Putting Sybils on a Diet: Securing Distributed Hash Tables using Proofs of Space

Christoph U. Günther
cguenthe@ista.ac.at
ISTA

Krzysztof Pietrzak
pietrzak@ista.ac.at
ISTA

January 18, 2025

Abstract

Distributed Hash Tables (DHTs) are peer-to-peer protocols that act as components for more advanced applications. Recent examples, driven by blockchains, include decentralized storage networks (e.g., IPFS, Autonomi, Hypercore, and Swarm), data availability sampling, or Ethereum's peer discovery protocol.

In the blockchain setting, DHTs are susceptible to Sybil attacks, where an adversary disrupts the network by adding numerous malicious nodes. Preventing such attacks requires limiting the adversary's ability to create a large number of Sybil nodes. Surprisingly, the aforementioned applications implement no such measures. Seemingly, existing techniques are unsuitable for these applications.

For example, a straightforward technique described in the literature uses proof of work (PoW), where nodes periodically challenge their peers to solve computational puzzles. This approach, however, performs poorly in practice. Since these applications do not require honest nodes to have substantial computational power, the challenges cannot be too difficult. As a result, even moderately capable hardware can sustain many Sybil nodes.

In this work, we explore using Proof of Space (PoSp) to limit the number of Sybils in DHTs. While PoW proves that a node wastes computation, PoSp proves that a node wastes disk space. This aligns better with the resource needs of these applications: Many of them are storage-focused and rely on honest nodes contributing significant disk space to ensure functionality.

With this in mind, we propose a mechanism to limit Sybils where honest nodes dedicate a constant fraction of their disk space to PoSp. This ensures that an adversary cannot control a constant fraction of DHT nodes unless it contribute a constant fraction of the whole disk space contributed to the application overall. Since this is typically a considerable amount, Sybil attacks become economically unfeasible.

1 Introduction

Distributed hash tables (DHTs) offer an efficient key lookup functionality in a network of nodes. Each node is responsible for some part of the key space. Given a key, a lookup

query returns the node responsible for it. Amongst other things, this functionality suffices to implement the eponymous hash table.

To facilitate lookups, each node is connected to other nodes called *peers*. These connections are not arbitrary, but follow a protocol-specified network *structure*. Nodes have an *identifier* (often a random one) that dictates their place in the network. To be efficient, DHTs require a well-designed structure. In a network of n nodes, modern DHT constructions achieve lookups in $O(\log n)$ hops while only keeping track of $O(\log n)$ peers (e.g., [SMK+01, MM02]).

Initially, DHTs were used in peer-to-peer file-sharing systems, e.g., to enable trackerless torrents in BitTorrent [LN08]. Recently, blockchains opened new applications for DHTs. Storage networks such as IPFS [Ben14] (with incentives possible using Filecoin [Pro17]), Autonomi [aut], Hypercore [hyp], and Swarm [Tró24] use a DHT for content discovery. Ethereum employs a DHT for peer discovery in form of its discv4 protocol [disa] and its designated successor discv5 [disb]. In addition, DHTs are currently being researched in the context of data availability sampling [CGKR+24].

All of these applications use Kademlia [MM02] (or variations thereof). It is practically efficient and comes with redundancy features. So, to some degree, it is resilient against, e.g., nodes crashing or basic denial-of-service attack. However, thwarting more elaborate adversarial attacks is not one of its design goals.

1.1 Adversarial Attacks

In the blockchain context, DHTs are generally used without any central authority. So they need to withstand *Sybil* attacks [Dou02]. In such an attack, the adversary acts as multiple nodes with different identifiers. There are two aspects to Sybil attacks in DHTs:

1. The *number* of Sybil nodes (in short, *Sybils*) in the network. A Sybil attack injecting sufficiently many nodes may block lookups or return incorrect results, effectively rendering the DHT useless.

2. The *location* of Sybils in the network. If the adversary can freely choose identifiers, Sybil attacks become more powerful. Since DHTs are structured, the identifier of a node determine its position in the network. Thus, by strategically placing Sybils, the adversary can, e.g., *eclipse* (i.e., cut off) a specific honest node from other honest nodes.

The impact of an attack depends on the DHT construction. For example, Kademlia's [MM02] redundancy makes it somewhat resistant against Sybil attacks, but it does not provide any concrete guarantees. Nevertheless, it is reasonable to assume that an attack is unlikely to succeed if there are only few Sybils (Aspect 1) whose identifiers are uniformly distributed (Aspect 2).

There do exist prior works that propose Sybil-resistance mechanisms with provable guarantees. For example, randomly grouping nodes into committees where each committee acts as a singular DHT node (e.g., [AS06, JPS+18]). This is secure as long as there are not too many Sybils in the network (Aspect 1).

1.2 Proof of Work is not an Ideal Countermeasure

The literature proposes many defenses against Sybil attacks (we review them in § 2). The most promising approach is *proof of work* (PoW) because it is simple to implement, does not require any special infrastructure, and, in theory, defends against both aspects of Sybil attacks:

To limit the amount of Sybils (Aspect 1), nodes periodically challenge peers to solve computational puzzles. This bounds the number of adversarial nodes as a function of the adversary's computational resources [LMCB12, TF10, JPS+18].

To prevent the adversary from choosing identifiers (Aspect 2), an identifier is only valid if it comes with a solved PoW challenge.[1] This makes identifier generation more expensive and therefore harder for the adversary to strategically choose identifiers to, e.g., perform an eclipse attack [BM07, JPS+18].

In practice, we argue that PoW does not defend against Aspect 1, but still offers reasonable protection against Aspect 2. The reason is that there is a resource mismatch. PoW challenges require computation, while most of the above application require disk space (e.g., storage networks or data availability sampling).

Consider using PoW to limit the number of Sybils (Aspect 1). In the storage applications above, nodes use general-purpose hardware equipped with a significant amount of disk storage, but only average computational power. As a consequence, PoW challenges cannot be too difficult, as honest nodes could not pass them otherwise. Thus, an adversary focussing their resources only on computation may sustain a lot of Sybil nodes. So PoW does not meaningfully limit the number of Sybils. Apart from that, even if PoW challenges were reasonably difficult, then honest nodes would need to continually waste a lot of energy (and thereby money).

Nevertheless, using PoW to harden identifier generation (Aspect 2) works reasonably well. Since honest nodes only generate an identifier once, the PoW difficulty may be set to a relatively high level. So it may take a while for honest nodes to join the DHT, but makes it considerably harder for adversarial nodes to pick suitable identifiers.

1.3 Our Contribution: Proof of Space to Limit Sybils

In light of the above, we investigate using *proofs of space* (PoSp) [DFKP15] instead of PoW challenges to limit the number of Sybils (Aspect 1) in DHTs. On a high level, PoSp is the disk space analogue to PoW; it proves that a node is wasting a lot of disk space. Crucially, after a moderately expensive initialization phase, proofs are efficient to compute and verify, i.e., polylogarithmic in the size of the wasted space. Two PoSp constructions [Fis19, AAC+17] are already used in practice by Filecoin [Pro17] and Chia [chi].

Resource Synergy. As mentioned above, the services provided by the above applications mostly revolve around storing data. So participating nodes usually have a lot of disk

[1] For example, the identifier id must be accompanied by an x such that $h(\mathsf{id}, x) < D$ where h is a cryptographic hash function and D is the PoW difficulty parameter.

space at their disposal.[2] Thus, in contrast to PoW, the resource used to limit Sybils is the same resource that the application requires. In addition, the periodic PoSp challenges are computationally efficient, so nodes do not need to continually waste energy.

Our Constructions and their Guarantees. The Basic protocol we propose is simple and modular. It is compatible with existing DHT constructions, e.g., Kademlia [MM02]. In principle, it could be used in other peer-to-peer protocols, but we focus on the DHT use-case due to the synergy with applications. In a nutshell, Basic prescribes that every node provably wastes a fixed amount of disk space perpetually. Other nodes verify this by sending PoSp challenges to their peers periodically. The fixed amount of disk space is a global system parameter, e.g., 128 GiB. This bounds the number of Sybil nodes by a function of the adversary's total available disk space.

Our extension of Basic protocol, dubbed Virt, bounds the fraction of Sybils in the network as a function of adversarial and honest disk space. The idea is that every physical node acts as one or more *virtual nodes* [DKK+01] running Basic. Every physical node wastes, say, 1/10th of its disk space by running as many basic protocol instances as fit inside this space. The remaining 9/10ths are dedicated to the actual application, e.g., storing files in IPFS [Ben14].

