# 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

November 6, 2024

**Abstract**

Distributed Hash Tables (DHTs) are peer-to-peer protocols that serve as a building block for more advanced applications. Examples include storage networks such as IPFS, Autonomi, Hypercore and Swarm, data availability sampling, or Ethereum's peer discovery protocol.

Like any distributed protocol, when used without a central authority, DHTs are susceptible to Sybil attacks. However, none of the above applications use a Sybil-resistant DHT. While prior work proposes many Sybil-resistant constructions, they are either impractical or unsuitable for the above applications. The most promising approach is proof of work (PoW) where nodes periodically challenges their peers to solve computationally-expensive challenges. In theory, this limits the number of adversarial Sybil nodes, but in practice PoW often does not really work well. Since the above applications do not require honest nodes to have a lot of computational resources, PoW challenges cannot be too difficult. So an adversary using specialized hardware (e.g., FPGAs or ASICs) can easily solve PoW challenges and create a lot of Sybil nodes as a consequence.

In this work, we propose using Proof of Space (PoSp) instead of PoW challenges. While a PoW proves that a node is wasting computation, a PoSp proves that a node is wasting disk space. This is a better Sybil-resistance mechanism because disk space is also the resource required by many of the applications using DHTs. Indeed, most of the aforementioned ones are related to storage. So nodes supply a lot of disk space for the application to function correctly. If they now use a fraction of this space for PoSp, the underlying DHT's Sybil-resistance increases drastically. In particular, to control a constant fraction of all DHT nodes, the adversary's disk space must scale proportionally to the disk space of all honest nodes combined.

## 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. A well-designed structure makes an efficient DHT. 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 often used without any central authority. So they need to withstand *Sybil* attacks [Dou02]. In such an attack, the adversary acts as multiple, different nodes. 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, honest nodes can be *eclipsed* (i.e., cut off) from other honest nodes.

Since DHTs in the wild are usually based on Kademlia [MM02], they are somewhat resistant to Sybil attacks. For example, Kademlia nodes prefer peers with long uptimes. Some implementations also use ad-hoc approaches (e.g., to address Aspect 2, a node's identifier is the hash of its IP address). Still, prior literature performs attacks with modest resources, e.g., eclipse [PMZ22] or content-censorship attacks [SAK+24] on IPFS [Ben14].

## 1.2 Existing Countermeasures are Insufficient

The literature proposes many defenses against Sybil attacks on DHTs (and distributed networks in general) with more or less provable guarantees. For example, using social relationships between nodes (e.g., [DLLKA05, LLK10]), or ensuring redundancy by grouping nodes together (e.g., [AS06, JPS+18]). However, these approaches are not applicable to or efficient enough for the blockchain setting.

The most promising approach is *proof of work* (PoW). In principle, it 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].

Yet, PoW is no panacea. First, in the above applications (e.g., IPFS [Ben14]), honest nodes participate with general-purpose hardware. Thus, PoW challenges cannot be too difficult as honest nodes cannot solve them otherwise. As a consequence, they offer little protection against an adversary running dedicated hardware (e.g., FPGAs or ASICs). This is especially problematic when using PoW to limit the number of Sybils (Aspect 1). Using PoW to harden identifier generation (Aspect 2) still works reasonably well. Since identifiers are generated only once, the PoW difficulty may be set to a reasonable level. Second, repeated PoW challenges (Aspect 1) waste a lot of energy. This is incompatible with many blockchains shifting to more environmentally-friendly protocols (e.g., proof of stake).

In summary, while PoW may be used to restrict the adversary's choice of identifiers to some degree (Aspect 2), it cannot limit the number of Sybils (Aspect 1) in a meaningful way.

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

We propose using *proof 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, these 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]. The former [Fis19] is better suited for securing DHTs as we discuss in § 6.

**Proofs of Space Synergize with Applications.** The services provided by the above applications mostly revolve around storing data (e.g., storage networks or data availability sampling). So participating nodes usually have a lot of disk space at their disposal.[2] Therefore, in contrast to PoW, nodes can waste a meaningful amount, say 512 GiB, of space using a PoSp. If this PoSp is checked periodically, nodes must waste space, making attacks expensive.

