Content Censorship in the InterPlanetary File System

Srivatsan Sridhar*, Onur Ascigil†, Navin Keizer¶, François Genon‡
Sébastien Pierre‡, Yiannis Psaras‖, Etienne Rivière‡, Michał Król§

*Stanford University †Lancaster University ¶University College London
‡ICTEAM, UCLouvain ‖Protocol Labs §City, University of London

***Abstract*—The InterPlanetary File System (IPFS) is currently the largest decentralized storage solution in operation, with thousands of active participants and millions of daily content transfers. IPFS is used as remote data storage for numerous blockchain-based smart contracts, Non-Fungible Tokens (NFT), and decentralized applications.**

We present a content censorship attack that can be executed with minimal effort and cost, and that prevents the retrieval of any chosen content in the IPFS network. The attack exploits a conceptual issue in a core component of IPFS, the Kademlia Distributed Hash Table (DHT), which is used to resolve content IDs to peer addresses. We provide efficient detection and mitigation mechanisms for this vulnerability. Our mechanisms achieve a 99.6% detection rate and mitigate 100% of the detected attacks with minimal signaling and computational overhead. We followed responsible disclosure procedures, and our countermeasures are scheduled for deployment in the future versions of IPFS.

I. Introduction

Inter-Planetary FileSystem (IPFS) is the largest decentralized peer-to-peer filesystem currently in operation. The platform underpins various decentralized web applications [13], including social networking and discussion (Discussify [54], Matters News [16]), data storage (Space [19], Peergos [18], Temporal [21]), content search (Almonit [1], Deece [6]), messaging (Berty [3]), content streaming (Audius [2], Watchit [73], DTube [7]), gaming (Gala [23], Splinterlands [20]), and e-commerce (Ethlance [10], dClimate [5]). IPFS is widely used as external storage for blockchain-based applications, including valuable NFT platforms. Support for accessing IPFS has further been integrated into HTTP gateways (*e.g.*, Cloudflare) and mainstream browsers such as Opera and Brave, allowing easy uptake. The IPFS network currently contains a steady number of 25,000 online nodes, spread across 2,700 Autonomous Systems and 152 countries, according to a recent study [66] that also observed widespread usage by clients with 7.1 million content retrieval operations observed from a single vantage point and during a single day.

IPFS is a content-centric network where each piece of content is identified by a Content Identifier (CID), similarly to BitTorrent [4] or Content-Centric Networking [72]. CIDs are derived by hashing the content and do not embed any network location information. Such an approach enables easy content deduplication and the retrieval of data from the closest available location, in addition to maintaining data integrity.

Data retrieval in a content-centric network requires mapping CIDs into network identifiers (*i.e.* IP addresses and port numbers) of nodes hosting the content, called *providers*. Without this resolution mechanism, nodes willing to fetch data, or *downloaders*, have no means to know where to send their requests for data. The design of IPFS results from decades of research on how to build efficient P2P systems [24], [49]. It uses resolution based on a Distributed Hash Table (DHT) combined with Bitswap, a flooding-based, unstructured search mechanism. Similarly to systems such as Gnutella [59], downloaders use Bitswap to establish connections to random peers in the network and send them content queries. Bitswap acts as a lightweight cache and speeds up the retrieval of popular content, but cannot provide discovery guarantees, in particular for newer or less popular data.

Reliable content discovery is provided by the DHT-based resolution system. Nodes hosting content advertise themselves as providers in the network. First, they create *provider records* linking their hosted content (identified by CIDs) to their network location (*i.e.*, IP address and port number). Second, the providers send the provider records to be stored on a fixed number of designated nodes. We refer to those nodes as *resolvers*. Downloaders wishing to fetch the content contact the same resolvers, retrieve the relevant provider records, and then directly contact the discovered providers to download the data. The DHT guarantees to find the content if it is stored in the network. Its proper operation is, therefore, of paramount importance to ensure content availability. IPFS uses the `libp2p` implementation [14] of the Kademlia DHT [52].

