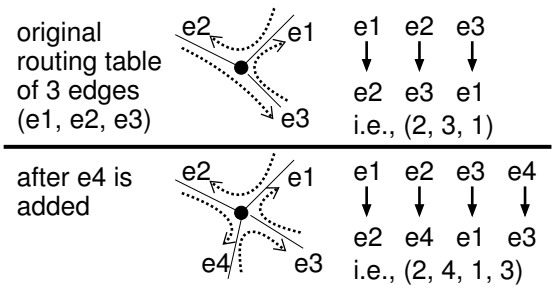

Figure 8: Incremental maintenance of routing tables. The example assumes that $d = 3$ and $k = 2$. Note that after edge $e4$ is added, only routes entering via edge $e2$ need to be redirected.

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 propagation may also be needed when IP addresses change, but such updating can be performed lazily (Section 4.5). Previous studies [6] 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 will explain such mechanism in Section 5.4.

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.

5.2 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 only need to discuss 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 (Figure 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 $1 \leq k \leq d$, 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 (if exists) 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. In the following, we show via induction why the resulting routing tables are indeed uniformly random permutations. We first consider edge creation. Suppose we start from a uniformly random permutation "$x_1, x_2, \ldots, x_d$". We want to show that after adding a new edge, the new permutation (denoted as "$y_1, y_2, \ldots, y_d, y_{d+1}$") remains a uniformly random permutation. In other words, the probability of being any specific permutation should be $1/(d+1)!$. It suffices to show that i) the probability of $(d+1)$ being at the kth (for $1 \leq k \leq n+1$) position in "$y_1, y_2, \ldots, y_d, y_{d+1}$" is exactly $1/(n+1)$, and ii) given that $(d+1)$ is in the kth position, the rest of the sequence (excluding d) is a uniformly random permutation. On the other hand, these two properties directly follow from the above incremental maintenance algorithm.

Next we consider the case of edge deletion, and start from a uniformly random permutation "$y_1, y_2, \ldots, y_d, y_{d+1}$". We want to show that after deleting an edge, the new permutation (denoted as "$x_1, x_2, \ldots, x_d$") remains uniformly random. For a given instance of "$x_1, x_2, \ldots, x_d$", there are $n+1$ possible sequences in the form of "$y_1, y_2, \ldots, y_d, y_{d+1}$" that can be the permutation before the deletion. These $n+1$ sequences are:

$$(d+1), x_2, x_3, \ldots, x_d, x_1$$
$$x_1, (d+1), x_3, \ldots, x_d, x_2$$
$$\ldots$$
$$x_1, x_2, x_3, \ldots, x_d, (d+1)$$

Each of these sequences occurs with probability of exactly $1/(n+1)!$, which means that the given instance of "$x_1, x_2, \ldots, x_d$" occurs with probability of $(n+1) \cdot 1/(n+1)! = 1/n!$.

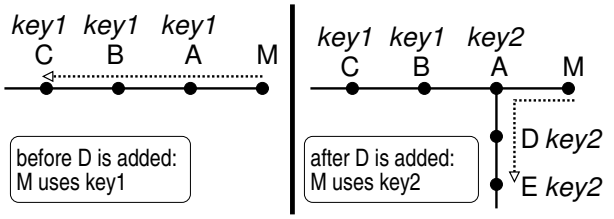

Figure 9: A potential attack by M during node dynamics.

5.3 Attacks Exploiting Node Dynamics

This section shows that performing random routes along *all* directions (Section 4.4) actually 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 Figure 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, there must exist another node F somewhere, such that one of F's routes visits B at the ith hop. 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". 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.

5.4 Bypassing Offline Nodes during Registry/Witness Table Propagation

To further facilitate registry and witness table propagation in a social network where nodes can be offline, SybilGuard allows a node to bypass offline nodes when propagating such tables. Our initial design of SybilGuard used lookahead routing tables for this purpose; these lookahead routing tables record which nodes the route should traverse on the next k hops.

Our current, improved SybilGuard design avoids the need for explicitly constructing such lookahead routing tables. Instead, SybilGuard directly leverages *stale* registry tables to facilitate propagating *new* registry tables. The same is for witness tables. To enable this functionality, SybilGuard needs to record node IP addresses in both registry tables and witness tables (instead of only in witness tables as in Section 4.5). These IP addresses are always used as hints to locate the nodes – stale/incorrect IP addresses will not comprise security and will only prevent a node from bypassing other offline nodes.

Design for bypassing offline nodes. Given that registry tables also record IP addresses, registry tables and witness tables now become completely symmetric. Thus, we only need to discuss registry table update and propagation – witness table propagation is similar. Consider a random route $\ldots \rightarrow X \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow \ldots$, and assume that all registry and witness tables are up-to-date initially on the four nodes. Suppose B is offline when A receives a new registry table for the edge between X and A (making A's current table stale). A now tries to propagate the new table downstream. A's witness table for the edge between A and B should already contain C and D's IP addresses. Thus A should be able to locate C and D and send them the updated registry table despite B being offline. On the other hand, from C's perspective, its registry table for the edge between B and C should already contain A's public key. C can easily use such information to authenticate the update from A. The same is for D.

Generalizing from the above example, SybilGuard allows any node (the sender) to send an updated registry table to any other node (the receiver) in the sender's witness table. More specifically, suppose the sender has an updated registry table coming along some edge e_1, and according to its local routing table, it needs to propagate the table along some edge e_2. Our design allows the sender to directly propagate the table to any of the w nodes downstream. In the extreme, the sender may even propagate the table, in parallel, to all downstream nodes that are online. To propagate the registry table to the hth node downstream, the sender uses the IP address recorded in the hth entry of the witness table for e_2. The sender then sends to that IP address i) the new registry table, ii) the hop number h, iii) its own public key key_{own}; and iv) the public key key_{next} in the first entry of the witness table for e_2. The last three pieces of information serve to help the receiver to determine how to apply the update. The entire message is signed by the sender's private key.

