SybilControl: Practical Sybil Defense with Computational Puzzles

Frank Li[1] Prateek Mittal[2] Matthew Caesar[3] Nikita Borisov[3]

[1]Massachusetts Institute of Technology
frankli@mit.edu

[2]University of California, Berkeley
pmittal@eecs.berkeley.edu

[3]University of Illinois, Urbana-Champaign
{caesar,nikita}@illinois.edu

ABSTRACT

Many distributed systems are subject to the *Sybil attack*, where an adversary subverts system operation by emulating the behavior of multiple distinct nodes. Most recent works addressing this problem leverage social networks to establish trust relationships between users. However, social networks are not appropriate in all systems. They can be subverted by social engineering techniques, require nodes to maintain and be aware of social network information, and may require overly optimistic assumptions about the fast-mixing nature of social links.

This paper explores an alternate approach. We present SybilControl, a novel decentralized scheme for controlling the extent of Sybil attacks. It is an admission and retainment control scheme for nodes in a distributed system that requires them to periodically solve computational puzzles. SybilControl consists of a distributed protocol to allow nodes to collectively verify the computational work of other nodes, and mechanisms to prevent the malicious influence of misbehaving nodes that do not perform the computational work. We investigate the practical issues involved with deploying SybilControl into existing DHTs, particularly with handling churn. SybilControl is shown to provide strict bounds on the size of Sybil attacks, given adversaries with finite resources. We also show through simulations that the performance overhead of enabling SybilControl is manageable using commonplace DHT churn-handling techniques. This provides strong evidence that SybilControl can be practically deployed.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Design, Security

Keywords

Sybil Attack, Distributed Systems, Computational Puzzles

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STC'12, October 15, 2012, Raleigh, North Carolina, USA.
Copyright 2012 ACM 978-1-4503-1662-0/12/10 ...$15.00.

1. INTRODUCTION

Decentralized distributed systems, such as peer-to-peer networks, are a basis for scalable computing. However, many systems are particularly susceptible to the *Sybil attack* [16]. In this attack, an adversary exploits the low cost of entry into a system by obtaining multiple identities, thus taking the guise of many distinct nodes. These *Sybil nodes* can then collude to launch further attacks, for example by taking over resources and disrupting connectivity, to subvert the system's operation. Researchers have documented this vulnerability in real-world systems, including the Maze peer-to-peer file-sharing system [23, 40] and Vanish [19, 39], a system for self-destructing data that relies on a distributed hash table (DHT). Also, Sybil attacks can be launched on the Tor network [14], where an adversary inserts a large number of bad relays.

Addressing this vulnerability is important for establishing trust on such systems. Recent research has focused on leveraging information from social networks to protect honest users, resulting in several state-of-the-art approaches [9, 12, 22, 24, 25, 35, 38, 41]. The key observation in these approaches is that it is expensive for a malicious adversary to establish social links with honest users, and hence social network graphs can be used to detect and mitigate Sybil attacks. If the number of social links between the adversary and honest users is g, these defenses bound the number of Sybils to a function of g. For example, SybilLimit [41] guarantees a bound of $O(g \lg n)$ Sybil identities, where n is the number of honest nodes in the system.

However, the reliance of these techniques on social networks has disadvantages. The use of a social network is not appropriate in many systems, as social relationships require nodes to be aware of their social contacts and users to exert manual effort to set up and maintain accurate relationships. Also, a recent study indicates that social network-based approaches do not perform as well in real-world systems [36]. These approaches are dependent on the assumption that the number of connections between Sybil nodes and honest nodes is small relative to the number of honest nodes in the network. This assumption is not necessarily valid, as it requires the social networks to be *fast-mixing*, meaning a random walk on the graph approaches the stationary distribution relatively quickly. Recent investigations [26] into the mixing times of real-world social networks have found that they may not be as fast as what the approaches assumed. Furthermore, studies [6, 8, 36] indicate social networks are prone to social engineering attacks (for example, some Facebook users agree to many friend requests regardless of whether the other party is known to them). This can drastically inflate the number of relationships between Sybil and honest nodes. These disadvantages lead to questions about the practicality of social network methods in real-world contexts.

To address this, we present SybilControl, a novel admission and retainment control scheme that limits the extent of Sybil attacks. If an adversary with *finite resources* must dedicate some resources to support each node in a system, then it can only support a limited number of adversarial nodes. SybilControl enforces that nodes dedicate computational resources to support their system identities. Thus, we differ from social network approaches in both our assumptions and provided bounds. We assume a computationally-bounded adversary, instead of making assumptions on the trust relationships between nodes. Our bounds, as we will show, limit the proportion of Sybils in the network to equal the proportion of resources the adversary controls, instead of depending on the number of social connections between Sybils and honest nodes. SybilControl does not limit the adversary's ability to position nodes in the network, as other schemes to address this issue have been developed [32] and can be used orthogonally with SybilControl.

SybilControl's enforcement is conducted through two main components: a distributed protocol to allow nodes to collectively verify computational work of other nodes, and defense mechanisms that protect honest nodes from the influences of misbehaving nodes that do not perform computational work. In particular, a SybilControl-enabled node can distribute challenges to its neighbors, and can verify that those neighbors recently used the challenges to solve computational puzzles. Computational puzzles [18, 21] can be created to require significant amounts of computation to solve but are easy to verify. Challenges are propagated throughout the network in a scalable fashion using aggregation, enabling nodes to verify other nodes that are not their adjacent neighbors. To provide the property that the number of identifiers owned by a user remains commensurate with the computational power of the user, SybilControl nodes periodically re-issue fresh challenges, forcing other nodes to periodically solve new puzzles to remain in the system. Defense mechanisms protect honest nodes from Sybils that do not perform computational work, preventing those Sybils from subverting the system.

SybilControl can be applied to a wide variety of distributed systems to allow for scalable trusted computing. To investigate performance issues, we focus on distributed hash tables, or DHTs, and deploy it in the context of the Chord DHT [34] for concreteness. DHTs enable a number of scalable systems such as peer-to-peer networks, distributed file systems, content distribution networks, and decentralized recommendation systems, among others. Use of SybilControl protocols can exacerbate the effects of churn, or the dynamic arrival and departure of nodes in the network, in DHTs. However, we evaluate a SybilControl-enabled version of Chord through simulations, and find that when using common DHT churn-handling techniques, the performance overhead of SybilControl is minimized. Additionally, the security bounds provided by SybilControl are analysed, showing a strict bound on the size of Sybil attacks given an adversary with finite resources. This bound's effectiveness is quantified through a basic cost analysis. These evaluations provide strong evidence of SybilControl's practicality.

The remainder of the paper is structured as follows: In Section 2, we discuss background and related work in Sybil defense. In Section 3, we provide an overview of our approach, followed by the design details of SybilControl in Section 4. Section 5 documents the deployment of SybilControl, specifically for DHTs. The security and cost analysis, performance and practicality of SybilControl is evaluated in Section 6, followed by concluding remarks in Section 7.

2. BACKGROUND AND RELATED WORK

2.1 Distributed Hash Tables

Distributed hash tables (DHTs), such as Chord [34], Pastry [31], CAN [29], and Tapestry [43], are decentralized systems for storing key-value pairs where keys are numerical values within a certain range (the keyspace). The primary operation is the lookup: given a key, a node in the system can efficiently retrieve the associated value. DHTs form the infrastructure for a variety of scalable systems such as peer-to-peer networks, distributed file systems, anonymity networks, and decentralized file-sharing systems.

In a DHT, nodes are assigned unique numerical identifiers. A keyspace partitioning scheme divides the ownership of the keyspace amongst the nodes, and key-value pairs are stored at the node who owns the key. For example, in Chord, node IDs are drawn from the keyspace, and the partitioning is dependent on the node's ID. If a node has ID i and the node with the next higher ID has j, node j owns the keyspace between i and j.

Nodes also maintain a set of *neighbor* links to other nodes, forming an overlay network. The DHT lookup protocol utilizes this overlay network to conduct lookups and routing. In Chord, a node keeps track of $O(\lg n)$ other nodes (neighbors), where n is the number of nodes in the system, in data structures called the *successor table* and the *finger table*. A node's successor table tracks the nodes with the next $\lg n$ higher IDs (successors), and the finger table tracks $\lg n$ strategically chosen nodes (fingers). This allows the Chord lookup protocol to conduct a lookup in $O(\lg n)$ hops. Because DHTs experience *network churn*, or the dynamic arrival and departure of nodes in the network, churn-handling protocols dynamically maintain neighbor relationships by discovering new potential neighbors and periodically inspecting existing neighbor links to determine liveness.