Contributions. We make four main contributions.

First, we present a content censorship attack targeting the main IPFS DHT-based resolution system. The attack relies on strategically placing Sybil identities in the network so that they replace honest resolvers for a given CID. As a result, downloaders cannot discover provider records for the target CID and are unable to download the content. The attack can be performed from a single, resource-constrained machine at very little cost ($4 using AWS) and makes the provider records unavailable after a time that ranges from a few seconds to up to 48h depending on the initial setup. Currently, IPFS has no mechanisms to counter the attack, threatening the security of systems using IPFS as a storage platform. This includes collaborative file hosting solutions such as Filecoin [58] and systems building upon it [36], [45]. It also concerns the many proposals combining IPFS for storage with blockchain-hosted application logic, e.g., to implement social

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23153
www.ndss-symposium.org

networks [69], domain-specific data sharing applications [46], [55], or decentralized equivalents to centralized services such as ride-sharing [44].

Second, we present a reliable attack detection technique that analyzes the distribution of peer IDs in the network using the KL Divergence metric [53]. This method extends previous work [30] and leverages a local density-based network size estimator [47], [50], [61] to automatically adjust the detection to the dynamic size of the IPFS network. The detection allows providers to execute mitigation techniques, which may be more costly than the default mode, only when an attack is detected, thereby minimizing the overhead when there is no attack. The detection can be performed by any node during regular content resolution operations and does not incur any additional message overhead. In our experiments on the live IPFS network, our attack detection method was able to detect 99% of the attacks with a false positive rate of 4%, while allowing users to trade off these rates based on individual preferences. A higher detection rate ensures better security while also leading to more false positives that increase the overhead when there is no attack.

Third, we introduce a mitigation technique that allows us to reliably discover provider records regardless of the number of Sybil nodes placed by an attacker around the target CID. The mitigation replaces the regular *put* and *get* DHT operation by hash space region-based queries. Using these, providers always find honest resolvers to store their provider records and querying nodes always discover these honest resolvers and receive true provider records. While introducing an overhead sub-linear in the number of Sybil nodes placed close to the target CID, this mitigation is only enforced when suspicions exist about the existence of an attack, as indicated by our detection mechanism.

Finally, we implement the attack using a custom IPFS DHT server node, and we implement our detection and mitigation techniques on top of the `libp2p` DHT [14].[1] Importantly, the detection and mitigation implementations are fully compatible with the unmodified IPFS clients and can be incrementally deployed in the system. Therefore, while nodes that have not upgraded may remain vulnerable to the censorship attack, they continue to interoperate with nodes that have upgraded. We evaluate the feasibility of the attack and the efficiency of the countermeasures using simulations as well as actual experiments on the live IPFS network. Our proposed detection and mitigation schemes are scheduled to be deployed in the next release of the `libp2p` DHT.

Related Attacks and Mitigations. Similar DHT vulnerabilities have been previously discussed in the literature [31] and multiple prevention mechanisms have been proposed [31]–[34]. However, mostly due to practical reasons [26], [31], [32] or unrealistic assumptions [33], [34], [57], these mechanisms cannot be deployed in modern decentralized systems. We provide a detailed discussion on this topic in Section IX. As a result, multiple top-tier systems currently rely on a vulnerable DHT for various purposes. The peer discovery mechanism for several blockchains (*e.g.*, Ethereum [22], Celestia [29], or Polkadot [27]) uses the same `libp2p` DHT implementation as IPFS. File sharing in I2P [64] and data dissemination in Dat [35] also use the Kademlia DHT, although with a different implementation. Our contributions (both the attack, its detection, and mitigation mechanisms) are expected to apply to these systems as well and more broadly to systems using Kademlia or a Kademlia-like DHT.