We can take this even further by linearly scaling the amount of wasted disk space with nodes' total available space. To attack the DHT, the adversary must now provide disk space in proportion to what all honest nodes combined dedicate to the application, e.g., IPFS [Ben14].

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

[2] Depending on the application, storage nodes could be different from the nodes participating in the DHT. In practice, however, they usually do (e.g., in IPFS [Ben14] by default). Thus, we assume that DHT nodes have meaningful amounts of disk space available.

**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 also applies to other peer-to-peer protocols, but we focus on the DHT use-case. In a nutshell, the basic protocol ensures 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.

An extension of the basic protocol 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 the basic protocol. 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., for storing files in IPFS [Ben14]. Essentially, this yields guarantees often needed by Byzantine-fault-tolerant systems (e.g., Byzantine agreement).

**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* $n_{\mathsf{adv}}$ *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* $\alpha$ *as well as system parameters.*

Both of our approaches limit the number of Sybil nodes (Aspect 1). However, they cannot be used to rate-limit identifier generation (Aspect 2) for subtle reasons. We defer the discussion of Aspect 2 to § 4.1.

### 1.4 Outline

We further motivate the need for our solution by describing related works, especially Sybil-defenses, in § 2. Then, we cover preliminaries (DHTs and PoSp) in § 3 before stating our constructions in § 4. § 5 covers their theoretical guarantees and shows how they might be used in combination with prior works. We also discuss practical matters in § 6, most importantly, which existing PoSp construction to use and why. We conclude and discuss possible future work in § 7.

## 2 Related Work

**Distributed Hash Tables.** Chord [SMK+01] is a simple and efficient construction. Kademlia [MM02] is the most popular in practice. It has inspired follow-up works, e.g., [ZBV24, BM07]. Constructions with better (asymptotical) efficiency exist [KK03, GV04], but they are not used in practice.

**Proofs of Space.** PoSp were initially introduced by Dziembowski et al. [DFKP15]. Their construction is based on graph labeling. Works building on this idea include *proofs of catalytic space* [Pie19], which allow storing useful data instead of wasting

space, and the *stacked expander graphs* construction [RD16]. The latter inspired *stacked depth-robust graphs* [Fis19] which serve as the basis for Filecoin's [Pro17] PoSp and its *proof of replication* [Fis19]. 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.

**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, the identifier of a node is a hash of its IP address [DKK+01]. Some also apply to very specific contexts. In the Ethereum context, one solution against Sybil attacks are *proofs of validator* [KMNC23]. Validators stake large amounts of Ethereum, so they are economically incentivized to behave correctly.

**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

[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.

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 $\lambda$ 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 common notation 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 $\mathsf{key}$.

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. A well-designed structure enables efficient lookups. Important metrics include the number of peers and the number of hops between nodes per $\mathsf{lookup}$ query.

Our idea of using PoSp for Sybil resistance are quite general. We only make mild assumptions on how nodes manage their peers. We abstract this by the following functions below. For example, our construction may be used with Kademlia [MM02]. While understanding its inner workings is not necessary to follow the paper, it is instructive (cf. App. A).

**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 network and 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.
- $\mathsf{lookup}(\mathsf{key}) \to (\mathsf{id}, \mathsf{aux})$: On input of a key $\mathsf{key}$, returns the node responsible for $\mathsf{key}$.

## 3.2 Proof of Space

A proof of space (PoSp) as defined by Dziembowski et al. [DFKP15] is, informally speaking, the disk space analogue to a PoW. It 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. Both 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, on input $\mathsf{seed}$, the prover generates an output file $\mathsf{file}$ 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 $\mathsf{com}$ to $\mathsf{file}$ and data produced in course of its derivation from $\mathsf{seed}$. It sends the commitment $\mathsf{com}$ to the verifier who then checks that $\mathsf{com}$ is mostly correct—what "mostly correct" precisely means depends on the protocol. We assume that this check is non-interactive.[4]

In the online protocol, the verifier challenges the prover to demonstrate that they are still storing $\mathsf{file}$ in its entirety. They do so by sending a uniformly random challenge[5] $\mathsf{chal}$ to the prover. The prover responds with a proof $\pi_{\mathsf{chal}}$ which the verifier verifies with the help of $\mathsf{com}$ and $\mathsf{seed}$. By periodically executing the online protocol, the verifier ensures that the prover is storing $\mathsf{file}$ for some span of time.