2.2 Sybil Defense Schemes

After the initial classification of the Sybil attack [16], several centralized Sybil defense schemes [10, 30] were proposed that focused on using a central authority to certify nodes for admission control and limit the number of Sybil identities. However, the centralized nature of these schemes conflict with many of the guiding principles of distributed systems, and a central authority creates trust issues, a single point of failure, and a possible performance bottleneck.

In an alternate direction, several proposals have focused on decentralized schemes that work better in conjunction with distributed systems. One approach [11] is to utilize the bootstrap graph of DHTs. The idea is that a small number of malicious users would introduce, or bootstrap, many Sybil nodes into the system. These nodes would be highly connected in the bootstrap graph. However, it is possible that honest nodes might also bootstrap many other honest nodes, making it difficult to accurately distinguish honest from malicious users. A distributed registration system was introduced in [13] that regulated the number of identities a specific IP address could obtain. However, it is possible for adversaries to control multiple IP addresses through spoofing or prefix hijacking attacks. Additionally, multiple honest users could share the same IP address if they are behind a NAT, which makes this approach less practical.

The most recent schemes have focused on exploiting social network information. X-Vine [24], Gatekeeper [35], SybilDefender [38], SybilLimit [41], SybilGuard [42], and Whanau [22] all provide protocols to defend against Sybil attacks based on random walks over a social network. SybilInfer [12] uses Bayesian inference on information gathered from the social network to identify

Sybils. The LC Model approach [9] applies machine-learning algorithms to social networks for Sybil detection. However, all of these defenses assume that social networks are "fast-mixing", which a recent study [26] has shown is not true in real-world networks to the extent these schemes assume. This leaves open questions about the strengths of the security guarantees these mechanisms provide and their true real-world effectiveness. Furthermore, use of social networks may not be appropriate in all systems since nodes must then be aware of their social contacts. Social relationships require manual effort from users to instrument and correctly maintain.

The concept of using computational puzzles for Sybil defense is not unique to SybilControl. A similar approach by Borisov [7] also proposed challenge distribution and puzzle verification in overlay networks, and inspired several techniques in our scheme. However, the design was slightly flawed, which we discuss in detail and compare with SybilControl in Section 4.1.4. Additionally, the proposal was purely conceptual, without any implementation, analysis or evaluation of effectiveness. It did not investigate necessary modifications to existing distributed protocols to maintain system performance under the scheme. Node capacity heterogeneity, a common argument against the practical use of computational puzzle, was not dealt with. Most importantly, the scheme was *not* a complete defense system. A resource-testing scheme can limit Sybil attacks only if adversaries conduct the work. Obviously, this is not in their best interest, and Borisov provides no discussions about enforcement. The scheme did not prevent or penalize misbehaving nodes who avoided computational work, and did not protect honest nodes from these Sybils. SybilControl is a complete defense system that addresses all these short-comings.

2.3 Computational Puzzles

Computational puzzles have been applied for security purposes for quite some time. In addition to applications to Sybil attacks [7], puzzles have been used to defend against denial-of-service attacks [5, 15, 21, 28, 37] and email spam [5, 17, 18]. Computational puzzles are one-way cryptographic functions that require significant computational resources to find a solution, but are simple to verify. The functions can be CPU-intensive or memory-bound [4, 17], with parameters that can be modified to control puzzle difficulty. Memory-bound functions may be especially appropriate in heterogeneous networks where variations in memory storage size may be less than in CPU computing power. In this paper, we adopt the puzzle proposed by Borisov [7] for concreteness, but other puzzles should be appropriate as well.

3. OVERVIEW

The goal of SybilControl (or other Sybil defense schemes) is not to completely prevent adversaries from joining the system, but rather to place a limit on the number of additional Sybil nodes adversaries can join. This prevents them from obtaining significant influence over the system. SybilControl operates towards this goal using the following insight: if a computational cost is incurred by nodes before they are allowed to join the system, then adversaries with finite resources will have an upper bound on the rate they can acquire identities. Moreover, if nodes are required to periodically repay this computational cost to retain their identities, then the number of identities that can be maintained by the adversary will be limited.

To leverage this insight, SybilControl uses computational puzzles to control admission and retainment of nodes in a system. It provides a distributed protocol that allows nodes to collectively verify that their neighbors are paying computational costs (through solving puzzles). Then in this environment, it provides enforcement mechanisms that eliminate the influence of misbehaving nodes that do not solve puzzles. In this section, we describe the system model that characterizes this environment (Section 3.1). Then, we discuss how SybilControl addresses two key challenges in establishing this environment (Section 3.2 and 3.3).

3.1 System Model

In order to characterize the security properties of SybilControl, we define a system model in a DHT context by listing our assumptions on the abilities and goals of different principals in our system. We use this model when evaluating the security bounds of SybilControl.

System-wide: We make the following assumptions about all nodes in the system:

- Nodes may join or depart at any time. Departures may be graceful, graceless, or a result of node failure.
- The network may or may not be synchronous.
- Nodes in the network may or may not be homogeneous in computational capacities.

Adversary: We consider an adversary whose goal is to insert as many Sybil nodes into the system as possible. We make the following assumptions about the attacker:

- The adversary **is** computationally bounded. We do not consider, nor provide any guarantees, for powerful adversaries with unlimited computational capabilities.
- Since the adversary's goal is to maximize the number of inserted Sybils, the adversary will devote the entirety of her computational capabilities to increasing the number of Sybils in the network.
- The adversary may have unlimited resources of different nature. For example, the adversary may have access to unlimited bandwidth or unlimited IP addresses (which would defeat IP address based defenses such as [13]).
- Sybils in the network may be colluding and byzantine. They may or may not follow SybilControl or DHT protocols.

Honest users: The other nodes in the system belong to honest users, who aim to participate correctly in the system to obtain DHT services. We assume the following behavior for honest users:

- Honest users follow SybilControl and DHT protocols, and do not attempt to perform any attacks or collude maliciously.
- Honest users are also computationally bounded, and may or may not be limited in other resources.
- Honest users will devote exactly enough resources to support one node in the system, unless directed to do otherwise by SybilControl. Dependent on the computational capabilities of different users, some honest users may have spare computational capabilities.

Note that by assuming honest users might have spare computational power, we can leverage heterogeneity (as discussed in detail in Section 4.2.3). Honest users can use spare computation to support additional nodes. With the addition of more nodes, the influence of existing Sybils is deflated. Most importantly, these additional nodes are honest, and will behave as we outlined above. Note the adversary already devotes all capabilities to supporting Sybils, and cannot benefit further.

3.2 Collectively Verifying a Node

In a distributed system, we lack a centralized authority to verify puzzle solutions of new arrivals. To address this, SybilControl allows decentralized groups of nodes to collectively verify the computational work done by their neighbors. Then in this environment, before a node A trusts communication with another node B, A

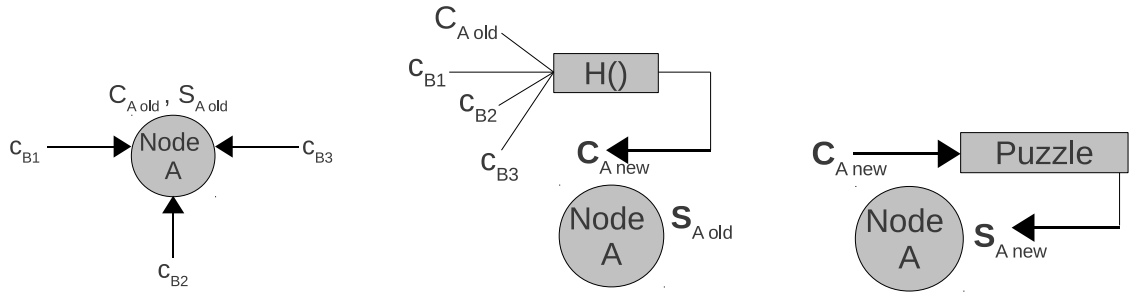


Figure 1: Overview of SybilControl. (a) Node A maintains a puzzle solution s_{A-old} and its associated challenge c_{A-old}. Node A receives challenges c from other nodes B_1, B_2, and B_3, which (b) are used along with node A's old challenge to periodically compute a new challenge c_{A-new}. (c) Node A periodically generates a fresh puzzle solution based on its new challenge.

requires B to prove that it recently solved a puzzle. If B is a malicious Sybil node and chooses not to solve the puzzles, it will not be able to provide proof of work. Defense mechanisms in SybilControl protect honest nodes like A from using B, essentially making B non-functional in the system and preventing it from doing harm. This forces adversaries to use only puzzle-solving Sybils, of which they can support a limited number.