Outline. In Section II, we discuss ethical considerations for our study. Section III presents background on IPFS and its content resolution mechanisms. Section V, Section VI, and Section VII respectively introduce the attack, its detection, and mitigation techniques. In Section VIII we evaluate all these mechanisms experimentally and we discuss related work in Section IX. Section X provides a discussion on the implication of the attack and its countermeasures, while Section XI concludes the paper. This paper has an accompanying artifact which contains implementations of the attack, detection and mitigation, and experiments. Appendix A describes how to access the artifact and run the experiments to reproduce the results stated in this paper.

II. Ethical Considerations and Responsible Disclosure

Our work discovers a vulnerability in an existing system with thousands of users, and our experiments involve mounting an attack on the live IPFS network. This may raise ethical concerns. However, we have worked closely with Protocol Labs, the company that created and maintains IPFS and `libp2p`, to develop mitigations for the attack and ensure that our experiments do not cause harm to any existing users.

As soon as we discovered the potential security impact of the vulnerability in June 2021, we initiated a responsible disclosure process by contacting Protocol Labs. Subsequently, the vulnerability has been assigned a CVE record CVE-2023-262481[2]. Details of the CVE record will be made public once the mitigation is deployed and this work is published. To ensure no harm or danger to regular operations or honest users, we limited our censorship attacks to randomly generated content published by our own nodes. Our attacker nodes were implemented to only drop messages related to our randomly generated content and behave normally otherwise to avoid any other side effects on the network.

Throughout the process, Protocol Labs actively supported our research by providing access to their datasets, including network crawls. They welcomed further research based on our findings and provided advice on performance requirements for detection and mitigation techniques. Public disclosure of the vulnerability was carried out at IPFS Camp 2022 [12] held by Protocol Labs, which included discussions with IPFS developers, maintainers, and other researchers. We are currently working with Protocol Labs on the deployment of our detection and mitigation methods in the next version of the `libp2p` DHT. This deployment will benefit other systems using `libp2p`, including Ethereum [22], Celestia [29], or Polkadot [27]. Our mechanisms are not specific to the `libp2p` implementation of the Kademlia protocol and can be ported to other implementations of the DHT by their maintainers. We commit to assisting these maintainers in this task.

[1] See Appendix A for details including where to find the implementations.

[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-26248

III. Background

In this section, we provide the necessary background information to ensure a comprehensive understanding of the attack described in this paper. We start with a description of the Distributed Hash Table (DHT) used by IPFS, followed by its content resolution mechanisms. We also detail techniques for network size estimation, necessary for our attack detection and mitigation mechanisms.

A. IPFS DHT

We review the features of the Kademlia DHT [52] and its `libp2p` implementation [14] that are the most relevant to our attack. To participate in the DHT, each peer generates a public/private key pair and derives an identity peerid $\in \{0,1\}^{256}$ as the hash of its public key. Ideally, each peer generates a random key pair and, therefore, peer IDs are distributed uniformly and independently over the space $\{0,1\}^{256}$. While honest nodes follow this rule, malicious nodes may generate and choose from an arbitrary number of key pairs. Each peer maintains a routing table consisting of $m = 256$ buckets. The i-th bucket contains the addresses of up to $k = 20$ peers whose peer IDs share a common prefix of exactly i bits with the peer's own peer ID.

A new participant node joins the IPFS network by contacting one of the hardcoded bootstrap nodes. This bootstrap node provides the new node with some initial peers allowing it to join the DHT. The new node uses this information to perform a walk through the DHT towards its own peer ID. The walk allows to: *(i)* make sure that there is no other node in the network with the same ID; *(ii)* discover new peers and fill the newcomer's DHT routing table. At the same time, the newcomer establishes *Bitswap* [36] connections to a subset of encountered peers (usually around 300 of them). The core role of the *Bitswap* protocol is to enable bilateral content transfer and to play the role of a cache for recently-accessed content.

