# 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

May 5, 2025

**Abstract**

Distributed Hash Tables (DHTs) are peer-to-peer protocols that serve as building blocks for more advanced applications. Recent examples, motivated by blockchains, include decentralized storage networks (e.g., IPFS), data availability sampling, or Ethereum's peer discovery protocol.

In the blockchain context, DHTs are vulnerable to Sybil attacks, where an adversary compromises the network by joining with many malicious nodes. Mitigating such attacks requires restricting the adversary's ability to create a lot of Sybil nodes. Surprisingly, the above applications take no such measures. Seemingly, existing techniques are unsuitable for the proposed applications.

For example, a simple technique proposed in the literature uses proof of work (PoW), where nodes periodically challenge their peers to solve computational challenges. This, however, does not work well in practice. Since the above applications do not require honest nodes to have a lot of computational power, challenges cannot be too difficult. Thus, even moderately powerful hardware can sustain many Sybil nodes.

In this work, we investigate using Proof of Space (PoSp) to limit the number of Sybils DHTs. While PoW proves that a node wastes computation, PoSp proves that a node wastes disk space. This aligns better with the resource requirements of the above applications. Many of them are related to storage and ask honest nodes to contribute a substantial amount of disk space to ensure the application's functionality.

With this synergy in mind, we propose a mechanism to limit Sybils where honest nodes dedicate a fraction of their disk space to PoSp. This guarantees that the adversary cannot control a constant fraction of all DHT nodes unless it provides a constant fraction of whole the disk space contributed to the application in total. Since this is typically a significant amount, attacks become economically expensive.

## 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. The efficiency of DHTs relies on their 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.[1] Recently, blockchains opened new applications for DHTs. Storage networks such as IPFS [Ben14] (with incentives possible using Filecoin [Pro17]), Autonomi[2], or Swarm[3] use a DHT for content discovery. Ethereum employs a DHT for peer discovery in form of its discv4 protocol and its designated successor discv5.[4] In addition, DHTs are currently being investigated 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:

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).

[1]`http://www.bittorrent.org/beps/bep_0005.html`
[2]`https://autonomi.com/`
[3]`https://www.ethswarm.org/`
[4]`https://github.com/ethereum/devp2p`

## 1.2 Proof of Work

The literature proposes many defenses against Sybil attacks (we review them in § 2). A popular and universal approach is *proof of work* (PoW). 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.[5] 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].

## 1.3 PoW does not Limit Sybils in Practice

In the above applications, honest nodes run general-purpose hardware that is equipped with a lot of *disk space* (e.g., storage networks of data availability sampling). Thus, honest nodes only have average computational power, so the PoW difficulty must be chosen with this in mind.

Limiting the amount of Sybils (Aspect 1) relies on periodic challenges, so the difficulty cannot be too high. 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 in DHTs.

Using PoW to harden identifier generation (Aspect 2) works reasonably well nonetheless. 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.

Apart from security considerations, periodic PoW challenges waste a lot of energy even if the system is not under attack. This does not work for many ecosystems which are trying to move to more environmentally-friendly designs.

## 1.4 Our Contribution: Proof of Space to Limit Sybils

PoW does not work because of a *resource mismatch*: Honest nodes are incentivized to invest in disk space, while an adversary invests in computational power. To fix this, 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 [chi24].

[5] For example, the identifier $\mathsf{id}$ 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.

**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 space at their disposal.[6] 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 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 in fixed time intervals. 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)**.** *For any constant $0 \leq \alpha < 1$, Virt bounds the fraction of adversarial nodes $f$ of all nodes $n$ is bounded by $f/n < \alpha$ if $S_{\mathsf{adv}} < c \cdot S_{\mathsf{hon}}$ where $S_{\mathsf{adv}}/S_{\mathsf{hon}}$ denote the adversarial/honest disk space, and the constant $0 < c \leq \alpha/(1-\alpha)$ depends on $\alpha$, the concrete guarantees of the PoSp construction, and the fraction of space honest nodes waste.*

Our results so far considered deterministically-issued challenges. We also analyze a probabilistic challenge schedule where a node challenges a peer with uniform, constant probability at every point in time. We show that this is still security, while at the same time reducing the amount of communication, making our schemes practical.

**Practicality & Applications.** While we focus on the theory behind using PoSp as a Sybil-limiting mechanism, we also estimate performance in practice. For this, we consider the PoSp deployed by Filecoin [Pro17] Setting the PoSp size to 128 GiB per node, a node sends and receives $\approx 16$ KiB every minute to a peer in expectation ($\approx 2.2$ kbps); for more details see § 6.2.

In terms of applications, we give two exemplary deployments using Virt with Filecoin's PoSp. The first uses s/Kademlia [BM07] as a DHT. It is 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 DHT is Kademlia [MM02] combined with the committee-based approach of Jaiyeola et al. [JPS+18] (cf. § 2). This gives provable guarantees, but we assume that the overhead of the committees may lead to performance problems in practice.