Main Result (Cor. 1 of Thm. 2)**.** *In the protocol using virtual nodes, for any constant* $0 \leq \alpha < 1$*, the fraction of adversarial nodes* n_{adv} *of all nodes* n *is bounded by*

$$n_{\mathsf{adv}}/n < \alpha \quad \textit{if} \quad S_{\mathsf{adv}} < c \cdot S_{\mathsf{hon}}$$

where $S_{\mathsf{adv}}/S_{\mathsf{hon}}$ *denote the adversarial/honest disk space, and* $0 < c \leq \alpha/(1-\alpha)$ *is a constant depending on* α *as well as system parameters.*

Both approaches limit the number of Sybil nodes (Aspect 1). However, they cannot be used to limit identifier generation (Aspect 2) for subtle reasons, which we discuss in § 4.1. As argued above, PoW is a reasonable defense against Aspect 2 in practice.

Applications. The focus of this paper is to investigate PoSp as a mechanism to limit Sybils from a theoretical and also practical perspective. Our protocols are modular and may be used with existing DHTs. Below we propose two exemplary instantiations. For both, we recommend using Virt together with Filecoin's [Pro17] PoSp [Fis19] (it is practically efficient and has good concrete parameters, cf. § 6).

The first instantiation uses s/Kademlia [BM07], a more robust version of Kademlia [MM02] implementing PoW puzzles for identifiers and more robust lookup routing. While this does not give any provable Sybil-resistance guarantees, we assume that it should be reasonably robust in practice (and leave simulations as future work). The second uses Kademlia [MM02] together with committee-based approach of Jaiyeola et al. [JPS+18] discussed above. This gives provable guarantees, but we assume that the overhead of the committees may lead to performance problems in practice.

[2] Depending on the application, storage nodes could be different from the nodes participating in the DHT. In practice, however, this is usually not the case (e.g., in IPFS [Ben14] by default). Thus, we assume that DHT nodes have meaningful amounts of disk space available.

2 Related Work

2.1 Sybil Resistance Techniques

The literature on Sybil resistance in DHTs and distributed systems is diverse and multiple surveys exist [MK13, SM02, LSM06, UPS11]. We give an overview of some approaches.

We already described PoW defenses in § 1. Many works [LMCB12, TF10, JPS+18] require peers to periodically solve PoW puzzles. This gives a bound on the number of Sybils as a function of the adversarial computational power. Others [BM07, JPS+18] enforce that an identifier is only valid if it is accompanied by a PoW solution. This complicates attacks because the adversary cannot freely choose specific identifiers but must bruteforce them.

A downside of PoW approaches is that honest nodes also need to spend a lot of energy to solve puzzles—even if there is no adversarial activity. A line of work [GSY18, GSY19b, GSY19a, GSY20, GSY21] optimizes resource burning[3] to minimize the resource expenditure of honest parties. It is an open problem to design a DHT using these techniques [GSY20, GSY21].

Redundancy is a popular approach to increase robustness against benign faults and also Sybil resistance. These approaches usually assume that the fraction of Sybil nodes in the network is bounded (relying on, e.g., PoW [JPS+18]). Multiple works [AS04, FSY05, AS06, SY08, YKGK13, JPS+18] ensure redundancy using groups. The core idea is that nodes do not directly participate in the protocol, but instead are randomly grouped together. Each group collaboratively acts as a single node using Byzantine agreement protocols. Another avenue is redundant routing using disjoint paths [KT08, BM07].

Many approaches use information about social relationships [DLLKA05, LLK10, YKGF06, YGKX10]. These relationships are usually modeled as a graph where an edge between nodes exists if there is a trust relationship between the node operators in reality. The systems are Sybil-resistant as long as gaining trust in real life (e.g., by social engineering) is hard. These techniques only work in certain scenarios (e.g., instant messengers [LLK10]) and do not seem applicable to blockchain applications.

Other approaches use statistical tests to spot attacks [SAK+24], or inhibit identifier generation in ad-hoc ways. For example, setting the identifier of a node to the hash of its IP address [DKK+01], or avoiding peers with a low round-trip-time between each other [ZBV24].

Specific to Ethereum, one proposed solution to limit the amount of Sybils is using a *proof of validator* [KMNC23] scheme. On a high level, it ensures that only Ethereum validators, who are assumed to be honest, can join the DHT. The barrier to becoming a validator is high, i.e., staking at least 32 ETH[4]. So honest nodes that do not have that much money cannot join the DHT.

[3] Here, the resource is unspecified on purpose. It could be computation, money, solving CAPTCHAs, etc. They mention disk space but do not characterize it further. In our view, space is not a resource that is *burned*, but continually allocated instead.

[4] ≈125,000 USD, converted on 2024-12-15.

2.2 Attacks

Wang and Kangasharju [WK12] describe attacks against BitTorrent Mainline DHT and also give evidence that attacks were happening in 2010. There are two recent Sybil attacks on IPFS. The first is an eclipse attack by Prünster et al. [PMZ22]. They pre-generate and store $\approx 1.46 \cdot 10^{11}$ identifiers to strategically target any node. Then, they exploit how IPFS implemented the eviction policy of Kademlia [MM02] peers to eclipse nodes. The peer management has since been improved to make the attack more expensive. The second is a content-censorship attack [SAK+24]. It strategically places Sybil nodes around the hash of the content to be censored. Their proposed countermeasure uses statistical tests [SAK+24].

3 Preliminaries

For notation, let λ be the security parameter. $[n]$ denotes the set $\{1, 2, \ldots, n\}$ and $x \leftarrow\!\!\$\ \mathcal{X}$ sampling uniformly at random from $\mathcal{X}$. log is the logarithm base 2. We use standard Big-O notation and well-known shorthands such as poly, polylog, etc. As usual, a tilde, e.g., $\tilde{\Theta}(\cdot)$, hides polylogarithmic factors.

3.1 Distributed Hash Tables

Consider a network of n nodes where each node has an identifier $\mathsf{id} \in \mathcal{I}$. Further, consider a key space $\mathcal{K}$ that usually coincides with the identifier space (e.g., $\mathcal{I} = \mathcal{K} = \{0,1\}^{160}$). In a *distributed hash table* (DHT), each node is responsible for some part of the key space. The protocol $\mathsf{lookup}\colon \mathcal{K} \to \mathcal{I}$ takes a key $\mathsf{key} \in \mathcal{K}$ as input and outputs the identifier of the node responsible for key.

To make lookup possible, every node is connected to multiple other nodes called *peers*. A node choose its peers in a structured manner depending on the identifiers. A well-designed structure enables efficient lookups. Important metrics include the number of peers and the number of hops between nodes per lookup query.

Chord [SMK+01] is a simple, yet efficient construction. Each node has $O(\log n)$ peers and lookup needs at most $O(\log n)$ hops. Kademlia [MM02] enjoys the same asymptotic efficiency, but is more robust and the most popular DHT in practice. For interested readers, we give a high-level overview of Kademlia's design in App. A. We remark that constructions with better asymptotic efficiency exist [KK03, GV04], but they are not widely used to the best of our knowledge.

Our constructions use the DHT in a black-box manner. We abstract the DHT by the following functions below. Our definition only makes mild assumptions on how nodes manage their peers.

Definition 1 (Distributed Hash Table)**.** A DHT stores peers as a tuple $(\mathsf{id}, \mathsf{aux})$ where id is the peer's identifier and aux is auxiliary data (e.g., a public key or an IP address). It offers at least the following functions:

- $\mathsf{join}() \to (\mathsf{id}, \mathsf{aux})$: Joins the network and returns the own id and auxiliary data aux.
- $\mathsf{addPeer}(\mathsf{id}, \mathsf{aux})$: Adds the node id as a peer.

- $\mathsf{pingPeer}(\mathsf{id}, \mathsf{aux}) \to b$: Checks whether the peer id is online and returns a bit $b \in \{\mathsf{true}, \mathsf{false}\}$. This function is executed periodically by the DHT to ensure that all peers are still online.
- $\mathsf{lookup}(\mathsf{key}) \to (\mathsf{id}, \mathsf{aux})$: On input of a key key, returns the node responsible for key.

3.2 Proofs of Space

Proofs of space (PoSp)[5] were introduced by Dziembowski et al. [DFKP15]. Informally speaking, they are the disk space analogue to a PoW. A PoSp is a proof system that allows a prover to efficiently (in terms of computation and bandwidth) convince a verifier that they are wasting disk space.