One essential requirement is efficiency, otherwise constructing PoSp is trivial.[6] While generating $\mathsf{file}$ takes $O(N)$ at best, all other computations, especially the ones of the verifier, must be fast, i.e., polylog$(N)$ time. Similarly, $\mathsf{com}$, $\pi_{\mathsf{chal}}$, 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 $(1-\varepsilon_{\mathsf{space}})\cdot N$ 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 $\mathsf{pr}_{\mathsf{det}}$.

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

- $\mathsf{Init}(\mathsf{seed}) \to (\mathsf{file}, \mathsf{com}[7], \pi_{\mathsf{com}})$
- $\mathsf{VerifyInit}(\mathsf{seed}, \mathsf{com}, \pi_{\mathsf{com}}) \to b$ 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 $\pi_{\mathsf{chal}} \leftarrow \mathsf{Prove}(\mathsf{seed}, \mathsf{file}, \mathsf{com}, \mathsf{chal})$.

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

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

[6] 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 huge.

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

**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 $\mathrm{poly}(\lambda, \log N)$.

**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 probability negligible in $\lambda$.

**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 $\mathsf{Verify}(x, \widetilde{\mathsf{com}}, \mathsf{chal}, \tilde{\pi}_{\mathsf{chal}}) \to \mathsf{true}$.

# 4 Constructions

## 4.1 Basic Construction

The Basic DHT construction requires every node to continually store a PoSp $\mathsf{file}$ of a fixed size, e.g., $N = |\mathsf{file}| = 128$ GiB. To ensure this, every node periodically challenges its peers to prove that they are still storing $\mathsf{file}$. Intuitively, this ensures that the number of Sybil nodes is bounded by $n_{\mathsf{adv}} < S_{\mathsf{adv}}/((1 - \varepsilon_{\mathsf{space}}) \cdot N)$ where $\varepsilon_{\mathsf{space}}$ is the space gap. 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 $\mathsf{join}$, $\mathsf{addPeer}$, and $\mathsf{pingPeer}$ as defined in Def. 1. It introduces the following global system parameters:

- $N$ is the size of the PoSp each node stores.
- $t_{\mathsf{ping}}$ 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 $\mathsf{pingPeer}$ waits for a response from the peer.
- $\kappa$ the number of PoSp challenges per $\mathsf{pingPeer}$.

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

A node joins the network using Basic.join[8] (Fig. 1). As part of this, it generates an identifier $\mathsf{id}$ according to DHT and then initializes a PoSp with $\mathsf{id}$ as input. It stores the resulting file $\mathsf{file}$ on disk and returns the $\mathsf{id}$ and auxiliary data.

Other nodes add a node as a peer using Basic.addPeer (Fig. 2) The function checks that $\mathsf{com}$ is a valid commitment to a PoSp with input $\mathsf{id}$.

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

Figure 1: Pseudocode of Basic.join.

**Basic.join() $\to$ (id, aux):**

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

Figure 2: Pseudocode of Basic.addPeer.

**Basic.addPeer(id, aux):**

01 Extract com and $\pi_{\mathsf{com}}$ from aux.
02 If PoSp.VerifyInit(id, com, $\pi_{\mathsf{com}}$) $\to$ false, abort.
03 DHT.addPeer(id, aux)

After having added a peer, Basic.pingPeer (Fig. 3) is run periodically with an interval of $t_{\mathsf{ping}}$ time. pingPeer checks that the peer is online and is still storing file associated with com. To this end, it sends $\kappa$ 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 $t_{\mathsf{timeout}}$, it is deemed offline. Else, all proofs $\pi_{\mathsf{chal}_i}$ are verified.

Figure 3: Pseudocode of Basic.pingPeer.

**Basic.pingPeer(id, aux):**

01 Extract and com from aux.
02 Send $\kappa$ uniformly random challenges $\mathsf{chal}_1, \ldots, \mathsf{chal}_\kappa$ to id.
03 Wait $t_{\mathsf{timeout}}$ 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, $\mathsf{chal}_i, \pi_{\mathsf{chal}_i}$)

Note that Basic does not influence the generation of id. It simply takes whatever DHT.Init outputs (Line 1 in Fig. 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 makes them more difficult.