The main DHT operation GetClosestPeers(key) returns the $k = 20$ closest peers to key. In Kademlia, the distance between two keys x and y in the key space is given by $x \oplus y \in \{0, ..., 2^{256}-1\}$, where $\oplus$ denotes the bitwise XOR operation on the keys; the resulting binary string is interpreted as an integer. When a client wants to find the peers with IDs closest to key, it sends a request to the $\alpha = 3$ peers in its routing table whose peer IDs are closest to key. Each of these peers returns the k closest peers to key in its own routing table and the addresses of these peers. The client again sends a request to the α peers closest to key, among peers in its routing table and those whose addresses it just received. This process repeats until the client does not find any more peers closer to key. Due to network churn and imperfect routing tables, we observed in our experiments that successive calls to GetClosestPeers(key) do not always return the true set of $k = 20$ closest peers (we provide more details in Section VIII, Figure 6).

B. Content Resolution in IPFS

IPFS is a content-centric network. It allows its participant to request files without specifying their location. Content is indexed by content IDs cid $\in \{0,1\}^{256}$ that are derived from a hash of that content. Both peer IDs and CIDs are used as keys in the DHT. Each node can play the role of a provider, downloader, or resolver. The process of content advertisement and resolution is illustrated in Figure 1.

When a provider wishes to publish content with a given cid on IPFS, it creates a *provider record* that contains *cid* and the provider's address. During a Provide(cid) operation, the provider first uses GetClosestPeers(cid) to locate the $k = 20$ peers with their peer IDs closest to cid, and then sends them a PutProvider message including the provider record (Figure 1(a)). We call the peers that hold provider records for cid the *resolvers* for cid.

Each CID can have several providers. In fact, by default, each IPFS client becomes a provider for each piece of content it downloads for a fixed amount of time (12h, 24h, or 48h depending on the client version or custom configuration). As a result, the system provides an auto-scaling feature with supply automatically rising with demand.

When a downloader wishes to fetch a piece of content, it first sends a request to all its *Bitswap* peers. If none of them has the content, the downloader uses the DHT-based resolution system. We stress that the *Bitswap* protocol plays the supporting role of a cache in the dissemination of popular files. However, the mechanism does not provide reliable content resolution, in particular for new or less popular content.

When *Bitswap* unstructured search fails, the downloader resolves cid using FindProviders(cid). This operation uses a DHT walk identical to that of GetClosestPeers(cid) to find k resolvers but also queries encountered nodes for a provider record for cid (Figure 1(b)). The process terminates when either 20 providers have been found, or all resolvers have been asked. Querying all encountered nodes (*i.e.*, not only the designated resolvers) is useful because some of the encountered nodes may have a provider record in their cache.

Upon receiving a provider record, the client connects to the address specified in the provider record to retrieve the actual content (Figure 1(c)). Provider records are not authenticated, and therefore malicious providers may respond with incorrect provider records (or may not respond at all). However, the integrity of the content is preserved because the hash of the retrieved content can be verified against its cid.

C. Network Size Estimator

The number of nodes in a decentralized system is generally unknown due to the avoidance of centralized membership management. This number is nonetheless useful for optimizations, deciding on individual node configurations, or security mechanisms. Various methods were proposed for the decentralized estimation of unstructured and structured networks [47], [50], [61]. We use in this work a mechanism developed initially by Protocol Labs as part of a mechanism for decreasing the latency of publishing content in IPFS [17], [65].

Each node in the DHT refreshes its routing table periodically (every 10 minutes in `libp2p`). For this, the node samples m random keys (one for each bucket of its routing table) and queries the DHT to obtain the $k = 20$ closest peer IDs to each key. Using these, the node then computes the average distance between each one of these keys key_j for