Multiple follow-up proposals for PoSp protocols exist. Like the initial paper of Dziembowski et al. [DFKP15], many of them are based on graph labeling. This includes *proofs of catalytic space* [Pie19], which allow storing useful data (e.g., backups) instead of wasting space, and the *stacked expander graphs* construction [RD16]. The latter inspired the *stacked depth-robust graphs* construction [Fis19] which serves as the basis for Filecoin's [Pro17] PoSp. Using the more general notion of predecessor-robustness, Reyzin [Rey23] proves tighter bounds, paying special attention to constants. An entirely different approach rooted in function inversion is taken by Abusalah et al. [AAC+17] on which Chia [chi] is based.

In all of the above protocols, the prover and verifier share a short, common input seed, e.g., a public key. They use seed in the following two protocols:

In the initialization protocol, on input seed, the prover generates an output file file of large size N (say, 128 GiB) and stores it locally on disk. In some constructions (e.g., [Fis19, Rey23]), the prover additionally computes a commitment com to file and data produced in course of its derivation from seed. It sends the commitment com to the verifier who then checks that com is mostly correct—what "mostly correct" precisely means depends on the protocol. We assume that this check is non-interactive.[6]

In the online protocol, the verifier challenges the prover to demonstrate that they are still storing file in its entirety. They do so by sending a uniformly random challenge[7] chal to the prover. The prover responds with a proof π_{chal} which the verifier verifies with the help of com and seed. By periodically executing the online protocol, the verifier ensures that the prover is storing file for some span of time.

One essential requirement is efficiency, otherwise constructing PoSp is trivial.[8] While generating file takes $O(N)$ at best, all other computations, especially the ones of the verifier, must be fast, i.e., polylog(N) time. Similarly, com, π_{chal}, etc. must be of size polylog(N) at most.

The most important PoSp property is *Space-Hardness* which we state as in [Rey23]. Intuitively, it ensures the following: A cheating prover $\tilde{P}$ storing at most $(1 - \varepsilon_{\mathsf{space}}) \cdot N$

[5] In this paper, we always refer to proofs of *persistent* space [DFKP15] as opposed to proofs of *transient* space [ABFG14].

[6] Note that interactive prior works [DFKP15, RD16, Fis19, Pie19, Rey23] are all public coin, so applying the Fiat-Shamir transform is possible.

[7] We leave the set of all possible challenges implicit.

[8] For example, define $\mathsf{file} = h(\mathsf{seed})$ where h is a hash function with N-bit outputs with the proof $\pi_{\mathsf{chal}} = \mathsf{file}$ of size of N bits. This is inefficient as N is large.

bits *and* taking less than $(1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$ to answer a challenge will be detected with probability $\mathsf{pr}_{\mathsf{det}}$. We say a proof of space has a space gap $\varepsilon_{\mathsf{space}}$, time gap $\varepsilon_{\mathsf{time}}$, and single-query detection probability $\mathsf{pr}_{\mathsf{det}}$.

Another important property in our setting is *Nonreusability*. On a high level, this means the space of one PoSp with seed_1 cannot be reused for another one with different seed_2. Thus, storing both needs at least $(1-\varepsilon_{\mathsf{space}})\cdot 2N$ space. Intuitively, for constructions based on hash functions with a proof in the random oracle model (e.g., [Fis19]), Nonreusability holds because seed is used for domain separation. So each PoSp depends on an independent random oracle.

Definition 2 (Proof of Space)**.** A (non-interactive) proof of space is defined via four algorithms:

- $\mathsf{Init}(\mathsf{seed}) \to (\mathsf{file}, \mathsf{com}, \pi_{\mathsf{com}})$[9]
- $\mathsf{VerifyInit}(\mathsf{seed}, \mathsf{com}, \pi_{\mathsf{com}}) \to b$ with $b \in \{\mathsf{true}, \mathsf{false}\}$
- $\mathsf{Prove}(\mathsf{seed}, \mathsf{file}, \mathsf{com}, \mathsf{chal}) \to \pi_{\mathsf{chal}}$
- $\mathsf{Verify}(\mathsf{seed}, \mathsf{com}, \mathsf{chal}, \pi_{\mathsf{chal}}) \to b$ with $b \in \{\mathsf{true}, \mathsf{false}\}$

It fulfills the following properties:

Completeness Honest provers storing file always pass verification. That is, $\mathsf{true} \leftarrow \mathsf{VerifyInit}(\mathsf{seed}, \mathsf{com}, \pi_{\mathsf{com}})$ for all $(\mathsf{file}, \mathsf{com}, \pi_{\mathsf{com}}) \leftarrow \mathsf{Init}(\mathsf{seed})$, and $\mathsf{true} \leftarrow \mathsf{Verify}(\mathsf{seed}, \mathsf{com}, \mathsf{chal}, \pi_{\mathsf{cha}}$ for $\pi_{\mathsf{chal}} \leftarrow \mathsf{Prove}(\mathsf{seed}, \mathsf{file}, \mathsf{com}, \mathsf{chal})$.

Efficiency Suppose $|\mathsf{file}| = N$. Init runs in time $\mathsf{time}(\mathsf{Init}) \in \tilde{\Theta}(N)$ and all other algorithms in time $\mathrm{poly}(\lambda, \log N)$. Apart from file, all other outputs are of size $\mathrm{poly}(\lambda, \log N)$.

Soundness In the following, the PPT algorithm $\tilde{P}$ is a cheating prover and the protocol defines when a commitment com is "mostly correct".

Soundness of Initialization If $\tilde{P}$ outputs a commitment $\widetilde{\mathsf{com}}$ that is not mostly correct, $\mathsf{VerifyInit}(x, \widetilde{\mathsf{com}}, \tilde{\pi}_{\mathsf{com}}) \to \mathsf{false}$ except with $\mathrm{negl}(\lambda)$ probability.

Space-Hardness Suppose that $\widetilde{\mathsf{com}}$ is mostly correct. Then, with probability at least $\mathsf{pr}_{\mathsf{det}}$ over chal, if $\tilde{P}$ uses at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ space, it needs time of at least $(1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$ to compute a proof $\tilde{\pi}_{\mathsf{chal}}$ such that $\mathsf{Verify}(x, \widetilde{\mathsf{com}}, \mathsf{chal}, \tilde{\pi}_{\mathsf{chal}}) \to \mathsf{true}$.

Nonreusability Consider k PoSp with distinct seeds $\mathsf{seed}_1, \ldots, \mathsf{seed}_k$ and mostly correct commitments $\widetilde{\mathsf{com}}_1, \ldots \widetilde{\mathsf{com}}_2$ and supppose $\tilde{P}$ stores at most $(1-\varepsilon_{\mathsf{space}})\cdot k \cdot N$ bits. Then, except for $\mathrm{negl}(\lambda)$ probability, there exists a PoSp i where, with probability $\mathsf{pr}_{\mathsf{det}}$ over chal, $\tilde{P}$ needs at least $(1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$ to compute a proof $\tilde{\pi}_{\mathsf{chal}}$ such that $\mathsf{Verify}(x, \widetilde{\mathsf{com}}_i, \mathsf{chal}, \tilde{\pi}_{\mathsf{chal}}) \to \mathsf{true}$.

[9] For readers familiar with PoSp: In [AAC+17] com and π_{com} is empty.

4 Constructions to Limit Sybils

4.1 Basic Construction

The Basic DHT construction requires every node to continually store a PoSp file of a fixed size, e.g., $|\mathsf{file}| = N = 128$ GiB. To prevent cheating, every node periodically challenges its peers to prove that they are still storing file. Intuitively, this ensures that the number of Sybil nodes is bounded by

$$f < S_{\mathsf{adv}}/((1 - \varepsilon_{\mathsf{space}}) \cdot N) \tag{1}$$

where S_{adv} is the adversary's space and $\varepsilon_{\mathsf{space}}$ is the space gap of the PoSp. We will later formalize this in Thm. 1.

Basic uses a PoSp protocol, denoted by PoSp, and builds upon some existing DHT construction, denoted by DHT. Since nodes in DHT perform regular pings anyway (e.g., to check whether peers are still online), DHT is easily adapted. Basic is mostly identical to DHT modifying only join, addPeer, and pingPeer (as defined in Def. 1). It introduces the following global system parameters:

- N is the size of the PoSp each node must store.
- t_{ping} is the time between two pingPeer(id, aux) executions, i.e., it controls how often each peer is pinged.
- t_{timeout} is the time pingPeer waits for a response from the peer.
- κ the number of PoSp challenges per pingPeer.