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

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

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 $\mathsf{com}$, the other using $\mathsf{com}'$). This would render all guarantees of $\mathsf{Basic}$ moot.

## 4.2 Virtual Nodes

While $\mathsf{Basic}$ attaches a disk space requirement to nodes, the resulting guarantees do not take the honest disk space into account. The reason is that $\mathsf{Basic}$ does not care whether a node has, e.g., $2N$ or $100N$ bits of disk space available. The next construction, $\mathsf{Virt}$, takes this into account. It ensures that the fraction of adversarial nodes is bounded by a function of the adversarial *and* honest disk space. Looking ahead, Thm. 2 shows that $n_{\mathsf{adv}}/n < S_{\mathsf{adv}}/(S_{\mathsf{adv}} + \beta \cdot S_{\mathsf{hon}})$ for some constant $0 < \beta < 1$ depending on system parameters.

The core idea of $\mathsf{Virt}$ is to use *virtual nodes* [DKK+01].[9] This means that one physical node acts as one or more virtual nodes. Each virtual node runs one $\mathsf{Basic}$ instance. Thus, the number of virtual nodes of an honest node is related to its available disk space.

Since $\mathsf{Virt}$ uses $\mathsf{Basic}$ as a building block, it inherits its system parameters of $N$, $t_{\mathsf{ping}}$, $t_{\mathsf{timeout}}$, and $\kappa$. In addition, $\mathsf{Virt}$ 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 $i$ wants to participate in the DHT. We denote its total disk space by $S^{(i)}_{\mathsf{hon}}$. It participates as

$$\left\lceil \frac{\delta \cdot S^{(i)}_{\mathsf{hon}}}{N} \right\rceil \tag{1}$$

virtual nodes in the network. Each of these virtual nodes runs one $\mathsf{Basic}$ protocol instance as described in § 4.1.

Note that rounding up in Eq. (1) implies that every physical node must have at least $N$ bits of space. Otherwise, it does not have the resources to run even one $\mathsf{Basic}$ instance. In practice, it should have more space because it also needs to store application data.

[9] 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].

Figure 4: Notation summary.

| | | | |
|---|---|---|---|
| $\lambda$ | Security parameter | $S_{\mathsf{hon}}$ | Total honest space |
| $n$ | Number of nodes | $S_{\mathsf{adv}}$ | Adversarial space |
| $n_{\mathsf{hon}}$ | Number of honest nodes | $N$ | Prescribed PoSp size |
| $n_{\mathsf{adv}}$ | Number of Sybil nodes | $\delta$ | 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 |

# 5 Theoretical Perspective

In this section, we will analyze the theoretical guarantees of Basic and Virt. To this end, we introduce an idealized system model. Afterward, we give an application of these theoretical guarantees.

So far, we have introduced a lot of parameters. All of them influence the guarantees, so we briefly summarize them in Fig. 4.

We emphasize that $\varepsilon_{\mathsf{space}}$, $\varepsilon_{\mathsf{time}}$, and $\mathsf{pr}_{\mathsf{det}}$ quantify the guarantees of the PoSp. Recall: A cheating prover storing at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits *and* taking less than $(1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$[10] time to answer a challenge will be detected with probability $\mathsf{pr}_{\mathsf{det}}$.

## 5.1 Idealized System Model

We use an idealized system model to bound the number of Sybil nodes $n_{\mathsf{adv}}$ in Basic and Virt. This allows us to make no assumptions about the underlying DHT, simplifying the analysis. In the following, we will start with the real system and simplify it step-by-step. In the end, we arrive at the ideal system model. If the system parameters $t_{\mathsf{ping}}$, $t_{\mathsf{timeout}}$, and $\kappa$ are set appropriately, each step is justified.

The real system runs for an indefinite amount of time; nodes repeatedly ping their peers within this time. The first simplification is the following: We assume that all nodes ping their peers at the same time and restrict our attention to a single ping. In other words, we analyze the state of the system at a specific point in time. This is a reasonable approximation as long as $t_{\mathsf{ping}} < (1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$.[11] In particular, this ensures that the adversary cannot use *one* space (i.e., $N$ bits) for *two* different Sybil identities by re-initializing the PoSp between pings of different nodes.