Upon receiving the message, the receiver first verifies the signature based on the public key (the sender's key_{own}) contained in the message. It then needs to decide which registry table should be updated, since the receiver may have multiple registry tables. The receiver picks the registry table where the hth and the $(h-1)$th entry of the table matches key_{own} and key_{next}, respectively. Because of the back-traceable property, there should be at most one registry table satisfying such conditions unless the system is under dynamics. In rare cases where there are zero (or multiple) registry tables satisfying such conditions, the receiver will ignore the update (update all of them, respectively). Also notice that an adversary may easily uses its private key to generate a signature and include its own public key in the message, thus making the signature verification successful. The real authentication step in the protocol is checking whether the ith entry in some registry table indeed matches the sender's public key. For each registry table satisfying the above conditions, the receiver updates its table in the following way. It first truncates its current registry table by removing all entries after entry h, and then it appends to the table the first $w-h$ entries from the new registry table received.

Correctness argument. The above design for bypassing offline nodes enables us to bypass any number of offline

nodes, but also introduces several complications. Previously, a given registry table on a node A had only a single "writer" node (i.e., one of A's neighbors) that may send a message to A and request that A updates the registry table. Now there are potentially up to w "writers" (i.e., all w nodes upstream). This may potentially cause some fresh entries to be overwritten by stale entries. Ultimately, it may not even be obvious that the process will stabilize.

A second complication is that the deign may appear to give the adversary some extra power under node dynamics. Again let us consider the scenario in Figure 9. Previously, we explained that after D is added, M will no longer "own" the registry table entries on B and C. Thus ultimately key_1 will be overwritten. Now with the above design, it is possible for M to claim that A is offline, and keep sending registry table updates to B and C.

Interestingly, a careful argument about the above design will show that none of the above problems will occur. We will claim that if at some point the social network stops changing, then ultimately all registry table and witness table entries will stabilize. We formalize our correctness arguments as follows. If the social network stops changing at some point, then the routing tables on all honest nodes will stop changing as well. Consider the ith entry in the registry table for edge e on node A. We say that the entry is *clean* if A is honest and if we do not encounter any sybil nodes when back-tracing the random route starting from A along edge e for i hops. A clean entry is *correct* if the entry indeed contains the public key and IP address of the node that is i hops upstream from A (along edge e). Otherwise the clean entry is *incorrect*. We say an entry is *polluted* if it is not clean. We do not define "correctness" for polluted entries. We define a similar classification for witness table entries. Intuitively, a clean entry should be under the control of honest nodes, while the adversary may pollute a polluted entry in arbitrary ways. We intend to show that regardless of the system state when the social network stops changing and regardless of what actions the adversary may take, all clean entries ultimately will become correct. This will then dismiss the earlier two concerns.

Theorem 3 *If the social network stops changing at time zero, then eventually all clean registry table and witness table entries will be correct. This is true regardless of i) the contents of registry and witness tables on various nodes in the system at time zero, and ii) how the sybil nodes propagate (potentially misleading) registry and witness table updates.*

Proof: We define a *round* to be the (minimum) period of time in which each honest node in the system has propagated its registry and witness tables to all its honest neighbors at least once. We will argue that after round i, none of the first i entries in any registry table or witness table in the system is incorrect. In other words, the entry is either both clean and correct, or the entry is polluted in the first place. After round w, all clean entries will have become correct.

We prove the argument via an induction on i. We first focus on registry tables, and let us consider the first round and the first entry in any registry table. By the definition of a round, the local node must have received registry table updates from all its honest neighbors. If the entry is clean, then the entry must have been updated at some point to contain the correct information of the corresponding neighbor. Later even though other nodes (non-neighbors) may also propagate registry tables to the local node, these updates can never change the first entry in the registry table because only a direct neighbor can change the first entry. The same argument can be made for the first entry of all witness tables.

Now our induction hypothesis is that immediately after round i ($1 \leq i \leq w-1$), the first i entries in any registry table or witness table are either correct or polluted. Lemma 4 below will prove that these entries will never become incorrect during round $i+1$. In other words, the updates applied in round $i+1$ will not revert the "good" outcomes from the previous i rounds.

Next we show that after round $i+1$, the $(i+1)$th entry in any registry table or witness table is either correct or polluted. Consider the registry table for a given edge e on a given node A where the $(i+1)$th entry is clean, and suppose the w upstream nodes along edge e from A are B_1, B_2, ..., B_w. For the entry to be correct, it needs to contain the public key and IP address of B_{i+1}.

Given that the $(i+1)$th entry on A is clean, by definition, B_1, B_2, ..., B_{i+1} are all honest. Also, it implies that the first i entries on B_1 are clean (and thus correct also, from the induction hypothesis). In particular, the ith entry on B_1 must contain the public key and IP address of B_{i+1}. By definition of a round, B_1 must have propagated its registry table to A during round $i+1$, and thus A must have updated its $(i+1)$th entry to contain the public key and IP address of B_{i+1} at some point. Finally, using a similar argument as in Lemma 4, we can show that this entry will never change afterward during the remainder of round $(i+1)$.

This completes our proof by induction and after round w, all clean entries will have become correct. □

Lemma 4 *Suppose that immediately after round i ($1 \leq i < w$), none of the first i entries in any registry and witness table in the system is incorrect. Then none of these entries will become incorrect during round $i+1$.*

Proof: We prove the lemma using an induction on the first j entries. The first entry in any registry table or witness table contains direct neighbor information and can only be

affected by a node's direct neighbor (who is authenticated using their shared edge key). Thus, it will never become incorrect during round $i+1$.

Now assume that the first j ($1 \leq j \leq i-1$) entries in any registry table or witness table will never become incorrect during round $i+1$. We intend to show that the same is true for the first $j+1$ entries. We first focus on registry tables and consider the registry table for a given edge e on a given node A. Suppose the w upstream nodes along edge e from A are $B_1, B_2, \ldots, B_w$. We will argue that if the $(j+1)$th entry in A's table is correct (and thus also clean) immediately after round i, then the entry will never change during round $i+1$.