Recall that a node is a tuple (id, aux) by Def. 1. The peer's identifier is id and aux is some auxiliary data (e.g., its IP address). In Basic, aux contains a PoSp commitment com with associated proof π_{com} and any auxiliary data required by DHT, denoted by $\mathsf{aux}_{\mathsf{DHT}}$.

A node joins the network using Basic.join[10] (Fig. 1). As part of this, it generates an identifier id according to DHT and then initializes a PoSp with id as input. It stores the resulting file file on disk and returns the id and auxiliary data.

Basic.join() $\to$ (id, aux):
01 DHT.join() $\to$ (id, $\mathsf{aux}_{\mathsf{DHT}}$) 02 PoSp.Init(id) $\to$ (file, com, π_{com}) 03 Store file on disk. 04 Return id and aux = $\{\mathsf{com}, \pi_{\mathsf{com}}\} \cup \mathsf{aux}_{\mathsf{DHT}}$.

Figure 1: Pseudocode of Basic.join.

A node adds another node as a peer using Basic.addPeer (Fig. 2) The function calls DHT.addPeer after performing two checks. First, it verifies that com is a valid commitment to a PoSp with input id. Second, it ensures that the peer is storing the PoSp by performing an initial pingPeer.

[10]We prefix functions with their protocol to avoid ambiguities, especially between Basic and DHT.

Basic.addPeer(id, aux):

01 Extract com and π_{com} from aux.
02 If PoSp.VerifyInit(id, com, π_{com}) $\rightarrow$ false, abort.
03 Await Basic.pingPeer(id, aux) $\rightarrow b$. If $b =$ false, abort.
04 Wait until t_{ping} time has passed since the previous ping.
05 DHT.addPeer(id, aux)

Figure 2: Pseudocode of Basic.addPeer.

After having added a peer, Basic.pingPeer (Fig. 3) is run periodically with an interval of t_{ping} time. pingPeer checks that the peer is online, and that it is still storing file associated with com. To this end, it sends κ uniformly random PoSp challenges chal to the peer and waits for the peers to answer with proofs. If the peer does not respond within time t_{timeout}, it is deemed offline. Else, all proofs π_{chal_i} are verified.

Basic.pingPeer(id, aux):

01 Extract com from aux.
02 Send κ uniformly random challenges $\mathsf{chal}_1, \ldots, \mathsf{chal}_\kappa$ to id.
03 Wait t_{timeout} time for a response $\pi_{\mathsf{chal}_1}, \ldots \pi_{\mathsf{chal}_\kappa}$:
04 If no response is received, return false.
05 Else, return $\bigwedge_{i\in[\kappa]}$ PoSp.Verify(id, com, $\mathsf{chal}_i, \pi_{\mathsf{chal}_i}$)

Figure 3: Pseudocode of Basic.pingPeer.

Note that Basic does not influence the choice of identifier. It relies on DHT.Init to generate id (Fig. 1, Line 1). So bruteforcing a specific identifier is as hard as in DHT. Recall that picking a specific identifier should be hard, as otherwise, e.g., eclipse attacks could be possible. The reason is that these attacks require identifiers to be strategically located in certain parts of the network structure. To this end, attacks often precompute (i.e., bruteforce) and store strategically-located identifiers in advance [PMZ22]. In practice, DHT could, e.g., impose a PoW of reasonable hardness on identifier generation [BM07, JPS+18] (ideally using a memory-hard function, e.g., Argon2 [BDK16]).

A benefit of Basic is that actually *using* identifiers is harder for adversaries. While they may precompute identifiers, whenever they want to use one, they need to initialize the PoSp first. This might prevent attacks or make them more difficult at least.

Remark. It may be tempting to make PoSp work double-duty and also use it as PoW scheme in the following way: Compute $(y, \mathsf{com}, \pi_{\mathsf{com}}) \leftarrow$ PoSp.Init(seed) and define id $=$ hash(seed, com). One might assume that Init must be computed for every identifier. This would make bruteforcing identifiers expensive. But this is not the case! The definition of PoSp (Def. 2) does not rule out the existence of two (or more) commitments com $\neq$ com′ such that

$$\mathsf{PoSp.VerifyInit}(\mathsf{seed}, \mathsf{com}, \pi_{\mathsf{com}}) = \mathsf{true}$$

and

$$\mathsf{PoSp.VerifyInit}(\mathsf{seed}, \mathsf{com}', \pi_{\mathsf{com}'}) = \mathsf{true}.$$

In fact, in existing constructions it is easy to find many distinct commitments that verify. Thus, apart from not achieving the intended goal, this modification even allows an attacker to store *one* PoSp but act as *two (or more)* identities (one derived using com, the other using com'). This would render all guarantees of Basic moot.

4.2 Virtual Nodes

Basic requires that nodes waste a fixed amount of disk space. As a consequence, the resulting guarantees, informally stated above in Eq. (1), only bound the total number of Sybils irrespective of the total number of nodes n. Ideally, we want a guarantee of the form

$$f/n < \alpha \quad \text{as long as} \quad S_{\mathsf{adv}} < c \cdot S_{\mathsf{hon}}$$

where S_{hon} is the space of all honest nodes combined, $0 < \alpha < 1$ is a constant, and c is also a constant that depends on system parameters and α. The following construction, called Virt, gives such a guarantee in Cor. 1, a consequence of Thm. 2.

The core idea of Virt is to account for the disk space of honest nodes using *virtual nodes* [DKK+01].[11] One physical node acts as one or more virtual nodes, where each virtual node runs one Basic instance. Thus, the number of virtual nodes of an honest node is related to its available disk space.

Since Virt uses Basic as a building block, it inherits its system parameters N, t_{ping}, t_{timeout}, and κ. In addition, Virt introduces the parameter $0 < \delta < 1$. It controls the fraction of disk space used for PoSp. The remaining $(1 - \delta)$ fraction of the space is used for the application of which the DHT is a part of (e.g., storing files in IPFS).

Suppose a physical node wants to participate in the DHT having S space. It participates as

$$\left\lceil \frac{\delta \cdot S}{N} \right\rceil \tag{2}$$

virtual nodes, each with a distinct identifier. Each of these virtual nodes runs one Basic protocol instance as described in the previous section § 4.1.

Note that rounding up in Eq. (2) implies that every physical node must have at least N bits of space. Otherwise, it does not have the resources to run even one Basic instance. In practice, it should have more space because it also needs to store application data.

5 Theoretical Perspective

Let us analyze the theoretical guarantees of Basic and Virt. Our goal is to bound the number of Sybils that are part of the DHT (i.e., that are connected to at least one honest node). To this end, we introduce a system model and specify parameter choices. Afterward, we combine the theoretical guarantees with existing Sybil-tolerance strategies.

So far, we have introduced a lot of parameters. All of them influence the guarantees, so we briefly summarize them in Fig. 4. Let us also recall the PoSp notation: $\varepsilon_{\mathsf{space}}$, $\varepsilon_{\mathsf{time}}$, and $\mathsf{pr}_{\mathsf{det}}$ quantify the guarantees of the PoSp. A cheating prover storing at most

[11] Using virtual nodes (also called virtual servers) in DHTs is not a new idea. Originally, they were introduced to alleviate load-balancing issue [DKK+01].

λ	Security parameter	$S^{(t)}_{\mathsf{hon}}$	Total honest space at time t
$n^{(t)}$	Number of nodes at time t	$S^{(t)}_{\mathsf{adv}}$	Adversarial space at time t
$f^{(t)}$	Number of Sybil nodes at time t	N	Prescribed PoSp size
$n^{(t)} - f^{(t)}$	Number of honest nodes at time t	δ	Fraction of space used by Virt
$\varepsilon_{\mathsf{space}}$	PoSp space gap	t_{ping}	$\mathsf{pingPeer}$ ping interval
$\varepsilon_{\mathsf{time}}$	PoSp time gap	t_{timeout}	$\mathsf{pingPeer}$ ping timeout
$\mathsf{pr}_{\mathsf{det}}$	PoSp detection probability	κ	# of challenges per ping

Figure 4: Notation summary.

$(1-\varepsilon_{\mathsf{space}}) \cdot N$ bits *and* taking less than $(1-\varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init})$[12] time to answer a challenge will be detected with probability $\mathsf{pr}_{\mathsf{det}}$.