[6]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

Sybil-resistance in DHTs and distributed systems has been covered by multiple surveys [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. Others [BM07, JPS+18] enforce that an identifier is only valid if it is accompanied by a PoW solution.

A downside of PoW approaches is that honest nodes also need to spend a lot of energy to solve puzzles, even without adversarial activity. A line of work [GSY18, GSY19b, GSY19a, GSY20, GSY21] optimizes resource burning[7] to minimize the resource waste of honest parties. Design a DHT with these techniques is an open problem [GSY20, GSY21].

Redundancy is a common approach to increase robustness against benign faults and Sybil-resistance. These approaches usually assume that the fraction of Sybils 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 participate in the protocol directly, but instead are randomly grouped together. Each group jointly 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 use ad-hoc measures. For example, setting the identifier to the hash of a node's IP address [DKK+01], or avoiding peers with short round-trip-times between each other [ZBV24].

Specific to Ethereum, one proposed solution is *proof of validator* [KMNC23]. It ensures that only Ethereum validators, who are assumed to be honest, can join the DHT. There are two downsides: This is not a universal approach and does not apply to, e.g., IPFS [Ben14]. The barrier to becoming a validator is high (staking 32 ETH[8]). So users without a lot of money cannot help running the DHT.

### 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.

[7]The resource 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.

[8]$\approx 50,000$ USD on 2025-04-11.

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

$\lambda$ is the security parameter, $[n] = \{1, 2, \ldots, n\}$, and log the logarithm base 2. $x \leftarrow\!\!\$\ \mathcal{X}$ denotes sampling uniformly at random from $\mathcal{X}$. We use Big-O notation and shorthands such as poly and negl. Tilde, e.g., $\tilde{\Theta}$, hides polylog. factors.

### 3.1 Distributed Hash Tables

The network consists of $n$ nodes, each having an identifier $\mathsf{id} \in \mathcal{I}$. The key space is $\mathcal{K}$; it usually equals $\mathcal{I}$ (e.g., $\mathcal{I} = \mathcal{K} = \{0,1\}^{160}$). In a *distributed hash table* (DHT), each node is responsible for some part of $\mathcal{K}$. The protocol $\mathsf{lookup}\colon \mathcal{K} \to \mathcal{I}$ takes a key $\mathsf{key} \in \mathcal{K}$ and outputs the identifier of the node responsible for $\mathsf{key}$.

To make $\mathsf{lookup}$ possible, every node is connected to multiple other nodes called *peers*. A node choose its peers in a structured manner depending on the identifiers. Important metrics include the number of peers and the number of hops between nodes per $\mathsf{lookup}$ query.

Kademlia [MM02] the most popular DHT in practice. Each node has $O(\log n)$ peers and $\mathsf{lookup}$ needs at most $O(\log n)$ hops. Constructions with better asymptotic efficiency exist [KK03, GV04], but they are not widely used to our knowledge.

Our constructions use the DHT in a black-box manner. We abstract the DHT by the following functions below.

**Definition 1** (Distributed Hash Table)**.** A DHT stores peers as a tuple $(\mathsf{id}, \mathsf{aux})$ where $\mathsf{id}$ is the peer's identifier and $\mathsf{aux}$ 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 DHT; returns the own $\mathsf{id}$ and auxiliary data $\mathsf{aux}$.
- $\mathsf{addPeer}(\mathsf{id}, \mathsf{aux})$: Adds the node $\mathsf{id}$ as a peer.
- $\mathsf{pingPeer}(\mathsf{id}, \mathsf{aux}) \to b$: Checks whether the peer $\mathsf{id}$ 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.
- $\mathsf{lookup}(\mathsf{key}) \to (\mathsf{id}, \mathsf{aux})$: Given key $\mathsf{key}$ returns the node responsible for $\mathsf{key}$.

### 3.2 Proofs of Space

Proofs of space (PoSp) [DFKP15][9] are, informally, the disk space analogue to a PoW. A PoSp is a proof system that allows a prover to efficiently (in terms of computation and

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

bandwidth) convince a verifier that they are wasting disk space. The prover and verifier share a short, common input $\mathsf{seed}$, e.g., a public key. They use $\mathsf{seed}$ in the following two protocols:

In the initialization protocol, the prover generates a file $\mathsf{file}$ (which depends on $\mathsf{seed}$) of large size $N$ (say, 128 GiB) and stores it on disk. In some constructions (e.g., [Fis19, Rey23]), the prover computes a commitment $\mathsf{com}$ to $\mathsf{file}$ (and other intermediate data). It sends the commitment $\mathsf{com}$ to the verifier who then checks that $\mathsf{com}$ is mostly correct—what "mostly correct" means depends on the construction. We assume that this is non-interactive by using Fiat-Shamir.