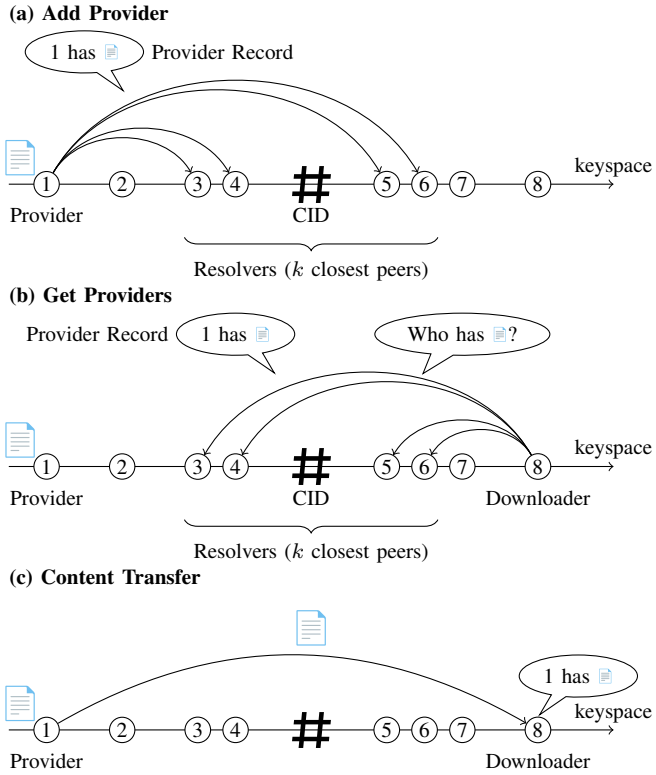

Fig. 1: Content resolution using the DHT. 1) The provider upload its provider record to designated resolvers. 2) The downloader fetches the provider record from resolvers. 3) The downloader uses the information in the provider record to download the content directly from the provider.

$j = 1, \ldots, m$ and their i-th closest peer ID for $i = 1, ..., k$ (with $m = 256$ and $k = 20$).

$$\overline{D}_i = \frac{1}{m} \sum_{j=1}^{m} \text{dist}(\text{key}_j, \text{peerid}_j^{(i)}) \tag{1}$$

where $\text{peerid}_j^{(i)}$ is the i-th closest peer ID to key_j. With N peers in the DHT and peer IDs uniformly distributed in the hash space, the expected distance between a key and its i-th closest peer ID is $\frac{2^{256} i}{N+1}$. The node then runs a least square regression to compute the value of N for which the expected distances best fit the empirical average distances, *i.e.*,

$$\hat{N} = \arg\min_{N} \sum_{i=1}^{k} \left(\overline{D}_i - \frac{2^{256} i}{N+1} \right)^2 . \tag{2}$$

The resulting estimate $\hat{N}$ can be computed in closed form.

When a node starts running, it must perform DHT queries for a few random keys to initialize its network size estimate. Since a larger number of queries will result in higher accuracy, making more queries than what is needed to initialize one's routing table is recommended. Thereafter, keeping the estimate up-to-date does not require any excess DHT queries beyond what is already used for refreshing the routing table as this is done frequently (every 10 minutes).

While the network size estimate has a stochastic variance resulting from the probability distribution of the honest peer IDs, it is hard for an attacker to bias the estimate significantly. Since the estimator uses the density of peer IDs around keys chosen uniformly at random, the adversary would require numerous Sybil nodes (on the order of the whole network size) to significantly affect the peer ID density around those keys.

IV. Threat Model

We assume N DHT nodes participating in the IPFS network. Multiple nodes may share the same IP address (due to NAT or being hosted by the same physical machine) [51]. However, two nodes cannot share the same ID.

We assume the presence of malicious actors in the network that may refuse to store valid provider records and distribute these to honest participants. Malicious actors can spawn multiple virtual nodes within one physical machine, operate multiple physical machines, and coordinate their actions. We assume that no honest node is fully eclipsed by malicious ones, *i.e.*, each honest node has *at least* one honest peer and the DHT routing allows honest nodes to reach any key and discover other honest peers. IPFS already implements multiple mechanisms preventing eclipse attacks at the DHT level [56].