5.1 System Model and Parameter Choices

Time is measured in discrete time steps, and messages between nodes are always delivered with a delay of at most Δ. Nodes do need not know Δ, nor do they need synchronized clocks. The only assumption we make is that Δ is not too large, namely that it is upper bounded by $\Delta \leq t_{\mathsf{timeout}}/2$.[13]

We consider a PPT adversary $\mathcal{A}$ that schedules message delivery (while respecting Δ), may attempt to join the DHT with $\mathrm{poly}(\lambda)$ many Sybil nodes that may arbitrarily deviate from the protocol, and has $S^{(t)}_{\mathsf{adv}}$ bits of disk space available at time t. We deem an adversarial node part of the DHT if it is connected to at least one honest node, and assume that an honest node removes a peer if it fails a ping. The cumulative space of all honest nodes at time t is denoted by $S^{(t)}_{\mathsf{hon}}$.

In terms of parameters, we require $t_{\mathsf{timeout}} \leq t_{\mathsf{ping}} < (1-\varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init})$.[14] Additionally, we set κ, the number of parallel PoSp challenges per ping, such that Verify detects a cheating node except with $\mathrm{negl}(\lambda)$ probability. A cheating peer storing at most $(1-\varepsilon_{\mathsf{space}}) \cdot N$ bits evades detection with probability at most $(1-\mathsf{pr}_{\mathsf{det}})^{\kappa}$. Thus, setting $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$ suffices since $(1-\mathsf{pr}_{\mathsf{det}})^{\lambda/\mathsf{pr}_{\mathsf{det}}} \leq e^{-\lambda}$ is exponentially small in λ.

Let us briefly explain these choices. t_{timeout} is set such that $\mathcal{A}$ cannot solve a PoSp challenge on-demand. Similarly, the choice of t_{ping} ensures that the adversary cannot use *the same* space (i.e., $(1-\varepsilon_{\mathsf{space}}) \cdot N$ bits) for *multiple* different Sybil identities by re-initializing the PoSp between pings of different nodes. Last, the upper bound on Δ is necessary, otherwise honest nodes could not respond to pings in time: Suppose $\Delta > t_{\mathsf{timeout}}/2$, then $\mathcal{A}$ could delay the PoSp challenge by Δ and the response by Δ. Since $2\Delta > t_{\mathsf{timeout}}$, the ping would time out.

[12] Note that $\mathsf{time}(\mathsf{Init})$ in wall-clock time is not known as it depends on the adversary's hardware. So it must be estimated, ideally assuming powerful hardware. For example, as we elaborate in § 6, the PoSp deployed by Filecoin has a latency of $\approx 35\,\mathrm{s}$, even assuming highly-parallel ASICs.

[13] Plain Kademlia [MM02] also implicitly assumes that message delay is bounded. Indeed, if a peer does not respond within t_{timeout} time, it is assumed offline.

[14] In practice, such frequent pings might not be feasible, primarily due to bandwidth constraints. In § 6.2 propose and analyze a different pinging schedule where $t_{\mathsf{ping}} \gg \mathsf{time}(\mathsf{Init})$ in expectation.

5.2 Security Analysis

Theorem 1 (Basic). *For the system model and parameter choices given in § 5.1,* Basic *bounds the number of Sybil nodes at time $t > 2t_{\mathsf{ping}}$ by*

$$f^{(t)} < \frac{\max_{i\in[t-2t_{\mathsf{ping}},t]} S^{(i)}_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N} \tag{3}$$

except with $\mathrm{negl}(\lambda)$ *probability.*

Proof. Consider any honest node and any of its Sybil peers at time t. The honest node has sent the last challenge to id in the time interval $[t-t_{\mathsf{ping}}, t]$. Similarly, the second-to-last challenge to id was issued at the time $t_c \in [t-2t_{\mathsf{ping}}, t-t_{\mathsf{ping}}]$. Note that t_c is always defined since Basic. addPeer challenges the peer once before adding it.

Since the peer is still connected at time t, it must have passed all past challenges, including the one at time t_c. There are three reasons why the peer passed the challenge:

1. $\mathcal{A}$ dedicated $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits of its space to the PoSp with seed id at some time
$$t_s \in [t_c, t_c + t_{\mathsf{timeout}}]. \tag{4}$$
2. The commitment com for id is incorrect.
3. The challenge failed to detect the cheating.

Note that Reasons 2 and 3 occur with at most $\mathrm{negl}(\lambda)$ probability by the choice of t_{timeout} and κ, and the Soundness of PoSp. Replacing t_c in Eq. (4) gives us $t_s \in [t-2t_{\mathsf{ping}}, t]$ since $t_{\mathsf{timeout}} \le t_{\mathsf{ping}}$.

Now recall that $\mathcal{A}$ has $f^{(t)} \in \mathrm{poly}(\lambda)$ Sybil nodes in total, each of which is connected to an honest node. Thus, all of them must have passed a challenge in time interval $[t-2t_{\mathsf{ping}}, t]$. By a union bound, the probability that *any* Sybil passed due to Reasons 2 and 3 is at most $\mathrm{negl}(\lambda)$. Consequently, due to the Nonreusability of PoSp, $\mathcal{A}$ must have dedicated more than $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits to every Sybil node at some point in the interval $[t-2t_{\mathsf{ping}}, t]$. Since changing the space dedicated to one Sybil to another requires at least $(1-\varepsilon_{\mathsf{time}})\cdot \mathsf{time}(\mathsf{Init}) > t_{\mathsf{ping}}$ time, the theorem follows from a pigeonhole argument. □

Theorem 2 (Virt). *For the system model and parameter choices given in § 5.1,* Virt *bounds the fraction of Sybil nodes at time $t > 2t_{\mathsf{ping}}$ by*

$$\frac{f^{(t)}}{n^{(t)}} < \frac{S_{\mathsf{adv}}}{S_{\mathsf{adv}} + (1-\varepsilon_{\mathsf{space}})\cdot\delta\cdot S_{\mathsf{hon}}}$$

except for $\mathrm{negl}(\lambda)$ *probability. Here,* $S_{\mathsf{adv}} = \max_{i\in[t-2t_{\mathsf{ping}},t]} S^{(i)}_{\mathsf{adv}}$ *and* $S_{\mathsf{hon}} = \min_{i\in[t-t_{\mathsf{ping}},t]} S^{(i)}_{\mathsf{hon}}$.

Proof. Due to Thm. 1, $f^{(t)}$ is upper bounded by Eq. (3). So we only need to lower bound $n^{(f)}$ to prove the theorem.

Denote the space of a physical, honest node i at time t by $S^{(t,i)}_{\mathsf{hon}}$ with $S^{(t)}_{\mathsf{hon}} = \sum_i S^{(t,i)}_{\mathsf{hon}}$. By construction, at every time t, the number of *potentially-available* honest nodes is lower bounded by

$$\sum_i \left\lceil \frac{\delta\cdot S^{(i,t)}_{\mathsf{hon}}}{N} \right\rceil \ge \sum_i \frac{\delta\cdot S^{(t,i)}_{\mathsf{hon}}}{N} = \frac{\delta\cdot S^{(t)}_{\mathsf{hon}}}{N}.$$

Note that $\mathcal{A}$ cannot prevent honest nodes from responding to pings since $\Delta < t_{\mathsf{timeout}}/2$ by assumption.

We emphasize that the above is *not* a lower bound on the number of honest nodes at time t. The reason is that newly-joined nodes are not immediately added as peers, but only after t_{ping} time (Fig. 2, Line 4). Therefore, the actual number of honest nodes at time t is lower bounded by

$$n^{(t)} - f^{(t)} \geq \min_{i \in [t - t_{\mathsf{ping}}, t]} \frac{\delta \cdot S_{\mathsf{hon}}^{(i)}}{N}. \tag{5}$$

Combining Eqs. (3) and (5) and rearranging yields the desired inequality. □

Both theorems talk about the maximum/minimum of disk space within a time span of length at most $2t_{\mathsf{ping}}$. If the disk space is large enough, it is reasonable to assume that disk space fluctuates at most by a constant within this time span.

Definition 3 (r-Fluctuation of Disk Space)**.** For a constant $r \geq 1$, we say the disk space r-fluctuates if

$$\max_{i \in [t - 2t_{\mathsf{ping}}, t]} S_{\mathsf{adv}}^{(i)} \leq r \cdot S_{\mathsf{adv}}^{(t)}$$

and

$$\min_{i \in [t - 2t_{\mathsf{ping}}, t]} S_{\mathsf{hon}}^{(i)} \geq \frac{1}{r} \cdot S_{\mathsf{hon}}^{(t)}$$

for every time t.

We will use this definition to simplify Thm. 2. In addition, recall that we wanted a statement of the form $f^{(t)}/n^{(t)} < \alpha$ as long as $S_{\mathsf{adv}}^{(t)} < c \cdot S_{\mathsf{hon}}^{(t)}$. Rearranging Thm. 2 gives such guarantees, as the following corollary shows.