In the online protocol, the verifier challenges the prover to show that they are still storing $\mathsf{file}$. They do so by sending a uniformly random challenge[10] $\mathsf{chal}$ to the prover. The prover responds with a proof $\pi_{\mathsf{chal}}$ that the verifier checks using $\mathsf{com}$ and $\mathsf{seed}$. By periodically executing the online protocol, the verifier ensures that the prover is storing $\mathsf{file}$ for this timespan.

One requirement is efficiency, otherwise constructing PoSp is trivial.[11] While generating $\mathsf{file}$ takes $O(N)$, everything else must be fast, i.e., $\mathrm{polylog}(N)$, especially the verifier. Similarly, $\mathsf{com}$, $\pi_{\mathsf{chal}}$, etc. must be of size $\mathrm{polylog}(N)$.

The most important property is *Space-Hardness* (stated as in [Rey23]). A cheating prover storing at most $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ 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 the PoSp has a space gap $\varepsilon_{\mathsf{space}}$, time gap $\varepsilon_{\mathsf{time}}$, and detection probability $\mathsf{pr}_{\mathsf{det}}$.

Another property is *Nonreusability.* This means that the space of one PoSp with $\mathsf{seed}_1$ cannot be reused for another one with different $\mathsf{seed}_2$. Thus, storing both needs at least $(1 - \varepsilon_{\mathsf{space}}) \cdot 2N$ space. For hash-function-based constructions proven secure in the random oracle model (e.g., [Fis19]), Nonreusability holds because $\mathsf{seed}$ is used for domain separation.

**Definition 2** (Proof of Space)**.** A proof of space is defined via:

- $\mathsf{Init}(\mathsf{seed}) \to (\mathsf{file}, \mathsf{com}, \pi_{\mathsf{com}})$[12]
- $\mathsf{VerifyInit}(\mathsf{seed}, \mathsf{com}, \pi_{\mathsf{com}}) \to b$ 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 $\mathsf{file}$ 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 $|\mathsf{file}| = N$. $\mathsf{Init}$ runs in time $\mathsf{time}(\mathsf{Init}) \in \tilde{\Theta}(N)$ and all other algorithms in time $\mathrm{poly}(\lambda, \log N)$. Apart from $\mathsf{file}$, all other outputs are of size $\mathrm{poly}(\lambda, \log N)$.

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

[11] 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.

[12] For readers familiar with PoSp: In [AAC+17] $\mathsf{com}$ and $\pi_{\mathsf{com}}$ is empty.

**Soundness** In the following, the PPT algorithm $\tilde{P}$ is a cheating prover and the protocol defines when a commitment $\mathsf{com}$ 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 $\mathsf{chal}$, 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 $\mathsf{chal}$, $\tilde{P}$ needs at least $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init})$ to compute a proof $\tilde{\pi}_{\mathsf{chal}}$ such that $\mathsf{Verify}(x, \widetilde{\mathsf{com}}_i, \mathsf{chal}, \tilde{\pi}_{\mathsf{chal}}) \to \mathsf{true}$.

# 4 Constructions to Limit Sybils

## 4.1 Basic Construction

The Basic DHT construction requires nodes to continually store a PoSp file of fixed size, e.g., $|\mathsf{file}| = N = 128$ GiB.[13] To prevent cheating, nodes periodically challenges their peers to prove that they are still storing file. Informally, this bounds the number of Sybil nodes as $f < S_{\mathsf{adv}}/((1 - \varepsilon_{\mathsf{space}}) \cdot N)$ where $S_{\mathsf{adv}}$ is the adversary's space and $\varepsilon_{\mathsf{space}}$ is the space gap of the PoSp (cf. Thm. 1).

Basic uses a PoSp protocol, denoted by PoSp, and a DHT construction, denoted by DHT. Basic is mostly identical to DHT modifying only join, addPeer, and pingPeer (cf. Def. 1). It introduces the following global system parameters: $N$ is the size of the PoSp each node must store; $t_{\mathsf{ping}}$ is the time between two $\mathsf{pingPeer}(\mathsf{id}, \mathsf{aux})$ executions, i.e., it controls how often each peer is pinged; $t_{\mathsf{timeout}}$ is the time pingPeer waits for a response from the peer; and $\kappa$ the number of PoSp challenges per pingPeer.

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

A node joins using Basic.join[14] (Fig. 1). It generates an identifier id according to DHT and then initializes a PoSp with id as seed. It stores the resulting file file on disk and returns the id and auxiliary data.

A node adds 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.

Every peer is periodically pinged with an interval of $t_{\mathsf{ping}}$ time using Basic.pingPeer (Fig. 3). It checks that the peer is online and that it is still storing file associated with com. To this end, it sends $\kappa$ uniformly random PoSp challenges $\mathsf{chal}_i$ to the peer and

[13] Users without sufficient space can still query the DHT as clients, but will not be used for routing.