Given that the $(j+1)$th entry in A's table is clean, by definition, $B_1, B_2, \ldots, B_{j+1}$ are all honest. Also, it implies that all entries in A's table before the $(j+1)$th entry are clean as well. From the condition in the lemma, we know that all these entries are correct immediately after round i. From our induction hypothesis, all these entries will remain correct throughout round $i+1$. Thus the first j entries in A's table must contain the public keys of $B_1, B_2, \ldots, B_j$. As a result, $B_1, B_2, \ldots, B_j$ are the only nodes in the system that may propagate a registry table to A and change the $(j+1)$th entry in A's table. In particular, it is possible for a sybil node to propagate a registry table to A, but the public key of the sybil node (corresponding to the private key it uses to sign the message) will not match any of the first j entries in A's table.

Now imagine that B_h ($1 \leq h \leq j$) propagates a registry table to A. It is easy to see that the hth entry in the *witness table* on B_h for the edge between B_h and B_{h-1} is clean. Thus again from the condition in the lemma and also by the induction hypothesis, the entry remains correct through round $i+1$, and must contain the public key and IP address of A. B_h can thus not only find A but also include in the message the correct h value. Finally, when A receives the update from B_h, the $(j+1)$th entry in A's table will become the same as the $(j+1-h)$th entry in B_h's table.

On the other hand, since the $(j+1)$th entry in A's table is clean, it is easy to see that at least the first j entries in B_1's table are clean, the first $j-1$ entries in B_2's table are clean, ..., and the first entry in B_j's table is clean. Again from the condition in the lemma and also by the induction hypothesis, all these entries remain correct throughout round $i+1$. Thus the $(j+1-h)$th entry in B_h's table must be correct and contain the public key and IP address of B_{j+1}. As a result, the $(j+1)$th entry on A will remain correct even after A processes the registry table update from B_h.

The same arguments can be made for witness tables, which completes our proof by induction on j. Thus the first i entries in any registry table or witness table will never become incorrect during round $i+1$. □

6 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 [1,2] are quite small, which prevents a thorough evaluation of probabilistic guarantees. Thus we use the widely accepted Kleinberg's synthetic social network model [13] in our evaluation, which generalizes from the Watts-Strogatz model [27]. We use the model to instantiate three different graphs: a million-node graph with average node degree of 24, a 10000-node graph with average degree of 24, and a 100-node graph with average degree of 12. We will focus on the million-node graph, and present only summary results for the other two graphs. All results below are for the million-node graph unless otherwise mentioned.

6.1 Model for Social Network

Kleinberg's social network model [13] 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 being 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.

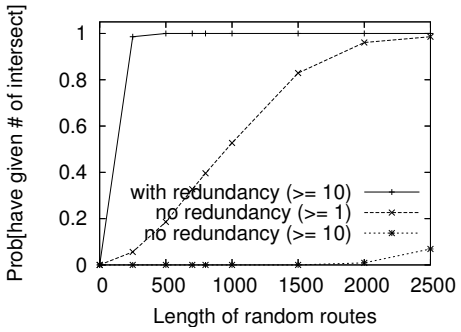

Figure 10: Probability of intersection. The legend "with redundancy" means that each node performs random routes along all directions, while "no redundancy" means performing a single random route. The legend "$(>= x)$" means that we are considering the probability of having at least x distinct intersections. SybilGuard corresponds to "with redundancy $(>= 10)$".

6.2 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 intersect; (ii) whether the random routes from the two nodes are loops (which will decrease the chance of intersection); (iii) whether there is at least one intersection node online; and (iv) whether the needed length of random routes is properly estimated.

Probability of random routes being loops. As discussed in Section 4.4, if a random route becomes a loop, then its effective length is reduced. Our simulation shows that 99.3% of the routes 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, with the redundancy technique in Section 4.4, all the nodes in our simulation have at least one route that is not a loop within their first 2500 hops. For the 10000-node topology, 99.7% of the routes do not form loops in their first 200 hops, which is above the needed route length. For the 100-node topology, 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%.

Figure 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 Figure 10 two other curves for the cases where each node performs 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 exploit redundancy, the needed length will be much larger. For our 10000-node topology, $w = 30$ yields a 99.29% probability of having at least 10 intersections. For the 100-node topology, $w = 15$ 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 4.7. 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, Figure 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 different 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

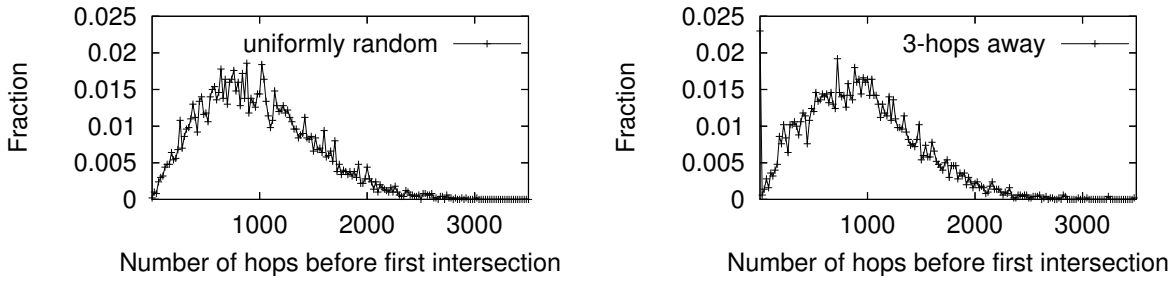

Figure 11: Probability distribution histogram for the number of hops needed before the first intersection.

samples and a worst case estimated w of 1606, Figure 10 still shows a close-to-100% intersection probability when using redundancy. 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, since 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 10000-node topology, the accurate w is 197, and the estimated w falls within 197 ± 30 after 35 samples. For the 100-node topology, the accurate w is 24, and the estimated w falls within 24 ± 7 after 40 samples.