Corollary 1 (of Thm. 2)**.** *For the system model and parameter choices given in § 5.1 and assuming r-fluctuation of disk space (Def. 3),* Virt *bounds the fraction of Sybil nodes at time $t > 2t_{\mathsf{ping}}$ by*

$$\frac{f^{(t)}}{n^{(t)}} < \alpha$$

for any constant $0 \leq \alpha < 1$ that satisfies

$$S_{\mathsf{adv}}^{(t)} < \frac{\alpha}{1 - \alpha} \cdot (1 - \varepsilon) \cdot \delta \cdot \frac{1}{r^2} \cdot S_{\mathsf{hon}}^{(t)}$$

except for $\mathrm{negl}(\lambda)$ *probability.*

5.3 Applications

As surveyed in § 2, prior works describe Sybil-resistance techniques that have provable guarantees. Generally, these guarantees only hold if the fraction of Sybil nodes $f/n < \alpha$ is bounded by a constant α. This bound is either simply assumed or justified using a PoW mechanisms. By Cor. 1, we may use these Sybil-resistance techniques in combination with Virt instead. We illustrate this by giving two examples of such Sybil-resistance techniques.

First, consider the construction due to Jaiyeola et al. [JPS+18]. Nodes are assigned to groups according to their identifier. Each group effectively acts as a single node in a non-Sybil-resistant DHT protocol, e.g., Chord [SMK+01]. Groups collaboratively decide on their actions by using a Byzantine agreement protocol. To be secure, every group needs an honest majority which, amongst other conditions, requires $f/n < \alpha$ for a certain α.

Second, Halo [KT08] is a technique that boosts the success rate of lookup queries. The idea is to perform lookups along disjoint paths by predicting the location of peers of the target node. Halo only makes black-box use of the underlying DHT, which is Chord [SMK+01] (though the techniques should transfer to other DHTs). To theoretically analyze and simulate Halo, an assumption of the form $f/n < \alpha$ is necessary.

6 Practical Considerations

6.1 Instantiating the Proof of Space

Many PoSp constructions exist [DFKP15, RD16, AAC+17, Fis19, Pie19, Rey23], but only two are practically efficient enough to run on off-the-shelf hardware. Both follow very different approaches. Filecoin's [Pro17] PoSp is based on *stacked depth-robust graphs* (SDR-PoSp) [Fis19], while Chia's [chi] follows a *function inversion* approach (FI-PoSp) [AAC+17]. For the DHT use-case, SDR-PoSp is better suited for two reasons.

Parallel vs. Sequential Time. An important issue is whether $\mathsf{time}(\mathsf{Init})$ measures *sequential* or *parallel* time. Sequential time is the total amount of computation steps required by a sequential algorithm. This captures the *cost* of Init, but does not rule out that parallelism may speed it up. In contrast, parallel time captures an adversary with unlimited parallelism, so it measures the *latency* of Init.

The security guarantees of Basic and Virt rely on latency. This rules out FI-PoSp since inverting a function is parallelizable. In contrast, SDR-PoSp achieves space-hardness against parallel time adversaries [Fis19]. Thus, SDR-PoSp is suitable for our use-case.

Parameter Guarantees. Irrespective of the above, SDR-PoSp achieves better asymptotic parameters and also better practical security. In the best case, the space gap $\varepsilon_{\mathsf{space}}$ and time gap $\varepsilon_{\mathsf{time}}$ should both be small.

Reyzin [Rey23] observes the following: FI-PoSp's guarantees are not ideal because its space gap $\varepsilon_{\mathsf{space}}$ grows as the PoSp size N increases. In contrast, SDR-PoSp fares better, allowing arbitrary $\varepsilon_{\mathsf{space}}$. For the concrete implementation deployed by Filecoin, $\varepsilon_{\mathsf{space}} = 0.2$ and $\mathsf{pr}_{\mathsf{det}} = 0.1$. With respect to $\varepsilon_{\mathsf{time}}$, there are two analyses giving different guarantees.

Fisch [Fis19] considers parallel time adversaries and shows that SDR-PoSp allows for arbitrary $\varepsilon_{\mathsf{space}}$ at the cost of $\varepsilon_{\mathsf{time}}$ increasing as $\varepsilon_{\mathsf{space}}$ decreases. For Filecoin's concrete parameters above, this gives $\varepsilon_{\mathsf{time}} < 54/55 \approx 0.98$ leading to $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init}) \approx 35\,\mathrm{s}$ assuming that the adversary has very efficient hardware (i.e., ASICs) [GN]. By only counting sequential time, Reyzin [Rey23] proves that SDR-PoSp achieves arbitrary $\varepsilon_{\mathsf{space}}$ and $\varepsilon_{\mathsf{time}}$. This allows for a better time gap; in the case of Filecoin, $\varepsilon_{\mathsf{time}} < 4/5 = 0.8$.

To summarize, SDR-PoSp achieves arbitrary space gap $\varepsilon_{\mathsf{space}}$ (irrespective of the analysis) which is better than FI-PoSp. For these, Reyzin's [Rey23] analysis allows for a tighter time gap $\varepsilon_{\mathsf{time}}$ than Fisch's analysis [Fis19]. Unfortunately, Reyzin's analysis only captures the total cost (i.e., sequential time) of Init, but our application cares about latency (i.e., parallel time).

As a consequence, for Filecoin's SDR-PoSp we may assume a space gap $\varepsilon_{\mathsf{space}} = 0.2$, detection probability $\mathsf{pr}_{\mathsf{det}} = 0.1$, and time gap $\varepsilon_{\mathsf{time}} = 54/55$ with $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init}) = 35\,\mathrm{s}$. Note that these parameters are pessimistic and also not accurate for adversaries storing significantly less than $(1 - 0.2)N$. Asymptotically, if $\varepsilon_{\mathsf{space}}$ increases, the other parameters improve (i.e., $\mathsf{pr}_{\mathsf{det}}$ increases and $\varepsilon_{\mathsf{time}}$ decreases).

Combining Proofs of Space. Due to hardware constraints, the PoSp size N is limited in practice. For example, Filecoin's implementation of SDR-PoSp uses $N = 16$ or $32\,\mathrm{GiB}$. This ensures that running Init is possible on off-the-shelf hardware. Consequently, to achieve larger PoSp sizes, multiple smaller ones need to be combined.

Combining k sub-PoSp of size N results in a larger PoSp of size $k \cdot N$. Naively, challenging this PoSp requires k challenges in parallel, one to each sub-PoSp. While this ensures that the detection probability does not decrease, the bandwidth is increased by a multiplicative factor of k. To save bandwidth, suppose we challenge the PoSp as follows: Sample $i \leftarrow\!\!\$\ [k]$ and then challenge the ith sub-PoSp.

Claim 1. *If the sub-PoSp has parameters $\varepsilon_{\mathsf{space}}$ and $\mathsf{pr}_{\mathsf{det}}$, then the combined construction's parameters are $\varepsilon'_{\mathsf{space}} = 2 \cdot \varepsilon_{\mathsf{space}}$ and $\mathsf{pr}'_{\mathsf{det}} = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$.*

Proof. By an averaging argument [AB09, Lem. A.8], if the adversary stores at most $(1 - \varepsilon'_{\mathsf{space}}) \cdot k \cdot N$ bits of the combined PoSp, at least $0.5 \cdot \varepsilon'_{\mathsf{space}} \cdot k$ sub-PoSp have at most $(1 - 0.5 \cdot \varepsilon'_{\mathsf{space}}) \cdot N$ bits dedicated to them. Setting, $\varepsilon'_{\mathsf{space}} = 2 \cdot \varepsilon_{\mathsf{space}}$ leads to $\mathsf{pr}'_{\mathsf{det}} = 0.5 \cdot \varepsilon'_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}} = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$. □

We emphasize that neither $\varepsilon'_{\mathsf{space}}$ nor $\mathsf{pr}'_{\mathsf{det}}$ depend on k. For Filecoin's (pessimistic) parameters stated above, $\varepsilon'_{\mathsf{space}} = 0.4$ and $\mathsf{pr}'_{\mathsf{det}} = 0.02$ for a single challenge. As usual, the detection probability may be boosted by using multiple parallel challenges (again, this is irrespective of k). For example, for 10 challenges, $(1 - \mathsf{pr}'_{\mathsf{det}})^{10} \approx 0.2$.