To establish recurring proof of work, SybilControl uses a distributed *collective verification scheme*, where nodes periodically challenge each other to solve new puzzles. Following this scheme, if a group of nodes B_1, B_2, and B_3 collectively desire to communicate with another node A, they each periodically create, record, and send new challenges to A, as in Figure 1(a). A also periodically creates a new challenge using the latest received challenges, as in Figure 1(b), and uses that new challenge to solve a new puzzle, as shown in Figure 1(c). When any one of the nodes requests a service from A, for example B_1, A responds with the service as well as information from the latest puzzle it solved. If B_1 still has recorded the original challenge used in the puzzle, B_1 can verify A's puzzle solution. The duration for which B_1 records challenges can put a bound on how recent A's solution is, validating its timeliness. If all nodes in the network formed a single group and collectively challenged each other, then all pairs of nodes can perform direct verification.

3.3 Verifying Across Multiple Hops

Performing direct verification between all pairs of nodes in the network can be prohibitively expensive. While it may scale to systems that communicate in a full mesh (e.g., one-hop overlay networks), many systems restrict the number of neighbors a node is allowed to communicate with for scaling purposes (e.g., in DHTs like Chord, nodes maintain regular communication relationships with only a logarithmic number of adjacent neighbors). To support these systems, SybilControl provides *multi-hop verification*.

To do this, nodes in SybilControl only exchange challenges with their neighbors. Neighbor relationships may be selected arbitrarily, or in a manner based on the instrumented distributed system (e.g., when applying SybilControl to the Chord DHT, challenges may be exchanged only with a node's fingers and successors). Each node, as in Figure 1(b), then performs a *cryptographic aggregation* step to combine the challenges received by its neighbors, and uses this aggregation as input to construct its own challenges to be sent to its neighbors. This process repeats, allowing a node's challenge to be distributed throughout the network through aggregations. SybilControl provides a multi-hop verification process, allowing a node to check whether its challenge is indirectly incorporated by a remote node. This allows indirect proof of work to be established between a node and non-neighbors.

4. SYBILCONTROL DESIGN

In this section, we discuss the details of SybilControl's design. SybilControl limits the extent of Sybil attacks by enforcing that nodes perform periodic computational work to remain in the system. This is achieved using two parts: a collective verification scheme and protection mechanisms. The collective verification scheme establishes an environment where nodes can collectively verify the computational work of other nodes. The protection mechanisms utilize this verification environment to allow honest nodes to protect themselves from Sybil nodes. These mechanisms eliminate the threat of Sybil nodes that do not solve puzzles. This forces adversaries to use only verifiable Sybils, of which they can support a limited number. Section 4.1 describes the collective verification scheme, while Section 4.2 details the protection mechanisms.

4.1 Collective Verification Scheme

The collective verification scheme of SybilControl allows nodes to challenge one another to solve computational puzzles and to collectively verify the completion of the work. This creates an environment where honest nodes can detect and avoid contact with misbehaving nodes that do not perform work. Section 4.1.1 discusses the distribution of puzzle challenges, while Section 4.1.2 illustrates the process of solving puzzles and directly verifying neighbors. Multi-hop verification is used to verify non-neighbor nodes, as described in Section 4.1.3.

4.1.1 Distributing Challenges

SybilControl-enabled nodes must efficiently distribute numerical challenges to other nodes. Since in most distributed systems, nodes regularly ping their neighbors to ensure availability, these ping messages provide a convenient medium for carrying challenges. We assume use of a modified ping message that additionally includes the challenge. This results in challenge distribution only directly to neighbors, an efficient method which scales with the underlying system and requires little overhead.

The difficulty with this method is propagating a challenge throughout the network while communicating with only neighbors. To overcome this, a node's new challenge is computed periodically as a cryptographic aggregation of received challenges, as in Figure 1(b). Let node A be the neighbor of m nodes B_1,...,B_m, and the

latest challenges A has received are c_{B_1},...,c_{B_m}. When computing a new challenge, A generates a new random number r_A. Then:

$$c_{A_{new}}=H(B_1||c_{B_1}||...||B_m||c_{B_m}||r_A||c_{A_{old}})$$

where $||$ represents concatenation and $H()$ can be a secure hash function such as SHA-2 [27]. Due to the one-way nature of a secure hash function, a node cannot modify r to create a predetermined challenge based on received challenges. Since new challenges are constructed using a cryptographic aggregation of received challenges, challenges are indirectly propagated throughout the network without direct all-to-all distribution. When A distributes $c_{A_{new}}$, A's neighbors will receive a challenge that indirectly includes the challenges of B_1,...,B_m.

Each time a new challenge is computed, the data associated with its construction must be maintained for a period of time. This *state record* $R_{c_{A_{new}}}$ allows future verification of a puzzle that uses the new challenge. When $c_{A_{new}}$ is computed, $R_{c_{A_{new}}}$ consists of $c_{A_{new}}$, $c_{A_{old}}$, r_A, and a table mapping B_i to c_{B_i} for i in $[1, m]$.

$R_{c_{A_{new}}}$ must be maintained as long as the challenge remains a viable candidate for puzzle verification at other nodes. Say each node sends a ping every p seconds (not necessarily in-sync), network latency is no greater than p, and the network has diameter d. It takes at most $2pd$ seconds for a challenge to indirectly propagate across the network. In most N node overlay networks such as Chord [34], $d = O(\lg N)$. Now, if node A just received a challenge from node B, it will receive another challenge within $2p$ seconds. If A doesn't start a new puzzle within $2p$, the challenge it just received will not be used. Thus, the maximum time for a used challenge to propagate the network is $2pd + 2p$. If a puzzle is solved every s seconds, a specific challenge may be needed for $2s$: s seconds for solving a puzzle with that challenge and s seconds before a new puzzle is solved (with a different challenge). Finally, the verification protocol may take up to $2pd$ before verifying a node across the diameter (discussed later in Section 4.1.3). Therefore, records should be stored for $2s + 4pd + 2p$ seconds. Note that this time period bounds the time a challenge may influence puzzle solutions. If a puzzle solution S is reused, nodes across the network will no longer store the challenges that influenced S, and will not verify S.

4.1.2 Solving Puzzles and Directly Verifying Neighbors

For concreteness, here we describe the computational puzzle proposed by Borisov [7], but any computational puzzle from prior works (Section 2.3) should be appropriate. A node A's puzzle solution S is a value such that:

$$H(A||c_A||S) = h$$

where $H()$ is again a secure hash function (SHA-2 [27]) and h is a number with zeros as the last p bits. Solving this puzzle requires $O(2^p)$ hash computations using random guesses for S, while verification requires a single hash evaluation. Varying p, a system parameter, can control the difficulty of puzzles. Note that since c_A is dependent on received challenges, S cannot be precomputed.

A SybilControl-enabled node periodically (every s seconds) solves a new puzzle using the node's latest computed challenge. Thus, puzzle solutions depend on received challenges. The challenge used should be computed from newly received (direct and indirect) challenges. Since it takes at most $2pd$ seconds (see Section 4.1.1) for a challenge to indirectly propagate across a network, a reasonable value for s is at least $2pd$ seconds. Note s has a lower bound because the slowest user must be capable of solving a puzzle in s seconds, and p should be set accordingly.

Whenever a node A solves a puzzle, it records the *puzzle state* P. P consists of the puzzle solution S_P, the used challenge c_{A_P}, and the challenge's state record $R_{c_{A_P}}$. To provide puzzle verification, nodes provide P. Note only the latest puzzle state is necessary because even nodes across the diameter of the network will still have on record the challenges that indirectly influenced c_{A_P} (see Section 4.1.1). If a puzzle solution is reused, this no longer holds true, and nodes across the network will no longer verify P.

To directly verify a neighbor B, A requests B's puzzle state P_B. A can validate the freshness of the P_B by confirming that $R_{c_{B_P}}$ contains a challenge from A still on record. This guarantees that P was generated within the last $2s + 4pd + 2p$ seconds. Then A can use the remaining data in P_B to compute and verify S_{P_B}. More specifically, A can use the challenge records in $R_{c_{B_P}}$ to recompute/confirm c_{B_P}, then rehash to check S_{P_B}. If any step in the validation fails, A can assume B is misbehaving and avoid further communication with it, inhibiting B's negative influence.

4.1.3 Verifying Non-Neighbors along a Path

Neighbors-only challenge distribution prevents non-neighbor direct verification. However, non-neighbors can still be indirectly verified through *multi-hop verification*. During multi-hop verification, validation is conducted iteratively along a path starting at a neighbor, as such:

As described in Section 4.1.2, node A directly verifies neighbor B by requesting and validating P_B. However, when verifying a path, A requests B to also (in addition to P_B) send its on-record challenges (say c_B^i,...,c_B^j) and their state records ($R_{c_B^i}$,...,$R_{c_B^j}$). This allows A to verify the correctness of any particular challenge c_B^x in c_B^i,...,c_B^j. A can confirm the challenge from A stored in $R_{c_B^x}$ is still on A's record, and recompute c_B^x using challenge records in $R_{c_B^x}$ to ensure it matches. Using this method, A verifies the correctness of most, although possibly not the tail-end (oldest few challenges), of c_B^i,...,c_B^j.