6.3 Results with Sybil Attackers

Next we study the behavior of SybilGuard when there are malicious users. In most security research, the term "malicious user" typically refer to a single malicious user who does not assume additional identities. In this paper, however, malicious users refer to powerful attackers who have the sophistication and computation power to launch sybil attacks. For clarity, we use "sybil attackers" to refer to these users in our evaluation. Each of these sybil attackers can potentially create an *unlimited* number of "malicious users".

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 "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. The reason for better results under `cluster` is that the random routes are more likely to cross attack edges under `random`.

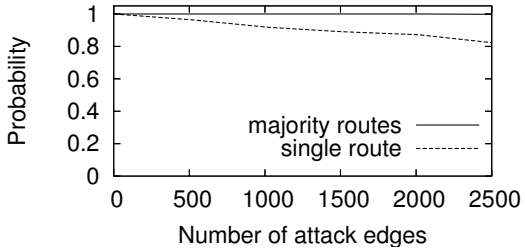

Figure 12: Probability of routes remaining entirely within the honest region.

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 all sybil nodes. As explained in Section 4.4, SybilGuard uses redundant routes and majority voting to limit the influence of such problematic routes. The curve labeled "majority routes" in Figure 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 we can see from the figure, the probability is always almost 100% before $g = 2000$, and only drops to 99.8% when $g = 2500$. This means that 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 10000-node topology and the 100-node topology, $g = 204$ and $g = 11$ will result in 0.4% and 5.1% nodes unprotected, respectively. For better understanding, Figure 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 majority voting as in Section 4.4. This means that 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 "prunes" the routes. As in Section 6.2, we assume that a "pruned" route from V accepts S if it has at least 10 distinct intersections with S's "pruned" routes. Finally, V successfully accepts S if a majority of V's routes accept S.

Figure 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 without using redundancy is much worse (even if we seek only a single intersection), demonstrating that exploiting redundancy is necessary. For our 10000-node topology and 100-node topology, $g = 204$ 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 given a verifier, 12.3% of the nodes will not be accepted by that verifier. Each honest node, on average, should still be accepted by 87.7% 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 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

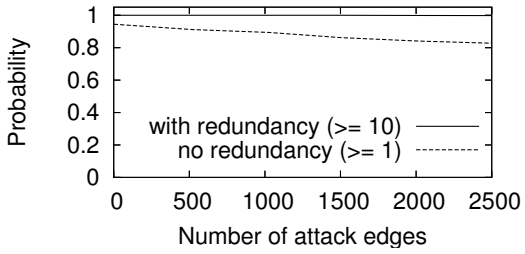

Figure 13: Probability of an honest node accepting another honest node (i.e., having at least a target number of intersections). The legends are the same as in Figure 10, and SybilGuard corresponds to "with redundancy ($>= 10$)".

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 roughly increases 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 our 10000-node topology and 100-node topology, the fraction of bad samples is always below 20% when $g \leq 204$ and $g \leq 11$, respectively.

7 Related Work

The sybil attack [10] 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 1, the first investigation [10] into sybil attacks already proved a series of negative results.

Bazzi and Konjevod [4] proposed using network coordinates [19] to foil sybil attacks, and a similar idea has also been explored for sensor networks [23]. 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 that 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.* [9] proposed a scheme for making DHT lookups more resilient to sybil attacks. The scheme lever-

ages 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, which helps to achieve much stronger properties. First, SybilGuard is able 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 results in [9] deteriorate as the adversary creates more and more sybil nodes. Second, SybilGuard guarantees roughly $\sqrt{n}$ equivalence groups to ensure sufficient diversity. A bootstrap tree can be in any shape and thus the number of equivalence groups can be rather small. Third, the sizes of different equivalence groups in SybilGuard are roughly the same. In the bootstrap tree approach the sizes can be quite different, which can lead to significant load imbalance. Finally, 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 [18]. 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* [20], 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 [20], 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 [8, 11, 22] against such attacks aim at preventing the sybil nodes from boosting a malicious user's rating (and attracting buyers, in the case of eBay). They cannot and do not aim to control the number or size of sybil groups. All the sybil nodes are able to obtain the same rating/reputation as the malicious user. Thus the sybil attack problem in reputation systems is fundamentally different from the one solved by SybilGuard.

In some other reputation systems such as Credence [26], 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 [8]), and thus Credence relies on a central authority to limit sybil nodes [26].

Trust networks and random walks. The social network in SybilGuard is one kind of trust network. Many previous works [8, 11, 26] 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 [14], 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.

Trust propagation or transitive trust is a technique that researchers often use on trust networks [8, 11, 22, 26]. SybilGuard is more related to exploiting graph properties rather than trust propagation. Random walks have also been used to infer worm origin [28] by identifying nodes with a small number of incoming messages but with a large number of outgoing flows. Such techniques are not, however, applicable or related to sybil attacks.

8 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. Currently we are working on obtaining real social network data to further validate SybilGuard.

9 Acknowledgments

We thank David Andersen, Michael Freedman, Petros Maniatis, Adrian Perrig, Srinivasan Seshan, and the anonymous reviewers for many helpful comments on the paper.

References

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

[2] International Network for Social Network Analysis, 2006. http://www.insna.org/INSNA/data_inf.htm.

[3] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. In *DISC*, 2003.

[4] R. Bazzi and G. Konjevod. On the establishment of distinct identities in overlay networks. In *ACM PODC*, 2005.

[5] B. Bollobás. Martingales, Isoperimetric Inequalities and Random Graphs. In A. Hajnal, L. Lovász, and V. T. Sós, editors, *Combinatorics*, number 52 in Colloq. Math. Soc. János Bolyai, pages 113–139. North Holland, 1988.

[6] 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 *ACM SIGMETRICS*, 2000.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *IEEE INFOCOM*, 2005.

[8] A. Cheng and E. Friedman. Sybilproof reputation mechanisms. In *ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems*, 2005.

[9] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson. Sybil-resistant DHT routing. In *European Symposium On Research In Computer Security*, 2005.

[10] J. Douceur. The Sybil attack. In *IPTPS*, 2002.