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

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

Figure 1: Pseudocode of Basic.join.

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

Figure 2: Pseudocode of Basic.addPeer.

waits for proofs. If the peer does not respond within time $t_{\mathsf{timeout}}$, it is deemed offline. Else, all proofs $\pi_{\mathsf{chal}_i}$ are verified.

```
Basic.pingPeer(id, aux):
01 Extract com from aux.
02 Send $\kappa$ uniformly random challenges $\mathsf{chal}_1, \dots, \mathsf{chal}_\kappa$ to id.
03 Wait $t_{\mathsf{timeout}}$ time for a response $\pi_{\mathsf{chal}_1}, \dots \pi_{\mathsf{chal}_\kappa}$:
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.[15] It relies on DHT.Init to generate id (Fig. 1, Line 1). So bruteforcing a specific identifier is as hard as in DHT, which should be moderately-hard to make targeted attacks harder. Attacks often precompute (i.e., bruteforce) identifiers. Basic makes *using* these harder as the PoSp needs to be initialized first.

## 4.2 Virtual Nodes

Basic requires that nodes waste a fixed amount of disk space. So the resulting guarantees only bound the total number of Sybils irrespective of the total number of nodes $n$. Ideally, we want to guarantee $f/n < \alpha$ as long as $S_{\mathsf{adv}} < c \cdot S_{\mathsf{hon}}$ where $S_{\mathsf{hon}}$ is the space of all

[15]One may be tempted to deriving id from the PoSp commitment com, but this is insecure. While it seems that Init must be computed for commitment and thus every identifier, this is not the case. PoSp do not rule out the existence of multiple commitments $\mathsf{com}_1 \neq \cdot \neq \mathsf{com}_n$ that pass VerifyInit and are easy to compute.

| | | | |
|---|---|---|---|
| $\lambda$ | 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 $t$ | $\delta$ | Fraction of space used by Virt |
| $\varepsilon_{\mathsf{space}}$ | PoSp space gap | $t_{\mathsf{ping}}$ | pingPeer ping interval |
| $\varepsilon_{\mathsf{time}}$ | PoSp time gap | $t_{\mathsf{timeout}}$ | pingPeer ping timeout |
| $\mathsf{pr}_{\mathsf{det}}$ | PoSp detection probability | $\kappa$ | # of challenges per ping |

Figure 4: Notation summary.

honest nodes combined, $0 < \alpha < 1$ is a constant, and $c$ is also a constant that depends on system parameters and $\alpha$. The construction Virt gives such a guarantee.

Crucially, Virt captures the total disk space of honest nodes using *virtual nodes* [DKK+01].[16] One physical node acts as one or more virtual nodes, where each virtual node runs one Basic instance. The number of virtual honest nodes linearly scales with the physically-available disk space.

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

Suppose a physical node has $S$ space. It participates as $\left\lceil \frac{\delta \cdot S}{N} \right\rceil$ virtual nodes, each running Basic with a distinct identifier. Note that rounding up in implies that every physical node must have at least $N$ bits of space for PoSp. Otherwise, it does not have the resources to run even one Basic instance. In practice, it should have sufficiently more to also store application data.

## 5 Theoretical Perspective

We now analyze the theoretical guarantees of Basic and Virt. Our goal is bounding the number of Sybils in the DHT (i.e., 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.

We recap all parameters introduced so far in Fig. 4. Also, recall the PoSp space-hardness: A cheating prover storing at most $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits *and* taking less than $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init})$[17] time to answer a challenge will be detected with probability of at least $\mathsf{pr}_{\mathsf{det}}$.

### 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 $\Delta$. Nodes do need not know $\Delta$, nor do they need synchronized

[16]Virtual nodes (virtual servers) were originally used for load-balancing in DHTs [DKK+01].

[17]Note that time(Init) in wall-clock time is not known as it depends on the adversary's hardware. Conservative estimates for Filecoin's PoSp yield a latency of $\approx 35\,\mathrm{s}$ [GN].

clocks. The only assumption we make is that $\Delta$ is not too large, namely that it is upper bounded by $\Delta \leq t_{\mathsf{timeout}}/2$.[18]