6.2 Lowering Bandwidth and Optimizing Pings

Bandwidth is a limited resource, and also sensitive to spikes. First, we show how to reduce bandwidth spikes induced by the PoSp commitments. Second, the parameters assumed in the theoretical analysis may require impractically large amounts of bandwidth. In short, this is due to the high ping frequency and overwhelming detection probability per ping. In the following, we will discuss an alternative approach that allows for infrequent pings and lower detection probability.

Commitment Bandwidth Spikes. Initially transmitting π_{com} incurs a large bandwidth spike when using SDR-PoSp. It achieves an exponentially small soundness error of $2^{-\lambda}$, but π_{com} is quite large (for Filecoin, to the order of 290λ). One solution is allowing a higher soundness error, but that is not ideal. Another is splitting π_{com} into smaller chunks

$\pi_{\mathsf{com}}^{(1)}, \ldots, \pi_{\mathsf{com}}^{(k)}$ and transmitting them chunk-by-chunk over a longer period of time. This smoothes out bandwidth spikes. SDR-PoSp enjoys the property that each chunk can be checked individually and reduces the soundness error by a factor $2^{-\lambda/k}$ [Rey23]. So the confidence in com's correctness increases gradually.

Optimizing pings. For the parameters stated in § 5.1, pings need large amounts of bandwidth. There are two reason for that.

First, we assume that κ (i.e., the number of parallel PoSp challenges per ping) is sufficiently large. Recall that we set $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$ to ensure an overwhelming detection probability of $1 - e^{-\lambda}$ overall. We do this for the sake of the proof because it allows us to reliably detect (and thus remove) a peer as soon as it starts cheating. In practice, however, DHTs run for a long amount of time, and also use heuristics to manage peers. For example, Kademlia [MM02] prefers peers with long uptimes, so peers who fail challenges—even if only occasionally—will be disconnected. In other words, trust is hard to gain, but easy to lose. Thus, a lower detection probability should suffice in practice (we will formalize this momentarily), reducing bandwidth.

Second, we assume frequent pings with a deterministic, sufficiently small interval t_{ping} This ensures that the adversary cannot use the same space for two different identities by re-initializing it between pings. A more practical (but harder to analyze theoretically) approach is sampling pinging peers probabilistically. At every point in time, a peer will be pinged with uniform probability $\mathsf{pr}_{\mathsf{ping}}$. This may be implemented by sampling the time of the next ping from the geometric distribution parameterized by $\mathsf{pr}_{\mathsf{ping}}$. Intuitively, since the adversary cannot predict pings, it cannot reliably switch between different identities. This allows for more infrequent pings, reducing bandwidth.

Theoretical Analysis. The latter two bandwidth improvements (i.e., lowering κ and pinging probabilistically) require a new theoretical analysis to give bounds on the number of adversarial nodes in the network. Our analysis splits the time into *epochs*. Each epoch lasts T time steps, and epochs start/end at the same time for all honest nodes. As a consequence, compared to the model we used in § 5.1, we need the additional assumption that honest nodes have (roughly) synchronized clocks.

When a node learns of a new potential peer, it starts challenging it probabilistically as described in the previous paragraph. It will only add it as a DHT peer after having challenged it for an *entire* epoch at least. If a (potential) peer fails a challenge, it is removed immediately.

For simplicity, we assume that $\mathcal{A}$'s space S_{adv} is fixed and large enough to sustain $f = \frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}$ Sybil nodes without meaningfully cheating (i.e., as much as the parameters of the PoSp allow). We now answer the following question: *How long should epochs be (i.e., the value of T) to ensure that the adversary has at most $2f$ Sybil nodes?*

Theorem 3. *Let S_{adv} be the $\mathcal{A}$'s disk space and define $f = \frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}$.*

If epochs last $T = \frac{6\lambda}{\mathsf{pr}_{\mathsf{ping}}\cdot\mathsf{pr}_{\mathsf{det}}}$ time, in every epoch, at most $2f$ Sybil nodes are part of the DHT except with $\mathrm{negl}(\lambda)$ *probability.*

In other words, Thm. 3 allows us to reduce $\mathsf{pr}_{\mathsf{ping}}$ and $\mathsf{pr}_{\mathsf{det}}$ at the expense of increasing T, i.e., making epochs longer. Even if both are constant, T stays linear in the security

parameter λ. This is an improvement since we previously required an overwhelmingly large (in λ) detection probability.

Proof of Thm. 3. In every epoch, $\mathcal{A}$ starts with at most $\text{poly}(\lambda)$ Sybils, and each Sybil may be connected to at most $\text{poly}(\lambda)$ honest nodes. Define W as the set of *winning* Sybils, i.e., those that are connected to at least one honest node at the end of an epoch (and thereby will be part of the DHT in the next epoch). To analyze W, we will define two disjoint sets W_1 and W_2 such that $W \subseteq W_1 \cup W_2$. By bounding the size of both, we will show that $|W| \leq |W_1| + |W_2| = 2f + 0$ except with $\text{negl}(\lambda)$ probability. Note that this is equivalent to the theorem statement.

Define W_1 as the set of Sybils to which $\mathcal{A}$ dedicated more than $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits for more than $T/2$ time steps. By a pigeonhole argument, it follows that

$$W_1 \leq \frac{T \cdot S_{\mathsf{adv}}}{T/2 \cdot (1 - \varepsilon_{\mathsf{space}}) \cdot N} = 2f.$$

Define $W_2 \subseteq W \setminus W_1$, i.e., the set of winning Sybils to which $\mathcal{A}$ *did not dedicate* more than $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits for more than $T/2$ time steps. We will show that $\Pr[W_2 = 0] \geq 1 - \text{negl}(\lambda)$. To do so, we introduce two bad events B_1, B_2, each occurring with $\text{negl}(\lambda)$ probability, and prove that $\Pr[W_2 = 0] \leq 1 - (\Pr[B_1] + \Pr[B_2 \wedge \neg B_1])$.

- B_1 is the bad event that at least one PoSp commitment is not "mostly correct" (as defined in Def. 2), i.e., at least one Sybil in W_2 managed to cheat during its initialization. By the PoSp's Soundness of Initialization and a union bound over all Sybils in W_2, $\Pr[B_1] \leq \text{poly}(\lambda) \cdot \text{negl}(\lambda)$ which is negligible.

- B_2 is the event that the connection of any honest node to any Sybil in W_2 is still active at the end of the epoch. Note that any Sybil in W_2 has at most $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ for at least $T/2$ steps, otherwise it would be in W_1 instead. For these $T/2$ time steps, assuming that the PoSp's commitment is mostly correct (i.e., B_1 did not occur), the Sybil will fail a PoSp challenge with probability at least $\mathsf{pr}_{\mathsf{det}}$ by the PoSp's Soundness. Since it is connected to at least one honest peer, it will be pinged/challenged with probability $\mathsf{pr}_{\mathsf{ping}}$ at every point in time. Define the event B_2' as the event that a specific honest node is still connected to a specific Sybil in W_2. From the above, we deduce that

  $$\Pr[B_2' \wedge \neg B_1] \leq (1 - \mathsf{pr}_{\mathsf{ping}} \cdot \mathsf{pr}_{\mathsf{det}})^{T/2} \leq 2^{-3\lambda}$$

  where the last inequality holds due to $T = \frac{6\lambda}{\mathsf{pr}_{\mathsf{ping}} \cdot \mathsf{pr}_{\mathsf{det}}}$. Using a union bound over the at most $\text{poly}(\lambda)$ number of Sybils with at most $\text{poly}(\lambda)$ number of connections each, we get that

  $$\Pr[B_2 \wedge \neg B_1] \leq \text{poly}(\lambda) \cdot \text{poly}(\lambda) \cdot \Pr[B_2' \wedge \neg B_1] \leq 2^{-\lambda}.$$

Putting the above together, we conclude that

$$\Pr[W_2 = 0] \geq 1 - (\Pr[B_1] + \Pr[B_2 \wedge \neg B_1]) \geq 1 - \text{negl}(\lambda)$$

which also completes the proof of the theorem. □

6.3 Useful Space

Instead of wasting disk space, our protocol could be adapted to store useful data. Proofs of catalytic space [Pie19] allow this with two caveats: Accessing this data takes as long as initializing the PoSp, and efficiently updating the data requires knowledge of it. So this is only useful to store long-term data, e.g., backups.

Let us remark that Filecoin [Pro17]'s infrastructure is sufficient to implement a protocol like Virt. This is because the Filecoin network globally tracks how much disk space each storage node commits to Filecoin. The committed data is not necessarily random, but may be useful data stored on behalf of a client.