[11] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *ACM Electronic Commerce*, 2004.

[12] A. Flaxman. Expansion and lack thereof in randomly perturbed graphs. Technical Report MSR-TR-2006-118, Microsoft Research, August 2006. Also available at ftp://ftp.research.microsoft.com/pub/tr/TR-2006-118.pdf.

[13] J. Kleinberg. The small-world phenomenon: An algorithm perspective. In *STOC*, 2000.

[14] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, and M. Baker. The LOCKSS peer-to-peer digital preservation system. *ACM TOCS*, 23(1), 2005.

[15] C. McDiarmid. On the Method of Bounded Differences. In *London Mathematical Society Lecture Note Series*, volume 141, pages 148–188. Cambridge University Press, 1989.

[16] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[17] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. Marsh. Efficient lookup on unstructured topologies. In *ACM PODC*, 2005.

[18] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil attack in sensor networks: Analysis & defenses. In *ACM/IEEE IPSN*, 2004.

[19] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *IEEE INFOCOM*, 2002.

[20] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *IEEE Symposium on Security and Privacy*, 2005.

[21] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *ACM SIGCOMM*, 2006.

[22] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *International Semantic Web Conference*, 2003.

[23] N. Sastry, U. Shankar, and D. Wagner. Secure verification of location claims. In *ACM Workshop on Wireless Security*, 2003.

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

[25] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Telling humans and computers apart. In *Eurocrypt*, 2003.

[26] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *USENIX NSDI*, 2006.

[27] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684), 1998.

[28] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhang. Worm origin identification using random moonwalks. In *IEEE Symposium on Security and Privacy*, 2005.

A Reversed Birthday Paradox Proof

Section 4.7 described how we use sampling to determine the required length of random routes before two random routes (starting from two uniformly random nodes) will intersect with good probability (e.g., with 95% probability). A naive approach would take a certain number of samples, record the route length before intersection for each sample, and finally pick the 95 percentile length as the estimate. Unfortunately, the estimate obtained in such approach can easily be manipulated by sybil nodes. As long as over 5% of the samples involve routes entering into the sybil region, the sybil nodes can manipulate the 95 percentile value. For this reason, SybilGuard uses the median (i.e., 50 percentile) and then multiplies the median by some multiplier to obtain the 95 percentile.

The critical property we intend to prove here is that the multiplier is a constant that does not change with n. In other words, the multiplier is a fundamental constant in the Birthday Paradox distribution. Following is our theorem showing that the multiplier should be approximately 2.1 if we use the median to obtain the 95 percentile:

Theorem 5 *There are n numbered balls in a bag, n is unknown. There are two players, A and B. In each step, A and B each (sequentially) take out a ball (with replacement) and record the number. Let T denote the number of steps until the set of numbers that A has recorded intersects with the set of numbers that B has recorded (the "steps needed").*

For any constants $\alpha, \epsilon > 0$, and for any sufficiently large n,

$$\Pr[T \geq \alpha\sqrt{n}] = e^{-(1\pm\epsilon)\alpha^2}$$

Let α_x be the value of α for which $\Pr[T \geq \alpha\sqrt{n}] = x$, and then we have

$$\alpha_p/\alpha_q = (1+\epsilon)\sqrt{\ln(1/p)}/\sqrt{\ln(1/q)}.$$

So, in particular, we have

$$\alpha_{.05}/\alpha_{.5} \approx 2.07892.$$

Proof: Consider the game at step $t = \alpha\sqrt{n}$, and let S denote the (random) set belonging to player A at step t.

Conditional on S, the probability that $T > t$ is the probability that B never drew a number in S. Since $|S| \leq t = \alpha\sqrt{n}$, this probability is given exactly by

$$\Pr\left[T > t \;\middle|\; S\right] = \left(1 - \frac{|S|}{n}\right)^t = e^{t\ln(1-|S|/n)} = e^{t\left(-|S|/n - (|S|/n)^2/2 - (|S|/n)^3/3 - \ldots\right)} = e^{-|S|t/n - \mathcal{O}(n^{-1/2})}.$$

A simple way to calculate the expected size of S at step t is to express $|S|$ as a sum of indicator variables for events of the form $i \in S$, where $i = 1, \ldots, n$. Then

$$\mathrm{E}[|S|] = n(1 - \Pr[i \notin S]) = n\left(1 - \left(1 - \frac{1}{n}\right)^t\right) = n\left(1 - \sum_{k=0}^{t}\binom{t}{k}\left(-\frac{1}{n}\right)^k\right) = t - \frac{t(t-1)}{2n} + \mathcal{O}(n^{-1/2}).$$

Then, by applying the Azuma-Hoeffding tail inequality for martingales, as described by McDiarmid [15] or Bollobas [5],

$$\Pr[\big||S| - t\big| \geq \epsilon t] \leq 2e^{-\epsilon^2 t/2},$$

and so,

$$\begin{aligned}
\Pr\left[T > t\right] &= \mathrm{E}[e^{-|S|t/n + \mathcal{O}(n^{-1/2})}] \\
&= \mathrm{E}\left[e^{-|S|t/n+\mathcal{O}(n^{-1/2})} \;\middle|\; \big||S| - t\big| < \epsilon t\right] \Pr\left[\big||S| - t\big| < \epsilon t\right] \\
&\quad + \mathrm{E}\left[e^{-|S|t/n+\mathcal{O}(n^{-1/2})} \;\middle|\; \big||S| - t\big| \geq \epsilon t\right] \Pr\left[\big||S| - t\big| \geq \epsilon t\right] \\
&= o(1) + e^{-(1\pm\epsilon)\alpha^2/n}(1 - o(1)).
\end{aligned}$$

So, for any sufficiently large value of n,

$$e^{-(1+2\epsilon)\alpha^2/n} \leq \Pr\left[T > t\right] \leq e^{-(1-2\epsilon)\alpha^2/n}.$$

□

B Mixing Time of Social Networks

SybilGuard leverages the fast mixing property of social networks. Based on synthetic social network models, this section summarizes previously known theoretical results for such fast mixing property and also presents some additional analytical proofs.