We consider a PPT adversary $\mathcal{A}$ that schedules message delivery (respecting $\Delta$), may join the DHT with poly($\lambda$) many Sybil nodes (arbitrarily deviating from the protocol), and has $S_{\mathsf{adv}}^{(t)}$ bits of disk space at time $t$. A Sybil node is *part of the DHT* if it is connected to at least one honest node. We assume that honest nodes removes peers that fail a ping, and that the cumulative space of all honest nodes at time $t$ is $S_{\mathsf{hon}}^{(t)}$.

In terms of parameters, we require $t_{\mathsf{timeout}} \leq t_{\mathsf{ping}} < (1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init})$.[19] Further, we set the number of PoSp challenges per ping $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$. Thus, pingPeer detects a cheating node except with negl($\lambda$) probability.[20]

Let us briefly explain these settings. $t_{\mathsf{timeout}}$ ensures $\mathcal{A}$ cannot solve a PoSp challenge on-demand. Similarly, $t_{\mathsf{ping}}$ 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. Last, the upper bound on $\Delta$ 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 and the response by $\Delta$ time each. Since $2\Delta > t_{\mathsf{timeout}}$, the ping would time out.

## 5.2 Security Statements

**Theorem 1** (Basic). *For the system model and parameters as in § 5.1,* Basic *bounds the number of Sybil nodes at time* $t > 2t_{\mathsf{ping}}$ *by* $f^{(t)} < \frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}$ *except for* negl($\lambda$) *probability. Here,* $S_{\mathsf{adv}} = \max_{i\in[t-2t_{\mathsf{ping}},t]} S_{\mathsf{adv}}^{(i)}$.

**Theorem 2** (Virt). *For the system model and parameter as in § 5.1,* Virt *bounds the fraction of Sybil nodes at time* $t > 2t_{\mathsf{ping}}$ *by* $f^{(t)}/n^{(t)} < \frac{S_{\mathsf{adv}}}{S_{\mathsf{adv}}+(1-\varepsilon_{\mathsf{space}})\cdot\delta\cdot S_{\mathsf{hon}}}$ *except for* negl($\lambda$) *probability. Here,* $S_{\mathsf{adv}} = \max_{i\in[t-2t_{\mathsf{ping}},t]} S_{\mathsf{adv}}^{(i)}$ *and* $S_{\mathsf{hon}} = \min_{i\in[t-t_{\mathsf{ping}},t]} S_{\mathsf{hon}}^{(i)}$.

The proofs are in App. A. 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 little within this time span.

Another way to view Thm. 2 is given in the following corollary. It tells us which gap between honest and adversarial space is necessary to ensure $f/n < \alpha$ for whatever fixed $\alpha$ is desired.

**Corollary 1** (of Thm. 2). *Let* $0 \leq \alpha < 1$. *Further, let* $S_{\mathsf{adv}}$, $S_{\mathsf{hon}}$, *system model, and parameters as in Thm. 2. Then, except for* negl($\lambda$) *probability,* Virt *bounds the fraction of Sybil nodes at any time* $t > 2t_{\mathsf{ping}}$ *by* $\frac{f^{(t)}}{n^{(t)}} < \alpha$ *as long as* $S_{\mathsf{adv}} < \frac{\alpha}{1-\alpha} \cdot (1 - \varepsilon_{\mathsf{space}}) \cdot \delta \cdot S_{\mathsf{hon}}$.

## 5.3 Applications

Prior works describe Sybil-resistance techniques that have provable guarantees. These guarantees only hold if the fraction of Sybil nodes $f/n < \alpha$ is bounded by a constant.

[18] Kademlia [MM02] also implicitly assumes that message delay is bounded. If a peer does not respond within $t_{\mathsf{timeout}}$ time, it is assumed offline.

[19] In practice, such frequent pings might not be feasible due to bandwidth constraints. In § 6.2 we analyze a probabilistic pinging schedule with $t_{\mathsf{ping}} \gg \mathsf{time}(\mathsf{Init})$.

[20] A cheating peer storing at most $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits evades detection with probability at most $(1 - \mathsf{pr}_{\mathsf{det}})^{\kappa} \leq e^{-\lambda}$.

This bound is either simply assumed or justified using a PoW mechanism. By Cor. 1, we may use $\mathsf{Virt}$ instead. We give two examples.

Jaiyeola et al. [JPS+18] assign nodes to groups according to their identifier. Each group effectively acts as a single node in a non-Sybil-resistant DHT protocol. Groups collaboratively decide on their actions by via a Byzantine agreement protocol. To be secure, every group needs an honest majority which, amongst other conditions, requires $f/n < \alpha$ for a certain $\alpha$.

Second, Halo [KT08] boosts the success rate of $\mathsf{lookup}$ 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 techniques should transfer to other DHTs). To theoretically analyze and simulate Halo, the assumption $f/n < \alpha$ is necessary.

## 6 Practical Considerations

### 6.1 Instantiating the Proof of Space

Many PoSp exist [DFKP15, RD16, AAC+17, Fis19, Pie19, Rey23], but only two are deployed in practice. Both follow different approaches. Filecoin's [Pro17] PoSp labels *stacked depth-robust graphs* (SDR-PoSp) [Fis19], while Chia's [chi24] 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 time required by a sequential algorithm. This captures the *cost* of $\mathsf{Init}$, but does not rule out that parallelism may speed it up. In contrast, parallel time captures an adversary with unlimited parallelism, so it measures *latency*.