Next, we assume that an adversarial node fails a challenge with probability $\mathsf{pr}_{\mathsf{det}}$ if it stores at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits. This is reasonable as long as $t_{\mathsf{timeout}} < (1-\varepsilon_{\mathsf{time}})\cdot\mathsf{time}(\mathsf{Init})$. Then, by the guarantees of the PoSp, the cheating node does not have enough time to produce a proof that verifies.

Finally, we assume that VerifyInit and Verify are perfectly sound, i.e., if an adversarial node stores at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits, Verify will return false. For VerifyInit this is reasonable since its soundness error is negligible in the security parameter $\lambda$ by Def. 2.

[10] Note that time(Init) in wall-clock time is not know as it depends on the adversary's hardware. So it must be estimated, ideally with a sufficiently large margin of error accounted for.

[11] Note that we are viewing this from a theoretical perspective. In practice, such frequent pings might not be feasible; we defer discussion to § 6.2.

For Verify, a similar argument holds as long as $\kappa$ is appropriately large. A cheating peer storing at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits evades detection with probability at most $(1-\mathsf{pr}_{\mathsf{det}})^{\kappa}$. So, setting $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$ suffices since $(1-\mathsf{pr}_{\mathsf{det}})^{\lambda/\mathsf{pr}_{\mathsf{det}}} \leq e^{-\lambda}$ is negligible in $\lambda$.

Given the above simplifications, pings are now independent of the time and their outcome is deterministic: An adversarial node fails all pings if it stores at most $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits.

## 5.2 Security Analysis

In the idealized system model, analysis of Basic and Virt is simple.

**Theorem 1** (Basic)**.** *In the idealized system model (§ 5.1),* Basic *bounds the number of adversarial nodes by*

$$n_{\mathsf{adv}} < \frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}.$$

*Proof.* In the ideal system model, observe that $n_{\mathsf{adv}}$ equals the number of adversarial nodes storing more than $(1-\varepsilon_{\mathsf{space}})\cdot N$ bits. Given a total space of $S_{\mathsf{adv}}$, the maximum number of Sybil nodes is bounded by $\frac{S_{\mathsf{adv}}}{(1-\varepsilon_{\mathsf{space}})\cdot N}$. □

To analyze Virt, we need to consider the space of each honest *physical* node. Recall that physical node $i$'s space is denoted by $S^{(i)}_{\mathsf{hon}}$ and $S_{\mathsf{hon}} = \sum_i S^{(i)}_{\mathsf{hon}}$ in total. Since each physical node $i$ dedicates $\delta \cdot S^{(i)}_{\mathsf{hon}}$ bits to Virt, $n_{\mathsf{hon}} = \sum_i \left\lceil \frac{\delta \cdot S^{(i)}_{\mathsf{hon}}}{N} \right\rceil$ by Eq. (1).

**Theorem 2** (Virt)**.** *In the idealized system model (§ 5.1),* Virt *guarantees that*

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

*Proof.* Observe that $n_{\mathsf{hon}} = \sum_i \left\lceil \frac{\delta \cdot S^{(i)}_{\mathsf{hon}}}{N} \right\rceil \geq \sum_i \frac{\delta \cdot S^{(i)}_{\mathsf{hon}}}{N} = \frac{\delta \cdot S_{\mathsf{hon}}}{N}$. In the ideal system model, $n_{\mathsf{adv}} < S_{\mathsf{adv}}/((1-\varepsilon_{\mathsf{space}})\cdot N)$ by Thm. 1. Since $n = n_{\mathsf{adv}} + n_{\mathsf{hon}}$, plugging both into $n_{\mathsf{adv}}/(n_{\mathsf{adv}} + n_{\mathsf{hon}})$ and rearranging yields the desired inequality. □

## 5.3 Application: Redundant Constructions

As described in § 2, prior works use redundancy to make DHTs more resilient. These works require some bound on the fraction of Sybils in the network. Virt gives such guarantees provided the honest parties combined have sufficiently more space than the adversary. The following corollary of Thm. 2 formalizes this.