Now, A temporarily stores the verified portion of c_B^i,...,c_B^j for the next hop's verification. The following node C in the path is a neighbor of B, and C's puzzles and challenges are computed based on challenges received from B. A can now request and verify C's puzzle state and challenge records (as described above in this section) using the verified B records, instead of A's own records. If C is verified, A can again verify a portion of C's records, then replaces its temporary records for B with the verified portion of C's records. In this manner, A can iteratively continue to verify nodes along a path, providing indirect verification to any node in the network. Assuming one-way latencies no larger than p, the most time this process can take is $2pd$. Additional proof of correctness is provided in Appendix B.

4.1.4 Comparison with Borisov's Scheme

SybilControl was inspired by many aspects of Borisov's scheme [7], but was designed differently primarily due to a protocol flaw. Borisov's scheme also distributes challenges along ping messages. However, it makes the clear assumption that every challenge sent will directly be incorporated in another node's new challenge. When factoring in network latencies on these pings, this assumption no longer necessarily holds.

In Borisov's scheme and SybilControl, a state is recorded every time a node A computes a new challenge (which is every ping message). This state contains the latest challenges received. However, it is possible (due to varying latencies) that a node B received two challenges from A in the time period between computing new challenges. In this scenario, the prior of the two challenges will be lost.

At a high level, Borisov's verification scheme operates by having A verify other nodes using a particular challenge c_A^x, chosen using temporal heuristics. Since in certain scenarios, c_A^x may have been lost, nodes may fail verification, leading to false positive detection of a misbehaving node. SybilControl avoids this "lost challenge" problem using a different verification protocol that doesn't operate using a specific challenge, but rather a range of possible challenges.

Borisov did not comment on the nature of the network with regards to synchronization. However, note that if the network is completely synchronized, this problem would no longer occur, since a new ping/challenge would not be sent until a node received all new challenges. Enforcing synchronization in such a system could be difficult and possibly impractical though. SybilControl does not depend on a synchronized network.

4.2 Protection Mechanisms

Section 4.1 describes a scheme that creates an environment where nodes can verify computational work of other nodes. However, to limit the influence of Sybils, this work must be enforced. This section discusses mechanisms that protect honest nodes from the influence of misbehaving Sybils, essentially enforcing the adversary performs computational work, thus limiting the attack strength. An adversary can use one of three strategies to perform a Sybil attack.

The first strategy for a Sybil attack is to use consistently unverifiable Sybil nodes. These Sybils never perform computational work, allowing an adversary to support a large scale attack. Section 4.2.1 details complete protection from this attack strategy, regardless of the scale of attack. The second attack method is using initially verifiable Sybil nodes, which bypass the defense in Section 4.2.1. After incorporation into the network, these Sybils halt computational work so an adversary can attempt to join and support even more malicious nodes. Section 4.2.2 provides simple mechanisms that control the influence from such an attack. The mechanisms described in these two sections inhibit the influence of unverifiable nodes, which would force adversaries to use consistently verifiable Sybils, achieving our enforcement of computational cost. Use of such Sybils is the final form of a Sybil attack. These Sybils correctly follow the collective verification scheme, making them indistinguishable from honest nodes, but may misbehave otherwise. However, an adversary with limited resources can only support a restricted number of verifiable Sybils, reducing their malicious influence. Section 4.2.3 provides a mechanism that leverages node capability heterogeneity for further minimizing the impact of this attack form.

4.2.1 Protecting against Consistently Unverifiable Sybils

With the collective verification scheme, nodes can establish trust in verifiable neighbors. However, to manage a node's neighbor relationships, churn-handling protocols encounter unverifiable new nodes. If these protocols immediately establish newly encountered nodes as neighbors, a large number of consistently unverifiable Sybils could hijack neighbor relationships.

To remedy this adversarial situation, SybilControl-enable nodes use a *delayed-adding* mechanism, where nodes only establish neighbor relationships upon verification. Thus, a Sybil that never solves a puzzle will never be considered a neighbor of any honest nodes, and will have no influence. With the delayed-adding mechanism, nodes newly discovered through churn-handling are temporarily managed in an *untrusted list*. Challenges are distributed to both established neighbors and nodes in the untrusted list. After the time to solve two puzzles, the untrusted node must be verifiable if behaving correctly, so a direct verification is conducted. Only if verification succeeds does that node move out of the untrusted list to become an established neighbor.

The untrusted list is bounded in size to $O(\#$ of neighbors$)$ to prevent a memory exhaustion attack. Let churn-handling protocols inspect a particular neighbor relationship every f seconds, and s be the puzzle time. At most $\lceil \frac{2s}{f} \rceil$ potential new neighbors can be discovered for a particular relationship before one is checked for verification and removed from the untrusted list. Thus, the constant factor is $\lceil \frac{2s}{f} \rceil$.

The delayed-adding mechanism provides complete protection from consistently unverifiable Sybil nodes in that those nodes will never hijack neighbor relationships. It does introduce a $2s$ time delay cost in establishing a neighbor. However, the newly discovered honest nodes are not immediately verifiable anyways, and should not be considered viable neighbors. Churn-handling protocols often rely on distributing information about neighbor relationships, meaning newly discovered nodes would not be included in churn-handling until verification. This can degrade churn-handling performance, which may not be acceptable to systems sensitive to churn.

To maintain efficient protocol performance, churn-handling protocols are modified to include nodes in the untrusted list. If churn-handling protocols are utilizing information about the i-th neighbor, and there is a node in the untrusted list that is next to replace that neighbor relationship, then the protocols distribute information on the untrusted node. This allows churn-handling to maintain fast convergence times by operating as if the newly discovered node was immediately established as a neighbor. Note that this does not reduce the protection of the delayed-adding mechanism, as a consistently unverifiable node will remain only in the untrusted lists of nodes during churn-handling. Therefore, consistently unverifiable Sybil nodes simply cannot obtain positions to maliciously influence the system. This mechanism now controls the admission rate of new nodes by requiring each to solve puzzles before establishing a position.

4.2.2 Protecting against Initially Verifiable Sybils

The mechanism in Section 4.2.1 removes the threat of consistently unverifiable Sybils. However, a malicious node can bypass the defense by initially performing work only until it is no longer in the untrusted list, and has become an established neighbor. The adversary, now no longer paying a computational cost, may attempt to join more nodes. This vulnerability arises because nodes are validated in the collective verification scheme only upon contact, which may be infrequent.

To provide nodes with protection from this attack, two mechanisms are introduced. The first mechanism is the *backup neighbors list*. When churn-handling with the delayed-adding mechanism replaces an old neighbor relationship with a new neighbor, the old neighbor is still maintained in the backup neighbors list if it is currently *safe*. A node can be determined safe by simply conducting a direct verification. This backup list serves as a level of redundancy should the primary neighbors become overrun with initially verifiable Sybils. Eventually, these Sybil will be detected and removed from the primary neighbors, which can be repopulated using both churn-handling and backup neighbors. Note that challenges must still be sent to nodes in the backup list because they may be utilized should a primary neighbor fail. The cost of this mechanism is a two-fold increase in the amount of memory used to maintain neighbor links and in ping messages.

To ensure that a node's neighbors do not go unverified for long periods of time, the second defense mechanism is simple: *periodic neighbor verification*. Neighbor and backup neighbors can be pe-

riodically verified, much like neighbor relationships being periodically updated with churn-handling protocols. If any neighbors fail verification, they can be replaced with a new node through churn-handling or the backup list. Thus, adversaries can enter only a limited number of initially verifiable Sybils, since Sybils no longer performing work will be soon removed from all neighbor relationships and become uninfluential. This mechanism now controls retainment of nodes in the system by requiring nodes to consistently compute puzzles.

Ideally, neighbor relationships would be verified every puzzle time s, to ensure all neighbors are consistently performing work. However, the main purpose of periodic neighbor verification is to ensure that at least a subset of a node's neighbors is still fully verifiable and can be used for system operations. Then, verifying a fraction of neighbors every s seconds can still be effective if the portion is large enough such that the node can still function if the non-verified portion is compromised. We leave it to specific applications to determine the most appropriate frequency of this periodic table verification.

If an adversary has large amounts of computational resources, it could support enough initially verifiable Sybils to fill up both the primary and backup neighbor list of some nodes. SybilControl cannot prevent this from occurring and does not offer guaranteed protection. However, use of additional backup lists can provide more protection, at the cost of additional memory storage. Furthermore, Section 4.2.3 discusses a mechanism that can reduce the initial influence of these initially verifiable Sybils, which may further reduce this threat.