The security guarantees of $\mathsf{Basic}$ and $\mathsf{Virt}$ rely on latency. This rules out FI-PoSp since computing a function table is parallelizable. In contrast, SDR-PoSp achieves space-hardness against parallel time adversaries [Fis19], so it is suitable.

**Parameter Guarantees.** Ideally, the space gap $\varepsilon_{\mathsf{space}}$ and time gap $\varepsilon_{\mathsf{time}}$ should both be small. Reyzin [Rey23] observes: FI-PoSp's space gap $\varepsilon_{\mathsf{space}}$ grows as the PoSp size $N$ increases, which is not ideal. 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 $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init}) > 35\,\mathrm{s}$ assuming powerful ASICs [GN]. By only counting sequential time, Reyzin [Rey23] proves that SDR-PoSp achieves arbitrary $\varepsilon_{\mathsf{space}}$ and $\varepsilon_{\mathsf{time}}$. This yields 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 $\mathsf{Init}$, but our application cares about latency (i.e., parallel time). As a consequence, we assume Filecoin's parameters as in [Fis19]: $\varepsilon_{\mathsf{space}} = 0.2$, $\mathsf{pr}_{\mathsf{det}} = 0.1$, $\varepsilon_{\mathsf{time}} = 54/55$, and $(1 - \varepsilon_{\mathsf{time}}) \cdot \mathsf{time}(\mathsf{Init}) > 35\,\mathrm{s}$.

**Combining Proofs of Space.** Due to memory constraints, the PoSp size $N$ is limited in practice. For example, Filecoin uses $N = 16$ or $32\,\mathrm{GiB}$. To achieve larger PoSp sizes, multiple smaller ones need to be combined.

Combining $k$ sub-PoSp of size $N$ results in a PoSp of size $k \cdot N$. Naively challenging this PoSp requires $k$ parallel challenges, one to each sub-PoSp. While this conserves the detection probability, bandwidth increased by $\times k$.

Instead, suppose we sample $i \leftarrow_{\$} [k]$ and then challenge the $i$th sub-PoSp. If the sub-PoSp has parameters $\varepsilon_{\mathsf{space}}$ and $\mathsf{pr}_{\mathsf{det}}$, then the combined PoSp has parameters $\varepsilon'_{\mathsf{space}} = 2 \cdot \varepsilon_{\mathsf{space}}$ and $\mathsf{pr}'_{\mathsf{det}} = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$.[21] Note that neither $\varepsilon'_{\mathsf{space}}$ nor $\mathsf{pr}'_{\mathsf{det}}$ depend on $k$. For Filecoin's parameters, $\varepsilon'_{\mathsf{space}} = 0.4$ and $\mathsf{pr}'_{\mathsf{det}} = 0.02$ for a single challenge. The detection probability may be boosted using parallel challenges (e.g., 18 challenges yield $1 - (1 - \mathsf{pr}'_{\mathsf{det}})^{18} > 0.3$).

## 6.2 Optimizing Bandwidth

**Commitment Bandwidth Spikes.** Initial transmission of $\pi_{\mathsf{com}}$ may cause a large bandwidth spike when using SDR-PoSp: To get a soundness error of $2^{-\lambda}$, $\pi_{\mathsf{com}}$ must be large concretely (for Filecoin $\approx 290\lambda$). Solution are using SNARKs (Filecoin also supports this) or simply allowing a higher soundness error $\lambda'$ just for the commitment, but that still weakens all other guarantees. Another solution is splitting $\pi_{\mathsf{com}}$ into smaller chunks $\pi^{(1)}_{\mathsf{com}}, \ldots, \pi^{(k)}_{\mathsf{com}}$ and transmitting them chunk-by-chunk over a period of time. This smoothes out bandwidth spikes. Note that in SDR-PoSp each chunk can be checked individually, so the confidence in $\mathsf{com}$'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, set the number of challenges per ping $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$ to ensure an overwhelming detection probability of $1 - e^{-\lambda}$ per ping. This simplifies our proof since we reliably detect a cheating peer immediately. 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 occasionally fail challenges will be disconnected. Thus, a smaller $\kappa$ suffices in practice (we formalize this below), reducing bandwidth.

Second, the ping interval $t_{\mathsf{ping}}$ is deterministic and sufficiently small. Thus, the adversary cannot use the same space for two different identities by re-initializing it between pings. A better approach is the following: At every point in time, a peer will be pinged with uniform probability $\mathsf{pr}_{\mathsf{ping}}$.[22] Intuitively, since the adversary cannot predict pings, it cannot reliably switch between different identities. So pings may be more infrequent, also reducing bandwidth.