7 Conclusion and Future Work

We have laid the theoretical groundwork for using PoSp as a mechanism to limit Sybils. Our constructions are simple and come with provable guarantees. Practical deployments seem feasible; we have given performance recommendations.

Two directions for future work are immediate: First, implementing our constructions and measuring their performance overheads. Second, simulating attacks against our constructions to verify their theoretical guarantees.

Other, more far-fetched directions are the following: First, investigating (Sybil-resistant) DHT constructions that are practical, yet also easy to analyze theoretically. Second, finding alternatives to virtual nodes since this approach cannot scale arbitrarily due to, e.g., bandwidth limitations. Other approaches to heterogeneity that are not using virtual nodes exist (e.g., [BBKK10]). Can they be combined with PoSp to get a Sybil-resistant DHT?

References

[AAC+17] Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 357–379. Springer, Cham, December 2017.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity*. Cambridge University Press, 4 2009.

[ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 538–557. Springer, Cham, September 2014.

[AS04] Baruch Awerbuch and Christian Scheideler. Group spreading: A protocol for provably secure distributed name service. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, pages 183–195, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[AS06] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust dht. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, page 318–327, New York, NY, USA, 2006. Association for Computing Machinery.

[aut] Autonomi.

[BBKK10] Marcin Bienkowski, André Brinkmann, Marek Klonowski, and Miroslaw Korzeniowski. Skewccc+: A heterogeneous distributed hash table. In Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah, editors, *Principles of Distributed Systems*, pages 219–234, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302, 2016.

[Ben14] Juan Benet. Ipfs - content addressed, versioned, p2p file system, 2014.

[BM07] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8, 2007.

[CGKR+24] Mikel Cortes-Goicoechea, Csaba Kiraly, Dmitriy Ryajov, Jose Luis Muñoz-Tapia, and Leonardo Bautista-Gomez. Scalability limitations of kademlia dhts when enabling data availability sampling in ethereum, 2024.

[chi] Chia.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Berlin, Heidelberg, August 2015.

[disa] Node discovery protocol.

[disb] Node discovery protocol v5.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. *SIGOPS Oper. Syst. Rev.*, 35(5):202–215, oct 2001.

[DLLKA05] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant dht routing. In Sabrina de Capitani di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Computer Security – ESORICS 2005*, pages 305–318, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Dou02] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 251–260, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Fis19] Ben Fisch. Tight proofs of space and replication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 324–348. Springer, Cham, May 2019.

[FSY05] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine attacks. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms – ESA 2005*, pages 803–814, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[GN] Irene Giacomelli and Luca Nizzardo. Filecoin proof of useful space - technical report.

[GSY18] Diksha Gupta, Jared Saia, and Maxwell Young. Proof of work without all the work. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, ICDCN '18, New York, NY, USA, 2018. Association for Computing Machinery.

[GSY19a] Diksha Gupta, Jared Saia, and Maxwell Young. Peace through superior puzzling: An asymmetric sybil defense. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1083–1094, 2019.

[GSY19b] Diksha Gupta, Jared Saia, and Maxwell Young. Resource-competitive sybil defenses, 2019.

[GSY20] Diksha Gupta, Jared Saia, and Maxwell Young. Resource burning for permissionless systems (invited paper). In Andrea Werneck Richa and Christian Scheideler, editors, *Structural Information and Communication Complexity*, pages 19–44, Cham, 2020. Springer International Publishing.

[GSY21] Diksha Gupta, Jared Saia, and Maxwell Young. Bankrupting sybil despite churn. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 425–437, 2021.

[GV04] A.-T. Gai and L. Viennot. Broose: a practical distributed hashtable based on the de-bruijn topology. In *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.*, pages 167–174, 2004.

[hyp] Hypercore protocol.

[JPS+18] Mercy O. Jaiyeola, Kyle Patron, Jared Saia, Maxwell Young, and Qian M. Zhou. Tiny groups tackle byzantine adversaries. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1030–1039, 2018.

[KK03] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In M. Frans Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II*, pages 98–107, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[KMNC23] George Kadianakis, Mary Maller, Andrija Novakovic, and Suphanat Chunhapanya. Proof of validator: A simple anonymous credential scheme for ethereum's dht, 2023.

[KT08] Apu Kapadia and Nikos Triandopoulos. Halo: High-assurance locate for distributed hash tables. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008*. The Internet Society, 2008.

[LLK10] Chris Lesniewski-Laas and M. Frans Kaashoek. Whānau: A sybil-proof distributed hash table. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 8, USA, 2010. USENIX Association.

[LMCB12] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. Sybilcontrol: practical sybil defense with computational puzzles. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, page 67–78, New York, NY, USA, 2012. Association for Computing Machinery.

[LN08] Andrew Loewenstern and Arvid Norberg. Bittorrent enhancement proposal 5: Dht protocol, January 2008.

[LSM06] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack. 2006.

[MK13] Aziz Mohaisen and Joongheon Kim. The sybil attacks and defenses: A survey, 2013.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 53–65, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Pie19] Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 59:1–59:25. LIPIcs, January 2019.

[PMZ22] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. Total eclipse of the heart – disrupting the InterPlanetary file system. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3735–3752, Boston, MA, August 2022. USENIX Association.

[Pro17] Protocol Labs. Filecoin: A decentralized storage network, 2017.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285. Springer, Berlin, Heidelberg, October / November 2016.

[Rey23] Leonid Reyzin. Proofs of space with maximal hardness. Cryptology ePrint Archive, Report 2023/1530, 2023.

[SAK+24] Srivatsan Sridhar, Onur Ascigil, Navin Keizer, François Genon, Sébastien Pierre, Yiannis Psaras, Etienne Rivière, and Michał Król. Content censorship in the interplanetary file system. In *Proceedings 2024 Network and Distributed System Security Symposium*, NDSS 2024. Internet Society, 2024.

[SM02] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 261–269, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, aug 2001.

[SY08] Jared Saia and Maxwell Young. Reducing communication costs in robust peer-to-peer networks. *Information Processing Letters*, 106(4):152–158, 2008.

[TF10] Florian Tegeler and Xiaoming Fu. Sybilconf: Computational puzzles for confining sybil attacks. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–2, 2010.

[Tró24] Viktor Trón. The book of swarm, February 2024.

[UPS11] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of dht security techniques. *ACM Comput. Surv.*, 43(2), February 2011.

[WK12] Liang Wang and Jussi Kangasharju. Real-world sybil attacks in bittorrent mainline dht. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 826–832, 2012.

[YGKX10] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: a near-optimal social network defense against sybil attacks. *IEEE/ACM Trans. Netw.*, 18(3):885–898, jun 2010.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *SIGCOMM Comput. Commun. Rev.*, 36(4):267–278, aug 2006.

[YKGK13] Maxwell Young, Aniket Kate, Ian Goldberg, and Martin Karsten. Towards practical communication in byzantine-resistant dhts. *IEEE/ACM Trans. Netw.*, 21(1):190–203, feb 2013.

[ZBV24] Yunqi Zhang and Shaileshh Bojja Venkatakrishnan. Kadabra: Adapting kademlia for the decentralized web. In Foteini Baldimtsi and Christian Cachin, editors, *Financial Cryptography and Data Security*, pages 327–345, Cham, 2024. Springer Nature Switzerland.

A Kademlia

In Kademlia [MM02] identifiers are $\mathsf{id} \in \{0,1\}^\ell$ (e.g., $\ell = 160$); the key space is also $\{0,1\}^\ell$. The distance between two nodes is $\mathsf{dist}(\mathsf{id}_1, \mathsf{id}_2) = \mathsf{id}_1 \oplus \mathsf{id}_2$ which is a symmetric metric.

Every node stores its peers in a routing table. It consists of ℓ buckets, each storing up to k peers to ensure replication (e.g., $k = 20$). Peers in the ith bucket have a distance of $(2^i, 2^{i+1} - 1)$ to the node. Buckets covering larger distances are usually full, so an eviction strategy is necessary, e.g., a least-recently seen one that never evicts online peers [MM02].

$\mathsf{lookup}(\mathsf{key})$ returns the k closest nodes to key iteratively. First, a node selects α peers from its routing table that are closest to key. Then, it asks them for their k peers closest to key. From these responses, the node again picks the α closest and iterates until it has found the k closest nodes to key. This requires at most $O(\log n)$ time. Note that α controls the concurrency of lookups (e.g., $\alpha = 3$).

A new node joins by connecting to an existing node and performing $\mathsf{lookup}(\mathsf{id})$ where id is its desired identifier. This allows it to find its place in the network and other peers.