4.2.3 *Leveraging Heterogeneity to Protect Against Consistently Verifiable Sybils*

The final strategy of a Sybil attacker is to support consistently verifiable Sybil nodes. These nodes are indistinguishable from honest nodes because they follow the collective verification scheme correctly, but may misbehave otherwise. While SybilControl cannot prevent these "invisible" Sybils from doing any harm, it can provide a mechanism that reduces the total influence these Sybils may have. The mechanism is *virtual node usage*, which leverages heterogeneity in node computational capacities.

Computational puzzle schemes tend to suffer from the heterogeneity of node capacities in a system. However, this heterogeneity is leveraged by SybilControl for improved Sybil protection. Puzzles must require a maximum amount of computational resources that should be manageable by nodes with less capacity. However, many honest nodes will have significantly more computational power than that maximum. Virtual node usage is a mechanism where honest users are allowed to optionally control extra virtual nodes if they have the resources to support them. With more honest nodes, a larger fraction of the network is controlled by honest users, and Sybil attacks would have less influence.

To provide some intuition as to the benefits of using virtual nodes, assume we have an n node DHT network. On average, each node controls $\frac{1}{n}$ of the keyspace. If an adversary's resources can support a maximum of m Sybil nodes in the system, the adversary can control on average $\frac{m}{n+m}$ of the keyspace. If the average honest user has enough computational resources to maintain work for q nodes, then by allowing users to support q virtual nodes, the m-node Sybil attack will now only influence an average of $\frac{m}{q*n+m}$ of the keyspace, which can be a significantly smaller portion. This illustrates that use of virtual nodes can protect more of the keyspace and reduce the influence of Sybil nodes. In addition, the number of these Sybils should already be limited by the collective verification scheme and the mechanisms in Section 4.2.1 and 4.2.2, which prevent unverified nodes from being a significant threat.

Note that it is inconsequential that we cannot distinguish honest from malicious users when assigning virtual nodes. Distributed systems vulnerable to the Sybil attack already have a low or no cost of entry, and allowing malicious users to host virtual nodes will not significantly reduce the cost of inserting Sybils. Attackers will devote full computational capacity regardless (as specified in Section 3.1) to supporting a maximal number of Sybils, whether through virtual nodes or other entry methods. Enabling virtual nodes simply allows honest users the ability to insert extra nodes to diminish Sybil influence.

5. A SYBILCONTROL-ENABLED DHT

SybilControl creates a unique environment since nodes may or may not be verifiable. This can exacerbate the effect of churn in distributed systems. In this section, we discuss practical issues associated with the deployment of SybilControl. Our focus is on ensuring distributed systems maintain reasonable performance with SybilControl enabled. To provide concrete examples of techniques, we describe SybilControl in the context of DHTs. Many distributed systems, such as distributed file systems, distributed databases, and peer-to-peer networks, rely on a DHT as an underlying structure for storage or routing. While we describe this section in the context of DHTs for clarity, the general principles should be applicable to most other distributed systems.

In Section 5.1, we discuss issues introduced by deploying SybilControl. Our focus is reusing common techniques for improving DHT performance, which many existing implementations are likely to have. The techniques for a practical SybilControl-enabled DHT involve resilient lookup protocols described in Section 5.2, and replication mechanisms described in Section 5.3.

5.1 Issues with Deploying SybilControl

In a SybilControl-enabled system, nodes may not immediately trust certain other nodes, such as newly joined nodes or un-established neighbors. SybilControl uses the delayed-adding mechanism, detailed in Section 4.2.1, to handle these untrusted nodes and include them in churn-handling protocols. While information about these changing neighbor relationships are propagated throughout the network, they do not manifest into new viable neighbors until the untrusted nodes provide proof of computational work, which requires some time for puzzle completion. This delay between when a new relationship is discovered and when it becomes viable can result in increased sensitivity to churn in SybilControl-enabled systems.

For example, in a DHT such as Chord, a newly joined node A immediately gains control over a portion of the keyspace that previously was supported by its immediate successor B. In a SybilControl-enabled DHT, the delayed-adding mechanism of B prevents it from immediately transferring control to A as it has not been established as a verifiable neighbor yet. This is desirable for Sybil defense since the newly joined node could be an unverified Sybil. However, B could fail before A becomes verifiable, and all data at B may be lost, including those that would have been transferred to A. This possible scenario, among many others, is created by the SybilControl environment, and must be handled to avoid system performance degradation.

5.2 Resilient Lookup Protocols

The primary DHT operation is the lookup for a key, where a node determines the path to the objects or nodes associated with that key. To prevent lookup performance degradation, a SybilControl-

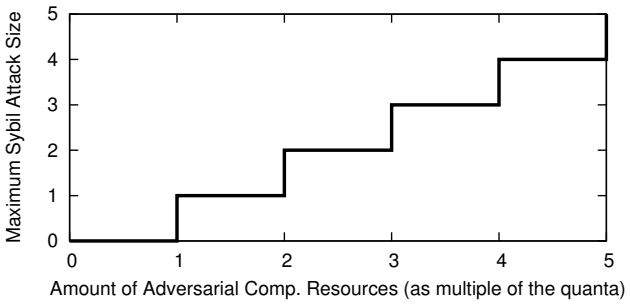


Figure 2: Maximum size of a Sybil attack given the amount of resources an adversary controls, as a multiple of the quanta.

enabled DHT can use resilient lookup protocols. These resilient lookups not only improve lookup performance with SybilControl, but are generally used to provide resilience against Sybil attacks. Many DHTs already support resilient lookups, so use of this technique does not require extensive modifications to existing DHTs.

Resilient lookups involve backtracking and redundant lookups. Since SybilControl requires iterative path verification, multi-hop lookups in SybilControl systems must be iterative. With backtracking lookups, the lookup initiator maintains a history of contacted nodes, and backtracks when encountering a failed or unverified node. This results in lookups resilient to node failures or unresponsive Sybils. Lookup query messages can additionally carry the list of contacted nodes, including failed nodes. This provides a heuristic for nodes responding to lookup queries to ignore already-contacted nodes when determining the next hop in the lookup path. DHT lookups are designed to provide the best next-hop suggestion, and revisiting nodes would result in less-than-optimal cycles.

With redundant lookups, multiple backtracking lookups are conducted for the same key while sharing a common list of contacted nodes. This results in multiple unique lookup paths. In the face of Sybil attacks or failed nodes, redundant lookups are more resilient because they fail only if all lookups contact Sybil or failed nodes, which is less likely with a larger number of redundant lookups.

5.3 Replications

In DHTs, the failure of a node maintaining some data can result in the loss of that data. A standard idea to improving DHT robustness, which applies also when enabling SybilControl, is to replicate, by storing additional copies of that data. With object-storage DHTs, where nodes maintain some data or files such as in a distributed file system, several replications can be supported by different nodes. Even if one node fails or is misbehaving, several other replicas are accessible. In routing DHTs, nodes hold pointers or addresses to other nodes to be used for routing. With replications, several nodes maintain the same pointers or addresses. Similar work has been done with the i3 Internet Indirection Infrastructure system [33].

Replications can be maintained locally with respect to identifiers. With local replications, a node and several consecutive predecessors and/or successors can maintain redundant copies. It is less likely that these nodes will all simultaneously fail or be subverted by an adversary. However, the system is most robust if non-local nodes also support redundant copies. This prevents a targeted attack to a specific portion of the keyspace to result in lost access to all data. The details of this are more specific to the underlying system, and are beyond the scope of our discussion.

6. EVALUATION

In this section, we analyse the security bounds provided by SybilControl, and evaluate the performance of a SybilControl-enabled DHT to determine the performance overhead of SybilControl. This investigation is to determine the practicality of incorporating SybilControl in real-world distributed systems. We investigated the DHT system performance using our own discrete event simulator of the Chord DHT. We chose Chord for its simplicity and scalability. Since SybilControl nodes only send messages to direct neighbors, it scales along with Chord. The simulator directly implements standard Chord protocols as well as the protocols and mechanisms described in Section 4 and Section 5.

6.1 Security Bounds Analysis

SybilControl is designed to ensure that all nodes, even if malicious, pay a recurring computational cost. A properly configured SybilControl would use a computational puzzle whose cost is affordable to the minimal resource node in the network. In this section, this minimal cost level will be referred to as a *quanta* of computational resource. Using the model described in Section 3.1, this section analyses the bounds SybilControl enforces on the extent of Sybil attacks, and quantifies their effectiveness through a simple cost analysis.

Let p_r be the fraction of adversary-controlled resources once the adversary joins the network, and p_i be the maximum fraction of Sybil identities in the system the adversary can support. There are two situations to consider: whether the network nodes are homogeneous or heterogeneous with respect to computational resources.