**Corollary 1** (of Thm. 2)**.** *In the idealized system model (§ 5.1),* Virt *guarantees that*

$$\frac{n_{\mathsf{adv}}}{n} < \alpha$$

*for any constant* $0 \leq \alpha < 1$ *if*

$$S_{\mathsf{adv}} < \frac{\alpha}{1-\alpha}\cdot(1-\varepsilon)\cdot\delta\cdot S_{\mathsf{hon}}.$$

For example, consider the group-based construction due to Jaiyeola et al. [JPS+18]. They assigning nodes to groups. 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. This is secure under some assumptions. The most important one is that at most $n_{\mathsf{adv}} < \alpha \cdot n$ nodes are Sybils (for a certain $\alpha$). This is precisely what Virt guarantees by Cor. 1.

Similarly, Cor. 1 is useful for constructions such as Halo [KT08]. It is a system that performs lookups in a redundant manner to achieve a form of Sybil-resistance. To provide guarantees, it also requires $n_{\mathsf{adv}} < \alpha \cdot n$.

# 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 used in practice. Both follow very different approaches. Filecoin's [Pro17] PoSp is *stacked depth-robust graphs* (SDR—PoSp) [Fis19], while Chia's [chi] is based on *function inversion* (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. It does not rule out that parallelism may speed up the computation. 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. Ideally, the space gap $\varepsilon_{\mathsf{space}}$ and time gap $\varepsilon_{\mathsf{time}}$ should be small. Reyzin [Rey23] observes the following: FI-PoSp's space gap $\varepsilon_{\mathsf{space}}$ grows as the PoSp size $N$ increases—this is bad. For SDR-PoSp, there are two analyses: Fisch [Fis19] achieves arbitrary $\varepsilon_{\mathsf{space}}$, but $\varepsilon_{\mathsf{time}}$ increases as $\varepsilon_{\mathsf{space}}$ decreases. Reyzin [Rey23] improves on this with arbitrary $\varepsilon_{\mathsf{space}}$ and $\varepsilon_{\mathsf{time}}$. For concrete implementation deployed by Filecoin, Reyzin proves that $\varepsilon_{\mathsf{space}} = 0.2$, $\varepsilon_{\mathsf{time}} < 0.8$, and $\mathsf{pr}_{\mathsf{det}} = 0.1$. Unfortunately for us, Reyzin's analysis only counts sequential time. It is an open problem to come up with a construction or analysis achieving comparable parameters against parallel time adversaries.

**Combining Proofs of Space.** Due to hardware constraints during Init, the PoSp size $N$ is limited in practice, e.g., for SDR-PoSp feasible values are $N = 16$ or 32 GiB as implemented by Filecoin. Combining $k$ sub-PoSp of size $N$ results in a large PoSp of size $k \cdot N$. Naively, this requires $k$ challenges in parallel to keep $\varepsilon_{\mathsf{space}}$ and $\mathsf{pr}_{\mathsf{det}}$ the same. This leads to a $\times k$ bandwidth increase.

To reduce bandwidth, suppose we challenge the PoSp as follows: Randomly pick one sub-PoSp $i \leftarrow\!\!\$\ [k]$ and send one challenge to the $i$th sub-PoSp. 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 $\mathsf{pr}'_{\mathsf{det}} = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$.[12] Note that both do not depend on $k$. For Filecoin's parameters [Rey23] stated above, $\varepsilon'_{\mathsf{space}} = 0.4$ and $\mathsf{pr}_{\mathsf{det}} = 0.02$ for a single challenge.

### 6.2 Reducing and Shaping Bandwidth

Bandwidth is a limited resource, so we discuss three ways to optimize it. First, in the ideal system model (§ 5.1) we assume that $\kappa$ (i.e., the number of challenges per ping) is sufficiently large. Recall we set $\kappa = \lambda/\mathsf{pr}_{\mathsf{det}}$ to ensure an overwhelming detection probability $\mathsf{pr}_{\mathsf{det}} = 1 - e^{\lambda}$. This was only necessary because the ideal model considers only one point in time. In practice, however, DHTs run for a long amount of time, and also use heuristics to manage peers. For example, Kademlia [MM02] prefer 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 seems sufficient in practice, reducing bandwidth.