Some of these results are based on the widely accepted Kleinberg's synthetic social network model [13] described in Section 6.1. Section 6.1 explained that the model uses a two-dimensional grid as the base structure. The general model [13] actually allows the use of a one-dimensional ring as the base structure as well. In such case, the grid distance in the model becomes the ring distance (i.e., the minimum number of hops needed to go from one node along the ring edges to the other). All other aspects of the construction remain the same. Using a ring as the base structure is perhaps simpler but less realistic than using a grid.

Previously, Boyd et al. [7] proved that Kleinberg's model is fast mixing when the base structure is a grid and $r = 0$. Remember from Section 6.1 that r is a tunable parameter in the model controlling the distribution of remote friends. In the remainder of this section, we will prove that Kleinberg's model is fast mixing when the base structure is a ring and $0 \leq r < 1$. After the proof in this paper was constructed, Flaxman [12] later proves that a much wider range of topologies are fast mixing. In particular, these fast mixing topologies include Kleinberg's model when i) the base structure is a ring and $0 \leq r < 1$, and ii) the base structure is a grid and $0 \leq r < 2$. In other words, the results in [12] are a superset of the results from [7] and the results from the remainder of this section. However, to make this paper self-contained, we still present our results below for Kleinberg's model when the base structure is a ring and $0 \leq r < 1$.

B.1 Notations

As in Kleinberg's model [13], we consider random graphs $G = (V, E)$ that are generated by starting with a base graph $\bar{G}$, and adding q edges out of every vertex independently at random, where the i-th edge out of vertex v is denoted by $e_{v,i}$ and is chosen according to the distribution

$$\Pr[e_{v,i} = vw] = \frac{d_{\bar{G}}(v,w)^{-r}}{\sum_{u \neq v} d_{\bar{G}}(v,u)^{-r}} \qquad \text{for all } w \neq v.$$

Undirected edges are sets of 2 vertices, but edge $\{u, v\}$ will be abbreviated as uv when it is not confusing to do so. For any graph H let $E(H)$ denote the edge set of H, let $V(H)$ denote the vertex set of H, and for sets $S, T \subseteq V(H)$, let $e_H(S,T)$ denote the number of edges between S and T in H, and for vertices $u, v \in V(H)$ let $d_H(u,v)$ denote to the distance (in edges) of the shortest path between u and v in H. Let $\deg_H(v)$ denote the degree of v in H. The subscripts for $e(S,T)$, $d(u,v)$, and $\deg(v)$ will be omitted when referring the graph G if it is not too confusing to do so. Let $\mathcal{SW}(\bar{G}, r, q)$ denote the distribution of random graphs with base graph $\bar{G}$ and parameters r and q. We consider the special case where the base graph $\bar{G}$ is a cycle on n vertices (denoted as C_n), r is a constant with $0 \leq r < 1$, and q is a constant.

B.2 Main Theorem

We intend to establish the following main theorem:

Theorem 6 *For $\bar{G} = C_n$ and constant r and q, with $0 \leq r < 1$, and q being any positive integer,* **whp** *$G \sim \mathcal{SW}(\bar{G}, r, q)$ has mixing time $\mathcal{O}(\log n)$.*

The theorem is proved via a bound on the *conductance* of the graph. The conductance of $G = (V, E)$ is defined by

$$\Phi = \min_{S \subseteq V \,:\, 2e(S) + e(S,\bar{S}) \leq |E|} \frac{e(S,\bar{S})}{2e(S) + e(S,\bar{S})}.$$

Φ provides a bound on the spectral gap of G,

$$\lambda \geq \Phi^2/2,$$

and λ gives a bound on the mixing time of G

$$\tau_2(\epsilon) \leq \frac{1}{\lambda}\left(\frac{1}{2}\log\frac{1-\pi_\star}{\pi_\star} + \log\frac{1}{\epsilon}\right), \tag{1}$$

where $\pi_\star = \min_{v \in V} \frac{\deg(v)}{2|E|}$.

With these inequalities at hand, in order to show that **whp** G is rapidly mixing, it is sufficient to show only that **whp** the conductance of G is bounded below by a constant.

Lemma 7 *For $\bar{G} = C_n$ and constant r and q, with $0 \leq r < 1$, and q being a positive integer, there exists $\epsilon = \epsilon(q, r)$ which is independent of n such that* **whp** *there does not exist $S \subseteq V$ with $2e(S) + e(S, \bar{S}) \leq |E|$ and $e(S, \bar{S}) \leq \epsilon q |S|$.*

Theorem 6 follows quickly from Lemma 7.

Proof of Theorem 6: Because of the way $G \sim \mathcal{SW}(\bar{G}, q, r)$ is generated, any $S \subseteq V$ has $e(S) \leq (q+1)|S|$. So

$$\begin{aligned}
\Phi &= \min_{S \subseteq V \,:\, 2e(S)+e(S,\bar{S}) \leq |E|} \frac{e(S, \bar{S})}{2e(S) + e(S, \bar{S})} \\
&\geq \min_{S \subseteq V \,:\, 2e(S)+e(S,\bar{S}) \leq |E|} \frac{e(S, \bar{S})}{2(q+1)|S| + e(S, \bar{S})} \\
&= \min_{S \subseteq V \,:\, 2e(S)+e(S,\bar{S}) \leq |E|} \frac{1}{\frac{2(q+1)|S|}{e(S,\bar{S})} + 1}.
\end{aligned}$$

Lemma 7 shows that **whp** the bound $e(S, \bar{S}) \geq \epsilon q |S| \geq \frac{1}{2}\epsilon(q+1)|S|$ holds for all sets considered in the minimum, so **whp**

$$\Phi \geq \frac{1}{\frac{2(q+1)}{\epsilon(q+1)/2} + 1} = \frac{\epsilon}{4 + \epsilon}.$$

Since Φ is at least a constant, λ is also at least a constant. Every $v \in G$ has $\deg(v) > q + 1$, so $\pi_\star > 1/(2n)$, and it follows from (1) that $\tau_2(\epsilon) \leq \mathcal{O}(\log n)$. □

The proof of Lemma 7 is an application of the first moment method, and relies on a moderately precise calculation of the expected number of sets S which violate the bound $e(S, \bar{S}) \leq \epsilon |S|$. This is achieved by considering separately the sets with $|S| \leq \delta n$ and $|S| > \delta n$ for an appropriately chosen constant δ.