6.1.1 Homogeneous Networks

Assume that a system has n honest users' devices connected, having uniform computational resources, such that each device has access to x amount. In this case, the quanta can be set to x, so each device supports one honest user node. The following properties hold for SybilControl, showing the proportion of Sybils in the network is limited to the proportion of adversary-controlled resources.

Property 1: *There exists a tight upper bound on the maximum size of a Sybil attack launched by an adversary with finite resource amount T.*

With a quanta of x, the adversary must utilize x resources per Sybil node. Thus, the maximum size of a launched Sybil attack is tightly upper bounded by $\lfloor \frac{T}{x} \rfloor$. This upper bound is a step function, as shown in Figure 2.

Property 2: *Assuming a finite resource adversary, $p_i = p_r$.*

The total resource amount of honest devices is nx. Let the finite resource adversary control T amounts of resources. Without loss of generality, let T be an integer multiple of x. If T is not an integer multiple, the "remainder" resources T *mod* x would simply go unused. Now, $p_r = \frac{T}{nx+T}$. From Property 1, the maximum Sybil attack size is $s = \frac{T}{x}$, and $p_i = \frac{s}{n+s}$. Thus, $p_i = \frac{T/x}{T/x+n} = \frac{T}{T+nx} = p_r$.

6.1.2 Heterogeneous Networks

Assume now that the n honest devices connected to a system have heterogeneous amounts of computational resources. Let there be D type of devices, $d_0, d_1, ..., d_{D-1}$, ordered in increasing computational capacity. The quanta should be set to the resource amount of d_0, the least capable device. Let n_i be the number of d_i devices in the system, such that $\sum_{0 \leq i < D} n_i = n$. Let a_i be the

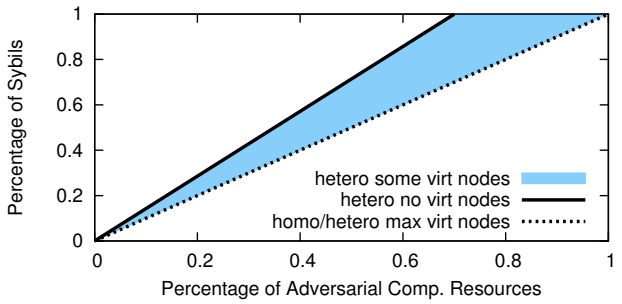


Figure 3: Maximum percentage of Sybil nodes in homogeneous and heterogeneous networks given the percentage of adversarial resources. In heterogeneous networks, varying degrees of virtual node usage lead to a range of maximum Sybil node percentages. No virtual node usage upper bounds that range, while maximum virtual node usage lower bounds the range and is equivalent to the relationship in homogeneous networks. The depicted heterogeneous network uses a hypothetical degree of heterogeneity.

resource amount of device d_i, as a multiple of x, so $a_0 = 1$ and $a_i \geq 1$. Then the system's total amount of honest resources is $R = x\sum_{0\leq i<D} n_i a_i$. Note that Property 1 of Section 6.1.1 still holds. The following properties hold for SybilControl, showing the proportion of Sybil nodes is bounded with relations to the proportion of adversary-controlled resources.

Property 3: *Assuming no virtual node usage, p_i is bounded, given a finite resource adversary. However, $p_i > p_r$.*

Assume the adversary has T resources, and without loss of generality, that T is an integer multiple of x. From Property 1, the maximum Sybil attack size is $s = \frac{T}{x}$. Thus $p_i = \frac{s}{n+s} = \frac{T/x}{n+T/x} = \frac{T}{nx+T}$. Also, $p_r = \frac{T}{R+T}$. Since the network is heterogeneous, there is some $a_i > 1$. Then, $R > nx$, and $p_i > p_r$.

Property 4: *Assuming maximum virtual node usage, $p_i = p_r$.*

Again, $p_r = \frac{T}{R+T}$ and the number of Sybil nodes supported is $s = \frac{T}{x}$. With maximum use of virtual nodes, all system resources are used, so the total number of honest nodes supported by the n honest devices is $v = \frac{R}{x}$. Then, $p_i = \frac{s}{v+s} = \frac{T/x}{R/x+T/x} = \frac{T}{R+T} = p_r$.

Property 5: $\frac{T}{R+T} \leq p_i \leq \frac{T}{nx+T}$, *depending on the degree of virtual node usage.*

This follows from the analysis provided for Property 3 and 4. If no virtual nodes are used, the fraction is $\frac{T}{nx+T}$. As more virtual nodes are used, this fraction decreases, down to $\frac{T}{R+T}$ when the maximum number of virtual nodes are used. Essentially, virtual nodes eliminate adversarial advantage due to a small value for the quanta, and result in a per computation fair share of identities. Figure 3 depicts the relationship between adversarial resources and adversarial nodes in a homogeneous and hypothetical heterogeneous network.

The more heterogeneous the system, the larger the values of a_i are. Therefore, the difference between nx (where every $a_i = 1$) and R increases, and the given range increases. Thus, the effect of using virtual nodes will increase as the system becomes more heterogeneous. A simulation of the benefits of virtual nodes in a SybilControl-enabled DHT is demonstrated in Appendix A. It shows that use of virtual nodes does improve the performance of DHTs under Sybil attacks by reducing the influence of Sybil nodes.

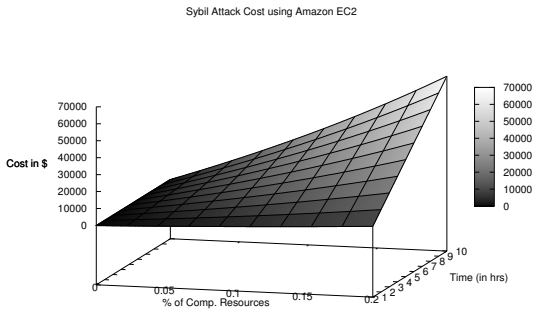


Figure 4: Cost of a Sybil attack using Amazon EC2 services

6.1.3 Preliminary Cost Analysis

SybilControl's upper bound is dependent on the percentage of computational resources the adversary controls. To provide some intuition as to the effectiveness of this bound, this section provides a simple cost analysis of Sybil attacks in a SybilControl environment.

One method of obtaining computational resources is through the Amazon Elastic Cloud Computer (EC2) web service [1]. EC2 power is measured in *compute units* (CU), which Amazon defines as "the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor." [2] The most cost effective service instance is the cluster computer eight extra large instance, which provides 88 CUs for $2.40 per hour [3].

In our example scenario, we assume a system totalling one million CUs. In Figure 4, the cost of a Sybil attack is depicted, as a function of the percentage of computational resources controlled by the adversary and length of time of the attack. For large attacks controlling 20% of the resources, every attack-hour costs $6,818.18. A one day attack would cost $163,636.36. In many situations, this high monetary cost may prevent adversaries from launching large and lengthy attacks. Note that honest users do not need extra computing power, so would not incur similar financial costs. Also, if the total system resources differ by a certain order of magnitude, then the costs will differ likewise. This provides evidence that adversaries may remain limited in controlled computational power to avoid incurring too large of a cost, which allows SybilControl to provide an effective bound.

While there are other ways to obtain computational resources, they also typically require payment for services, and would exhibit similar increasing financial costs for adversaries. For example, an adversary could rent a botnet to host Sybil nodes. The rental cost is likely cheaper than the Amazon EC2 estimates, but we leave an in-depth cost analysis for future works. We should note that in the botnet case, unrestricted use of the computational capacity of a compromised machine might raise users' suspicions. To avoid this, adversaries may have to use only a portion of a compromised machine's resources. To achieve the same amount of resources to support Sybils, adversaries would then need more machines, which would increase the financial cost.