Second, the ideal model assumes frequent pings with a deterministic, sufficiently small interval $t_{\mathsf{ping}}$. A more practical (but harder to analyze theoretically) approach is sampling $t_{\mathsf{ping}}$ probabilistic after every ping. This allows for more infrequent pings. Concretely, let $T$ be an upper bound on $t_{\mathsf{ping}}$. After having pinged a peer, a node samples $t_{\mathsf{ping}} \leftarrow\!\!\$\ [T]$. This ensures that the peer will be pinged with uniform probability $1/T$ at every point in time.

Third, initially transmitting $\pi_{\mathsf{com}}$ also incurs a large bandwidth spike when using SDR—PoSp. It achieves an exponentially small soundness error of $2^{-\lambda}$, but $\pi_{\mathsf{com}}$ is quite large. One solution is allowing a higher soundness error, but that is not ideal. Another 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 longer period of time. This smoothes out bandwidth spikes. The SDR-PoSp enjoys the property that each chunk reduces the soundness error by a factor $2^{-\lambda/k}$ [Rey23]. So the confidence in $\mathsf{com}$'s correctness increases gradually.

## 7 Conclusion and Future Work

We have laid the groundwork for using PoSp as a Sybil-resistance mechanism. Our constructions are simple and come with provable, theoretical guarantees in an idealized, yet reasonable system model. Practical deployments seem feasible; we have given recommendations for a performant deployment.

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.

[12] 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 $\mathsf{pr}'_{\mathsf{det}} = 0.5 \cdot \varepsilon'_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}} = \varepsilon_{\mathsf{space}} \cdot \mathsf{pr}_{\mathsf{det}}$.

Second, finding alternatives to virtual nodes since this approach to 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? Third, reducing the amount of space wasted by honest nodes. A possible solution might be proofs of catalytic space [Pie19]. They allow nodes to store useful data with the caveats that 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. Can the ideas of [Pie19] lead to more useful features?

**Acknowledgements.** This research was funded in whole or in part by the Austrian Science Fund (FWF) 10.55776/F85.

# 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.

[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.

[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

Kademlia [MM02] is a practically efficient DHT. A node is identified by a randomly chosen $\mathsf{id} \in \{0,1\}^\ell$ (e.g., $\ell = 160$); the key space is also $\{0,1\}^\ell$. Kademlia's structure defines the distance between two nodes as $\mathsf{dist}(\mathsf{id}_1, \mathsf{id}_2) = \mathsf{id}_1 \oplus \mathsf{id}_2$ which is a symmetric metric.

Every node stores its peers in a routing table. This table consists of $\ell$ buckets where each bucket may store up to $k$ peers. The peers in the $i$th bucket have a distance in between $(2^i, 2^{i+1}-1)$ to the node itself. The parameter $k$ controls the amount of replication and thereby affects the resilience of the system; $k = 20$ is common.

A node either actively searches for potential peers by performing lookup queries or passively discovers them by answering lookup queries of other nodes (here, the symmetry of the metric is important). A node adds a newly-discovered peer into its corresponding bucket, provided the bucket is empty. Buckets covering larger distances are usually full, so a node needs to decide which peers to keep in the bucket and which ones to evict. The original paper [MM02] describes a least-recently seen eviction strategy which never evicts online nodes. This ensures that nodes with longer uptime are kept in the routing table.

lookup(key) returns the $k$ closest nodes to key. First, a node finds the $\alpha$ closest peers in its routing table. Then, it asks them for their $k$ peers closest to key. From these responses, the node again picks the $\alpha$ closest and iterates until it has found the $k$ closest nodes to key. Intuitively, $\alpha$ controls how many lookups are performed concurrently which increases

resilience; $\alpha = 3$ is usual. By the structure of the routing table, lookups are possible in time $O(\log n)$.

A new node joins the network by connecting to an existing node which acts as the *bootstrap* node. Initially, the new node's routing table only contains the bootstrap node. Then, the new node performs $\mathsf{lookup}(\mathsf{id})$ where $\mathsf{id}$ is its desired identifier. This allows the node to find its place in the network.

Some implementations assume (or even require) that new participants look find a bootstrap nodes themselves (e.g., bootstrapping from a friend who is already part of the network). Commonly, however, there is a public list of bootstrap nodes run by reasonably trusted parties in the network. For example, the public bootstrap nodes of IPFS are run by Protocol Labs [PMZ22].