The proof of Lemma 7 also requires the following lemma, which gives an upper bound on the probability that random edge $e_{v,i}$ connects v to a vertex in a particular set of size s.

Lemma 8 *Let $\bar{G} = C_n$ with n sufficiently large and let r and q be constants, with $0 \leq r < 1$, and q a positive integer. Then for any $S \subseteq V$ with $|S| = s$,*

$$\Pr[e_{v,i} \sim S] \leq \left(\frac{s}{n}\right)^{1-r}.$$

The proof of Lemma 8 is left for after the proof of Lemma 7, which is given now.

Proof of Lemma 7: A straightforward way to obtain an upper bound on the probability that there exists a set $S \subseteq V$ with $2e(S) + e(S, \bar{S}) \leq |E|$ and $e(S, \bar{S}) \leq \epsilon q |S|$ is as follows. Let Z_S be an indicator random variable for the event that a particular set S satisfies these conditions, and calculate an upper bound on the expected value of the sum $Z = \sum_{S \subseteq V} Z_S$. Showing the expected value tends to 0 with n yields a bound which proves the lemma, because for any non-negative random variable, $\Pr[Z \geq 1] \leq \mathrm{E}[Z]$.

Part i **($Z' = \sum_{|S| \geq \frac{3}{4}n} Z_S$):** To begin, consider the contribution of Z_S for sets with $|S| \geq \frac{3}{4}n$. It is very unlikely that such a set will have $2e(S) + e(S, \bar{S}) \leq |E|$, so these sets will not add much to the expected value of Z. To make this observation formal, calculate that

$$\mathrm{E}[2e(S) + e(S, \bar{S})] = \sum_{v \in S} \mathrm{E}[\deg(v)] = |S|2(q+1) \geq \frac{6}{4}(q+1)n.$$

So, by the Azuma-Hoeffding inequality for martingales (as formulated by McDiarmid [15] and also by Bollobas [5]),

$$\Pr[2e(S) + e(S, \bar{S}) \leq (q+1)n] \leq e^{-2\left(\frac{1}{2}(q+1)n\right)^2/(qn)} \leq e^{-(q+1)n/2} \leq e^{-n}.$$

Therefore

$$\mathrm{E}[Z'] = \sum_{S\colon |S|\geq \frac{3}{4}n} \mathrm{E}[Z_S] \leq n2^n e^{-n} = o(1).$$

Part ii ($Z'' = \sum_{\delta n \leq |S| \leq \frac{3}{4}n} Z_S$)**:** When dealing with a set S that has $s = |S| \leq \frac{3}{4}n$, the bound on $\mathrm{E}[Z_S]$ will come from the unlikeliness of $e(S, \bar{S})$ being less than ϵqs. Making this calculation precise enough requires an examination of the structure of the set S in the base graph $\bar{G}$. Consider the connected components of graph induced by S in $\bar{G}$, which for this lemma is the cycle C_n. Each component consists of a contiguous block of vertices in the cycle. If S induces k such blocks in $\bar{G}$, then there are $2k$ edges of $\bar{G}$ in $e(S, \bar{S})$.

Since every vertex in S chooses q random neighbors, the value of $e(S, \bar{S})$ is at least $2k$ plus the number of times the qs neighbor choices pick something outside of S. Lemma 8 shows that the probability of picking a neighbor inside of S is at most $\left(\frac{s}{n}\right)^{1-r}$, so, letting $R = G - \bar{G}$ denote the graph of random edges added to $\bar{G}$ to form G, the expected number of random neighbors chosen which lie inside S is at most

$$\mathrm{E}[e_R(S)] \leq qs\left(\frac{s}{n}\right)^{1-r}.$$

For $e(S, \bar{S})$ to be less than ϵqs, it must hold that $e_R(S) \geq (1-\epsilon)qs + 2k$. A bound for this, given by the Azuma-Hoeffding inequality for martingales, is

$$\Pr[e_R(S) \geq (1-\epsilon)qs + 2k] \leq e^{-2qs\left(1-\epsilon-\left(\frac{s}{n}\right)^{1-r}\right)^2}.$$

Also, if $2k > \epsilon qs$, then there is probability 0 that the set is bad.

To find an upper bound on the contribution of sets of this type to $\mathrm{E}[Z]$, note that any set S consisting of k contiguous blocks can be specified by $2k$ start/stop points and a single bit to indicate if the first point is a start or a stop. Therefore there are at most $2\binom{n}{2k}$ sets of size s consisting of k contiguous blocks, so

$$\begin{aligned}
\mathrm{E}[Z''] &= \sum_{S\colon \delta n \leq |S| \leq \frac{3}{4}n} \mathrm{E}[Z_S] \\
&\leq \sum_{s=\delta n}^{\frac{3}{4}n} \sum_{k=1}^{\epsilon qs/2} 2\binom{n}{2k} e^{-2qs\left(1-\epsilon-\left(\frac{s}{n}\right)^{1-r}\right)^2} \\
&\leq \sum_{s=\delta n}^{\frac{3}{4}n} (\epsilon qs)\left(\frac{ne}{\epsilon qs}\right)^{\epsilon qs} e^{-2qs\left(1-\epsilon-\left(\frac{3}{4}\right)^{1-r}\right)^2} \\
&\leq (qn) \sum_{s=\delta n}^{\frac{3}{4}n} \left[\left(\frac{e}{\epsilon q\delta}\right)^{\epsilon} e^{-2\left(1-\epsilon-\left(\frac{3}{4}\right)^{1-r}\right)^2}\right]^{qs}.
\end{aligned}$$

For any δ with

$$\delta > \left(\frac{\epsilon}{eq}\right) \exp\left\{-2\epsilon^{-1}\left(1-\epsilon-\left(\frac{3}{4}\right)^{1-r}\right)^2\right\}, \tag{2}$$

this sum tends to 0, and, for $\epsilon < 1 - \left(\frac{3}{4}\right)^{1-r}$, any δ with $\delta \geq \epsilon$ is sufficient.