6.2 Performance Overhead of SybilControl Deployment

This section investigates the performance overhead of enabling SybilControl in the Chord DHT. To allow for consistent comparisons, all simulations followed the same network dynamics. Initially, the network consisted of 1000 verifiable nodes. Nodes sent ping messages carrying challenges every 5 seconds. We used a

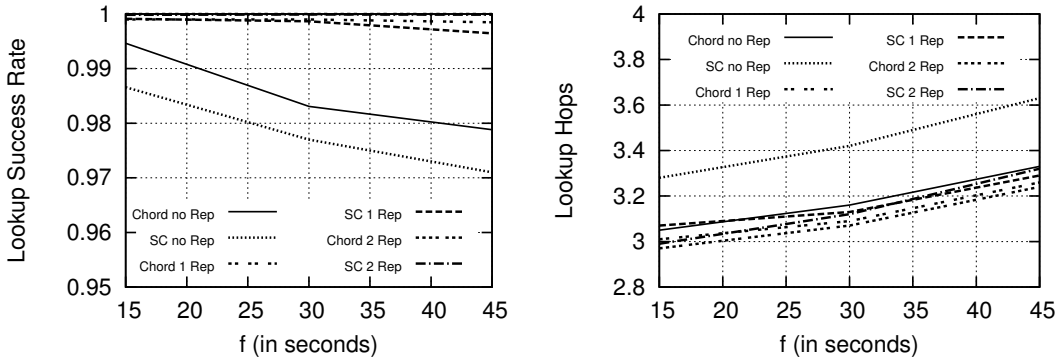


Figure 5: Lookup performance of regular Chord and SybilControl using different number of replications versus f, the frequency of churn-handling. (a) shows average lookup success rates, while (b) graphs the lookup hops.

puzzle time of 20 seconds, which is based off our suggestion for a puzzle time larger than pd in Section 4.1.1. We modelled churn similarly to other studies [20], with a pareto distribution for mean session time and an exponential distribution for mean downtime. Since the same studies indicate that the mean session time is typically more than 60 minutes, we used 60 minutes as our distribution means. Chord uses two churn-handling protocols: fix-fingers and stabilize. In the original Chord evaluation [34], the frequency f of each protocol execution was in a range between every 15 seconds to every 45 seconds. We used values of f also within that range. Each simulation was run for 10,000 seconds before measurements, to avoid cold-start effects.

The primary operation in Chord is a lookup, so we evaluated lookup performance metrics for Chord with and without SybilControl enabled. Each alive node in the network attempted 10 resilient lookups for random keys. We measured the success rate of attempted lookups and the number of hops in a successful lookup's path, which we define as the number of nodes contacted, including failed nodes. To accurately measure only performance overhead, the network excluded any misbehaving nodes. f, Chord's fix-finger and stabilize time interval, was varied between every 15, 30, and 45 seconds. Since a smaller f implies more frequent churn-handling, we expected that f is negatively correlated with lookup performance.

Since SybilControl creates an environment that may exacerbate churn, it was our initial hypothesis that the Chord lookup performance could degrade. However, incorporating the techniques in Section 5 should address these issues and improve performance. In particular, using multiple replications in SybilControl was expected to help improve performance. With more replicas, SybilControl lookups should be more likely to succeed in finding an alive, verifiable node that holds a copy. SybilControl and regular Chord were evaluated using 0, 1, and 2 replication.

Figure 5 shows the lookup performance measurements for both regular and SybilControl-enabled Chord using varying number of replications. Figure 5(a) plots the success rate of lookups versus f. Without replications, regular Chord outperformed SybilControl as expected, with lookup success rate about 1% higher for regular Chord. In many systems, this 1% difference in lookup success rate may not be an issue, providing some evidence already that SybilControl could be practically incorporated. However, when the lookup performance degradation is unacceptable, using even one extra replication can significantly improve performance. With one replication, SybilControl resulted in a higher success rather than no-replication Chord. At two replications, average lookup success rate was consistently near 100%. The lookup success rate of SybilControl was only slightly worse than regular Chord given the same positive number of replications. The differences were a fraction of a percent, and indicate that SybilControl's performance overhead can be small on lookup success rate.

Figure 5(b) plots the lookup hops versus f. Again, no-replication SybilControl performed worse than no-replication regular Chord, typically by approximately 0.3 hops. However, once one replication was introduced, SybilControl lookups roughly matched regular Chord lookups in hop count, varying by less than 0.1 hops. Additional replications gave diminishing improvements for both SybilControl and regular Chord. This provides evidence that SybilControl doesn't significantly impact lookup hop counts.

The effect of varying f can be quite drastic. With larger f, lookup success rates for no-replication Chord and no-replication SybilControl declined by as much as 2%, while average hop count increased by as much as 0.3 hops. One-replication Chord and SybilControl seemed to also be affected to a lesser degree. Lookup success rate with two replications seemed unaffected by f, although the hop count exhibited similar behavior as with one-replication. For certain systems extremely sensitive to lookup success rate, it may be necessary to frequently run churn-handling protocols, minimizing f.

These results indicate that SybilControl does slightly degrade Chord lookup performance, as expected. However, by incorporating replications, a fairly common Chord churn-handling technique, SybilControl can provide reasonable performance on par with regular Chord. This indicates that SybilControl can be practically incorporated in many distributed systems.

7. CONCLUSION

This paper presented SybilControl, a novel, decentralized scheme for controlling the extent of Sybil attacks. SybilControl consists of a distributed protocol to allow nodes to collectively verify computational work of their neighbors and defense mechanisms to enforce that nodes conduct computational work to stay functional in the system. Thus, adversaries with finite amounts of computational resources will be only able to support a limited number of Sybil nodes. A security analysis showed that SybilControl provides tight bounds on the maximum Sybil attack size given adversaries with finite resources. The cost of obtaining this bound may be financially costly to many adversaries, which indicates the potential effectiveness of the bound. To demonstrate the practicality of SybilControl, it was deployed in the context of DHTs with the aid of existing

DHT functionalities. The performance overhead when enabling SybilControl was shown to be manageable by using commonplace DHT churn-handling techniques.

8. ACKNOWLEDGEMENTS

We thank Abedelaziz Mohaisen and the anonymous reviewers for useful comments and feedback.

9. REFERENCES

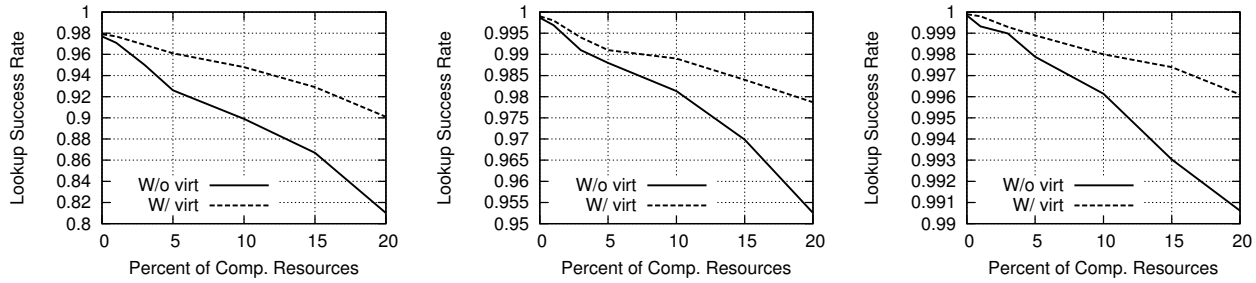
[1] Amazon EC2. `http://aws.amazon.com/ec2/`.
[2] Amazon EC2 instance types. `http://aws.amazon.com/ec2/instance-types/`.
[3] Amazon EC2 pricing. `http://aws.amazon.com/ec2/pricing/`.
[4] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *Proc. NDSS*, 2003.
[5] A. Back. Hashcash - a denial of service counter-measure. Technical report, 2002.
[6] L. Bilge, T. Strufe, D. Balzarotti, , and E. Kirda. All your contacts are belong to us: Automated identity theft attacks on social networks. *Proc. WWW*, 2009.
[7] N. Borisov. Computational puzzles as Sybil defense. *Proc. IEEE P2P*, 2006.
[8] G. Brown, T. Howe, M. Ihbe, A. Prakash, and K. Borders. Social networks and context-aware spam. *Proc. ACM CSCW*, 2008.
[9] Z. Cai and C. Jermaine. The Latent Community Model for detecting Sybil attacks in social networks. *Proc. NDSS*, 2012.
[10] M. Castro, P. Druschel, A. J. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. *Proc. OSDI*, 2002.
[11] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson. Sybil-resistant DHT routing. *Proc. ESORICS*, 2005.
[12] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil nodes using social networks. *Proc. NDSS*, 2009.
[13] J. Dinger and H. Hartenstein. Defending the Sybil attack in P2P networks: Taxonomy, challenges, and a proposal for self-registration. *Proc. IEEE ARES*, 2006.
[14] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *Proc. USENIX Security*, 2004.
[15] C. Dixon, T. Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. *Proc. NSDI*, 2008.
[16] J. R. Douceur. The Sybil attack. *Proc. IPTPS*, 2002.
[17] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fight spam. *Proc. CRYPTO*, 2003.
[18] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. *Proc. CRYPTO*, 1992.
[19] R. Geambasu, T. Kohno, A. Levy, and H. M. Levy. Vanish: Increasing data privacy with self-destructing data. *Proc. USENIX Security*, 2009.
[20] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing churn in distributed systems. *Proc. ACM SIGCOMM*, 2006.
[21] A. Juels and J. Brainard. Client puzzles: A cryptographic counter-measure against connection depletion attacks. *Proc. NDSS*, 1999.
[22] C. Lesniewski-Laas and M. F. Kaashoek. Whanau: A Sybil-proof distributed hash table. *Proc. NSDI*, 2010.
[23] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, , and X. Li. An empirical study of collusion behavior in the maze P2P file-sharing system. *Proc. IEEE ICDCS*, 2007.
[24] P. Mittal, M. Caesar, and N. Borisov. X-vine: Secure and pseudonymous routing in DHTs using social networks. *Proc. NDSS*, 2012.
[25] A. Mohaisen, N. Hopper, and Y. Kim. Keep your friends close: Incorporating trust into social network-based Sybil defenses. *Proc. IEEE INFOCOM*, 2011.
[26] A. Mohaisen, A. Yun, and Y. Kim. Measuring the mixing time of social graphs. *Proc. ACM IMC*, 2010.
[27] U. N. I. of Standards and Technology. Secure hash standard (shs). *Federal Information Processing Standards Publication 180-3*, 2008.
[28] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. Maggs, and Y. Hu. Portcullis: Protecting connection setup from denial-of-capability attacks. *Proc. ACM SIGCOMM*, 2007.
[29] S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. *Proc. ACM SIGCOMM*, 2001.
[30] H. Rowaihy, W. Enck, P. McDaniel, and T. L. Porta. Limiting Sybil attacks in structured P2P networks. *Proc. IEEE INFOCOM*, 2007.
[31] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proc. ACM Middleware*, 2001.
[32] S. Sen and M. Freedman. Commensal cuckoo: Secure group partitioning for large-scale services. *Proc. LADIS*, 2011.
[33] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *Proc. ACM SIGCOMM*, 2002.
[34] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *Proc. ACM SIGCOMM*, 2001.
[35] N. Tran, J. Li, L. Subramanian, and S. Chow. Optimal Sybil-resilient node admission control. *Proc. IEEE INFOCOM*, 2011.
[36] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based Sybil defenses. *Proc. ACM SIGCOMM*, 2010.
[37] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. *IEEE S&P*, 2003.
[38] W. Wei, F. Xu, C. Tan, and Q. Li. Sybildefender: Defend against Sybil attacks in large social networks. *Proc. IEEE INFOCOM*, 2012.
[39] S. Wolchoky, O. S. Hofmanny, N. Heninger, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating Vanish with low-cost Sybil attacks against large DHTs. *Proc. NDSS*, 2010.
[40] M. Yang, Z. Zhang, X. Li, and Y. Dai. An empirical study of free-riding behavior in the maze P2P file-sharing system. *Proc. IPTPS*, 2005.
[41] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A near-optimal social network defense against Sybil attacks. *IEEE S&P*, 2008.
[42] H. Yu, M. Kaminsky, P. Gibbons, and A. Flaxman. SybilGuard: Defending against Sybil attacks via social networks. *Proc. ACM SIGCOMM*, 2006.
[43] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Selected Areas in Communications*, 2003.