[21]By an averaging argument, 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, $\varepsilon'_{\mathsf{space}} = 2\varepsilon_{\mathsf{space}}$ leads to $\mathsf{pr}'_{\mathsf{det}} = \varepsilon'_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}/2 = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$.

[22]To implement this, sample from the geometric distribution parameterized by $\mathsf{pr}_{\mathsf{ping}}$.

**Theoretical Analysis.** Due to the lower $\kappa$ and fewer pings we cannot catch a cheating peer immediately. However, we can still guarantee to catch cheaters after a time span of sufficient length. As already mentioned above, this is sufficient for DHTs since they run for a long amount of time.

More formally, 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. So, compared to the model we used in § 5.1, we additionally assume that honest nodes' clocks are (roughly) synchronized.

When a node discovers a new potential peer, it starts challenging it probabilistically as described above. It only adds 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 $S_{\mathsf{adv}}$ 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 $S_{\mathsf{adv}}$ be the $\mathcal{A}$'s disk space and define $f = \frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}$.*

*If epochs last $T = \frac{4\lambda}{\mathsf{pr}_{\mathsf{ping}}\cdot\mathsf{pr}_{\mathsf{det}}}$ time, in every epoch, at most $2f$ Sybil nodes are part of the DHT except with* $\mathrm{negl}(\lambda)$ *probability.*

Thm. 3 allows us to reduce $\mathsf{pr}_{\mathsf{ping}}$ and $\mathsf{pr}_{\mathsf{det}}$ at the expense of making epochs longer. Even if both are constant, $T$ stays linear in $\lambda$. This improves on the previously required an overwhelmingly large (in $\lambda$) $\mathsf{pr}_{\mathsf{det}}$. The proof is in App. A. We remark that it is not tight, and we expect better security in practice.

**Practical Estimates.** Let us estimate the bandwidth in practice. We ignore the size of $\pi_{\mathsf{com}}$ since may be amortized over a longer period of time as discussed above. We set $\lambda = 50$, $N = 128\,\text{GiB}$, use Filecoin's PoSp with a size of $16\,\text{GiB} = 2^{29}\cdot 256\,\text{bits}$, $\kappa = 18$ parallel challenges to boost $\mathsf{pr}_{\mathsf{det}} \approx 0.3$ (as described above), and $\mathsf{pr}_{\mathsf{ping}} = 1/60$ (i.e., 1 ping per minute in expectation). This choice of parameters leads to an epoch length of $T \approx 11\,\text{h}$, so about half a day.

A challenge requires $\log(128/16) + 29 = 32\,\text{bits}$ (the former select the sub-PoSp, the latter the challenge within this sub-PoSp). A proof is a Merkle tree opening that requires $\approx 29\cdot 256\,\text{bits}$. For $\kappa = 18$ this amounts to a total communication of $18\cdot(32+29\cdot 256)\,\text{bits} \approx 16\,\text{KiB}$ per challenge.

A single node sends one challenge per peer per minute (in expectation). This amounts to 2.2 kbps on average per peer. Generously estimating 1000 concurrent peers, this totals 2.2 Mbps, which even a residential connection may sustain.

## 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 $\mathsf{Virt}$. 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 DHT constructions that provably tolerate a certain fraction of Sybils, yet are still practical. 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.

[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.

[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.

[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.

[chi24] The chia network blockchain, 2024.

[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.

[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.

[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.

[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.

[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 Proofs

*Proof of Thm. 1.* Consider any honest node and any Sybil peer $\mathsf{id}$ at time $t$. The honest node has sent the last challenge to $\mathsf{id}$ in the interval $[t - t_{\mathsf{ping}}, t]$. The second-to-last challenge to $\mathsf{id}$ was sent at time $t_c \in [t - 2t_{\mathsf{ping}}, t - t_{\mathsf{ping}}]$. Note that $t_c$ is well-defined since $\mathsf{Basic.addPeer}$ challenges $\mathsf{id}$ once before adding it.

Since $\mathsf{id}$ is connected at time $t$ by assumption, it passed all past challenges, including the one at time $t_c$. There are three ways how they can pass:

1. $\mathcal{A}$ dedicated $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits of its space to the PoSp with seed $\mathsf{id}$ at some time $t_s \in [t_c, t_c + t_{\mathsf{timeout}}]$.

2. The commitment $\mathsf{com}$ for $\mathsf{id}$ is incorrect.

3. The challenge failed to detect the cheating.

Reasons 2 and 3 occur with $\text{negl}(\lambda)$ probability by the choice of $t_{\mathsf{timeout}}$ and $\kappa$, and the Soundness of $\mathsf{PoSp}$. Replacing $t_c$ gives us $t_s \in [t - 2t_{\mathsf{ping}}, t]$ since $t_{\mathsf{timeout}} \leq t_{\mathsf{ping}}$.

Recall that $\mathcal{A}$ has $f^{(t)} \in \text{poly}(\lambda)$ Sybils in total, each connected to an honest node. So all of them passed a challenge within $[t - 2t_{\mathsf{ping}}, t]$. By a union bound, the probability that *any* Sybil passed due to Reasons 2 and 3 is $\text{negl}(\lambda)$. Thus, due to Nonreusability of $\mathsf{PoSp}$, $\mathcal{A}$ must have dedicated more than $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits to every Sybil node at some point in $[t - 2t_{\mathsf{ping}}, t]$. 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 by a pigeonhole argument. □

*Proof of Thm. 2.* Thm. 1 upper-bounds $f^{(t)}$. We now lower bound $n^{(t)}$.

Let the space of a physical, honest node $i$ at time $t$ be $S_{\mathsf{hon}}^{(t,i)}$ with $S_{\mathsf{hon}}^{(t)} = \sum_i S_{\mathsf{hon}}^{(t,i)}$. At every time $t$, the number of *potentially-available* honest nodes is lower bounded by $\sum_i \left\lceil \frac{\delta \cdot S_{\mathsf{hon}}^{(i,t)}}{N} \right\rceil \geq \sum_i \frac{\delta \cdot S_{\mathsf{hon}}^{(t,i)}}{N} = \frac{\delta \cdot S_{\mathsf{hon}}^{(t)}}{N}$. Note that $\mathcal{A}$ cannot prevent honest nodes from responding to pings since $\Delta < t_{\mathsf{timeout}}/2$.

The above is *not* a lower bound on $n^{(t)}$ because newly-joined nodes are not immediately added as peers, but only after $t_{\mathsf{ping}}$ time (Fig. 2, Line 4). So, the actual number of honest nodes at time $t$ is lower bounded by

$$n^{(t)} - f^{(t)} \geq \min_{i \in [t - t_{\mathsf{ping}}, t]} \frac{\delta \cdot S_{\mathsf{hon}}^{(i)}}{N}. \tag{1}$$

Combining Thm. 1 and Eq. (1) and rearranging yields the desired inequality. □

*Proof of Thm. 3.* In every epoch, $\mathcal{A}$ starts with $\leq \mathrm{poly}(\lambda)$ Sybils, and each may be connected to $\leq \mathrm{poly}(\lambda)$ honest nodes. Define $W$ as the set of *winning* Sybils, i.e., those connected to at least one honest node at the end of an 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 $\mathrm{negl}(\lambda)$ probability, which implies the theorem.

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

Set $W_2 \subseteq W \setminus W_1$, i.e., the set of winning Sybils with *at most* $(1 - \varepsilon_{\mathsf{space}}) \cdot N$ bits for more than $T/2$ time. We will prove that $\Pr[W_2 = 0] \geq 1 - \mathrm{negl}(\lambda)$ by introducing two bad events $B_1, B_2$, each occurring with $\mathrm{negl}(\lambda)$ probability, and showing $\Pr[W_2 = 0] \leq 1 - (\Pr[B_1] + \Pr[B_2 \wedge \neg B_1])$.

$B_1$ occurs when at least one PoSp commitment is not "mostly correct" (cf. Def. 2). By the PoSp's Soundness of Initialization and a union bound over all Sybils in $W_2$, $\Pr[B_1] \leq \mathrm{poly}(\lambda) \cdot \mathrm{negl}(\lambda)$.

$B_2$ occurs when any honest node is still connected to any Sybil in $W_2$ at the end of the epoch. Sybils in $W_2$ have $\leq (1 - \varepsilon_{\mathsf{space}}) \cdot N$ for $\geq T/2$ time. 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 challenge with probability at least $\mathsf{pr}_{\mathsf{det}}$. Since the Sybil is connected to at least one honest peer, it will be challenged with probability $\mathsf{pr}_{\mathsf{ping}}$ at every point in time. Define $B_2'$ as the event that a specific honest node is still connected to a specific Sybil in $W_2$. From the above, we get that $\Pr[B_2' \wedge \neg B_1] \leq (1 - \mathsf{pr}_{\mathsf{ping}} \cdot \mathsf{pr}_{\mathsf{det}})^{T/2} \leq 2^{-2\lambda}$ where the last inequality holds due to $T = \frac{4\lambda}{\mathsf{pr}_{\mathsf{ping}} \cdot \mathsf{pr}_{\mathsf{det}}}$. Union-bounding over the $\mathrm{poly}(\lambda)$ Sybils and $\mathrm{poly}(\lambda)$ number of connections each, we get that $\Pr[B_2 \wedge \neg B_1] \leq \mathrm{poly}(\lambda) \cdot \mathrm{poly}(\lambda) \cdot \Pr[B_2' \wedge \neg B_1] \leq 2^{-\lambda}$.

Putting the above together, we conclude that

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

which also completes the proof of the theorem. □