Part iii ($Z''' = \sum_{|S| \leq \delta n} Z_S$)**:** The upper bound on the expected contribution of the sets smaller than size δn is calculated in a manner quite similar to the bound for sets with size between δn and $\frac{3}{4}n$. Again it is necessary to consider the structure of the graph induced by S in $\bar{G}$, and to note that if this induced graph has k components then for $e(S, \bar{S})$ to be less than ϵqs, then the random edges which remain in S must satisfy $e_R(S) \geq (1-\epsilon)qs + 2k$. However, for $s \leq \delta n$, instead of using an exponential concentration inequality to bound the probability of $e_R(S)$ being this large, it is sufficient to use a more elementary inequality (which follows from selecting the $(1-\epsilon)qs + 2k$ random edges which should not leave S and applying

Lemma 8 to each of them):

$$\begin{aligned}\Pr[e_R(S) \geq (1-\epsilon)qs + 2k] &\leq \binom{qs}{(1-\epsilon)qs+2k}\left[\left(\frac{s}{n}\right)^{1-r}\right]^{(1-\epsilon)qs+2k} \\ &= \binom{qs}{\epsilon qs - 2k}\left[\left(\frac{s}{n}\right)^{1-r}\right]^{(1-\epsilon)qs+2k} \\ &\leq \left[\left(\frac{e}{\epsilon}\right)^{\epsilon}\left(\frac{s}{n}\right)^{(1-\epsilon)(1-r)}\right]^{qs}.\end{aligned}$$

As in the previous case, if $2k > \epsilon qs$, then $\Pr[e_R(S) \geq (1-\epsilon)qs + 2k] = 0$.

It follows from this upper bound on $\Pr[e_R(S) \geq (1-\epsilon)qs + 2k]$ that

$$\begin{aligned}\mathrm{E}[Z'''] &= \sum_{S:\ |S|\leq\delta n} \mathrm{E}[Z_S] \\ &\leq \sum_{s=1}^{\delta n}\sum_{k=1}^{\epsilon qs/2} 2\binom{n}{2k}\left[\left(\frac{e}{\epsilon}\right)^{\epsilon}\left(\frac{s}{n}\right)^{(1-\epsilon)(1-r)}\right]^{qs} \\ &\leq \sum_{s=1}^{\delta n}(\epsilon qs)\binom{n}{\epsilon qs}\left[\left(\frac{e}{\epsilon}\right)^{\epsilon}\left(\frac{s}{n}\right)^{(1-\epsilon)(1-r)}\right]^{qs} \\ &\leq \sum_{s=1}^{\delta n}(\epsilon qs)\left(\frac{ne}{\epsilon qs}\right)^{\epsilon qs}\left[\left(\frac{e}{\epsilon}\right)^{\epsilon}\left(\frac{s}{n}\right)^{(1-\epsilon)(1-r)}\right]^{qs} \\ &= \sum_{s=1}^{\delta n}(\epsilon qs)\left[\left(\frac{e^2}{q\epsilon^2}\right)^{\epsilon}\left(\frac{s}{n}\right)^{(1-\epsilon)(1-r)-\epsilon}\right]^{qs} \\ &\leq (\epsilon q)\sum_{s=1}^{\delta n} s\left[\left(\frac{e^2}{q\epsilon^2}\right)^{\epsilon}\delta^{(1-\epsilon)(1-r)-\epsilon}\right]^{qs}.\end{aligned}$$

For any δ with

$$\delta < \left(\frac{q\epsilon^2}{e^2}\right)^{\epsilon/((1-\epsilon)(1-r)-\epsilon)} \tag{3}$$

this sum tends to 0, and for $\epsilon < \frac{1-r}{2-r}$ it follows that $(1-\epsilon)(1-r)-\epsilon \in (0,1)$, and so $\delta < \left(\frac{q\epsilon^2}{e^2}\right)^{\epsilon}$ is sufficient.

Since $\mathrm{E}[Z] = \mathrm{E}[Z'] + \mathrm{E}[Z''] + \mathrm{E}[Z''']$, to complete the proof it only remains to check that a value of δ exists which satisfies both (2) and (3). And for $\epsilon \leq \min\left\{1 - \left(\frac{3}{4}\right)^{1-r}, \frac{1-r}{2(2-r)}\right\}$, to show a value of δ exists, it is sufficient to verify that for ϵ sufficiently small,

$$\epsilon < \left(\frac{q\epsilon^2}{e^2}\right)^{\epsilon}.$$

Since $\lim_{\epsilon\to 0^+}\left(\frac{q\epsilon^2}{e^2}\right)^{\epsilon} = 1$, this must be the case. □

Proof of Lemma 8: A shifting argument shows that when $\bar{G} = C_n$, the sum

$$\sum_{w\in S} d_{\bar{G}}(v,w)^{-r}$$

is minimized by the contiguous set of vertices centered at v.

Thus,

$$\begin{aligned}
\Pr[e_{v,i} \sim S] &\leq \frac{2\sum_{j=1}^{s/2} j^{-r}}{2\sum_{j=1}^{n/2} j^{-r}} \\
&\leq \frac{1 + \int_1^{s/2} x^{-r} dx}{\int_1^{n/2} x^{-r} dx} \\
&= \frac{(s/2)^{1-r} - r}{(n/2)^{1-r} - 1} \\
&\leq \left(\frac{s}{n}\right)^{1-r} \qquad \text{when } n \text{ is sufficiently large.}
\end{aligned}$$

□