APPENDIX

A. BENEFITS OF VIRTUAL NODES

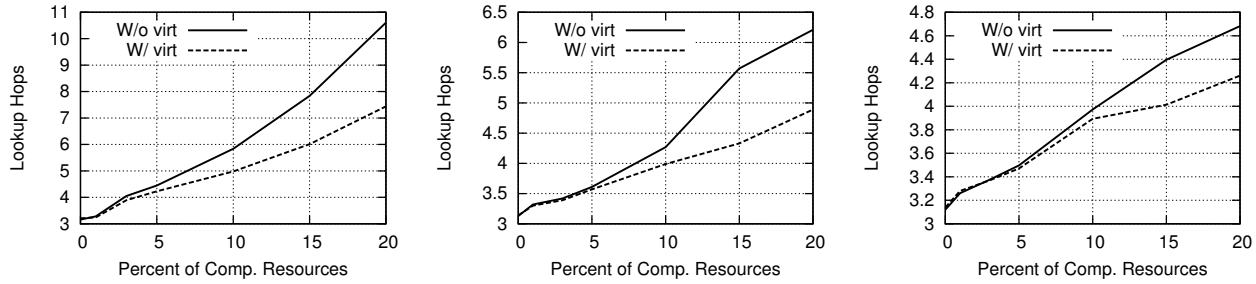
Recall that SybilControl only limits the number of Sybils that join the system, and does not provide much further protection against admitted puzzle-solving Sybils. However, by allowing honest nodes to support extra, virtual nodes, the influence of existing Sybils can be reduced (Section 4.2.3). With more virtual nodes, each Sybil node should claim a smaller portion of the keyspace, limiting the total influence of an adversary. In this section, we demonstrate the benefits of virtual nodes through simulations.

For our evaluations, we assume the adversary has a certain fraction of the computational resources, up to 20%. Note that since adversaries often do not control such large amounts of resources, some of these scenarios may not be commonplace in real-world systems. We allow the adversary to support a maximal number of



(a) No Replication (b) One Replication (c) Two Replications

Figure 6: Lookup success rates of SybilControl without virtual nodes and SybilControl where honest users support an average of 2 virtual nodes, while varying the fraction of computational resources the adversary controls.



(a) No Replication (b) One Replication (c) Two Replications

Figure 7: Lookup hop counts of SybilControl without virtual nodes and SybilControl where honest users support an average of 2 virtual nodes, while varying the fraction of computational resources the adversary controls.

Sybils, and compare SybilControl's performance with and without honest nodes supporting virtual nodes. We assume the distribution of honest user devices is a third smartphones, a third laptops, and a third desktops. We assume for simplicity that a laptop has twice the computational capabilities of a smartphone, and a desktop has three times the computational capacity of a smartphone. This allows honest users to support an average of 2 virtual nodes. We do not draw comparisons with regular Chord because it was not designed to withstand Sybil attacks. An adversary could support an unlimited number of Sybils in regular Chord, completely crippling the system.

We use the same performance metrics as described in Section 6.2. The frequency of churn-handling is set at every 30 seconds for all simulations. For this evaluation, we focus on lookup success, instead of integrity. We assume Sybil nodes drop all lookup queries, indicating lookup failure. Other works can be incorporated with SybilControl to improve lookup integrity. For resilient lookup protocols, nodes first perform one lookup with a time-to-live of 30 hops, and upon failure, perform two simultaneous redundant lookups. Again, we tested SybilControl using 0, 1, and 2 replications.

Figure 6 depicts the lookup success rate of SybilControl under Sybil attack, where the average honest user supports 2 virtual nodes. In each graph, a different number of replications was used. As predicted, use of virtual nodes improved lookup success in general. However, virtual nodes were most effective when using fewer replications. When an adversary controlled 20% of computational resources, use of virtual nodes improved no-replication SybilControl lookup success rate by about 10%, while two-replication SybilControl only improved by less than 1%. This is explainable since using large number of replications already improves success rate to near 100%, so there is little room for improvements from virtual nodes.

Figure 7 depicts the number of nodes contacted during SybilControl lookups using virtual nodes and under Sybil attack. As seen with lookup success rate, virtual nodes helped improve performance. It resulted in decreases in the number of nodes contacted, and the benefits were most apparent with fewer replications. With no-replication SybilControl, virtual node usage decreased the lookup hop count by about 3 nodes when 20% of computational resources was adversary-controlled. Under the same attack, two-replication SybilControl saw only a decrease of 0.4 hops, which is approximately a 10% improvement. An interesting observation is that virtual nodes do not seem to provide much benefit under smaller Sybil attacks, where the adversary controls only a small fraction of the computational resources. This is because the use of virtual nodes decreases the influence of each Sybil node slightly, and with few Sybil nodes, the total improvement is less noticeable.

These results show that virtual nodes do indeed reduce the influence of Sybil attacks. This signals that virtual node usage is a practical aspect to incorporate in SybilControl, along with replications and resilient lookups.

B. MULTI-HOP VERIFICATION CORRECTNESS

This section provides a sketch proof of the correctness of multi-hop verification. Let s be the puzzle time, p be the ping time, and one-way latencies be less than p. Challenge records are recorded for $2s + 4pd + 2p$ (Section 4.1.1). During multi-hop verification (Section 4.1.3), a node A can verify all but possibly the tail-end (oldest few) of neighbor B's sent challenges.

This tail-end loss is not an issue. B must use a challenge received from A no later than $2p$ after A sent the challenge. Thus, if A's oldest challenge was sent at time t, that challenge can verify a challenge from B that was created no later than $t + 2p$. So the largest size of B's tail-end is a timeframe of $2p$. With a network diameter of d, the largest size of the tail-end at a node across the network is $2pd$. The verification protocol requires another $2pd$ before reaching the node across the diameter, making the total tail-end size no larger than $4pd$. A can still verify the remaining challenges from the most recent $2s + 4pd + 2p - 4pd > 2s$ timeframe. Since this timeframe is larger than $2s$, this distant node's latest puzzle solution must have been solved using one of the verifiable challenges, and thus, A can verify even across the network. This does assume the verification path is acyclic, a reasonable assumption.