An attacker running Sybil nodes interfaces with the network using regular IPFS operations. We do not rely on bugs present in the operating system or any other components not related to the P2P network node implementation under attack. Any flaw in the P2P system protocols, however, may be exploited as these are considered part of the attack target. Non-Sybil DHT nodes (ones that are not spawned by the malicious actor) are assumed to be configured and operated as intended. Thus, importantly, the attacker only controls Sybil nodes that they create but does not corrupt or bring down any other nodes.

The goal of the attacker is to prevent downloaders from obtaining provider records for a target CID. This leads to *content censorship* as the downloader cannot find a provider to obtain the content from (recall that *Bitswap* is only a cache for popular content while the DHT is required for reliable content discovery). We measure the attack effectiveness a_{eff} as the ratio of unsuccessful FindProviders(cid) queries to the total number of queries for existing content cid issued by honest downloaders. A query is unsuccessful if it does not return any honest provider record. The attack effectiveness may vary depending on the placement of the CID and the downloader issuing the request in the hash space. We thus always consider a_{eff} as an average for multiple CIDs and multiple downloaders, both uniformly spread across the hash space.

The goal of honest participants is to detect the attack and mitigate its effects, *i.e.* to enable downloaders to discover valid provider records and later fetch the content despite the actions of the attacker.

To assess the effectiveness of our *detection* mechanism, we use the false positive f_p and false negative f_n rates. The false positive rate f_p is the proportion of erroneous detections (*i.e.* when there was no attack). The false negative rate f_n is the proportion of attacks that are not detected. A detection leads to a mitigation action. A false positive leads, therefore, solely to

General parameters	
N	Network size (number of nodes)
k	Bucket size, resolvers per CID, and number of closest peers obtained in a GETCLOSESTPEERS(cid) call (currently, $k = 20$ in libp2p/IPFS)
Attack general parameters	
a_{eff}	Effectiveness of the attack [%]
e	Number of Sybil nodes
$e(a_{\text{eff}})$	Number of Sybil nodes necessary to perform the attack with effectiveness a_{eff}.
Attack costs	
$s(e)$	Brute-force attempts necessary to generate e Sybil identities that are the closest to a target CID.
c_{gen}	Cost of generating $s(e)$ Sybil identities [$]
c_{oper}	Cost per unit time of operating e Sybil nodes to attack a single CID [$/$s$]
c_{att}	Total cost of attacking a single CID [$]
Attack performance	
t_{w}	Warmup time during which the e Sybil nodes need to be run but the attack is not yet fully effective [s]
t_{eff}	Time after t_{w} during which the attack remains fully effective. The attack maintains its effect only as long as the e Sybil nodes are present in the network [s].
Detection and mitigation performance	
thr	Threshold for the detection mechanism
f_n	Detection false negative rate [%]
f_p	Detection false positive rate [%]
m_{eff}	Effectiveness of the mitigation [%]

TABLE I: Parameters and characterization of the attack, detection, and mitigation.

additional overhead, while a false negative leads to effective censorship of content. Henceforth, we favor minimizing the false negative rate f_n.

The mitigation effectiveness m_{eff} is the ratio of the number of successful FINDPROVIDERS(cid) queries to the total number of queries issued by honest downloaders when the target, existing cid is under attack and the mitigation mechanism is used. A successful query is defined as one that returns at least one honest provider record. Similarly to the attack effectiveness, we report m_{eff} as an average for queries issued for multiple CIDs by multiple downloaders uniformly spread across the hash space.

V. CID CENSORSHIP ATTACK

We now proceed to detail our content censorship attack, targeting a specific victim CID. To describe and analyze this attack, we use a Sybil attack model with notations adapted from prior work [56], [57] and summarized in Table I.

A. Attack process

Overview. In IPFS, provider records for a target CID should be stored on the $k = 20$ peers closest to that CID (*i.e.* resolvers). We generate $e \geq k$ Sybil identities that are closer to the target CID than to the closest honest peer. As a result, those Sybil identities receive all new provider records from providers and resolution queries from downloaders for that CID. Figure 2 illustrates the position of Sybil peers. Sybils drop provider records and do not respond to queries for that target CID.

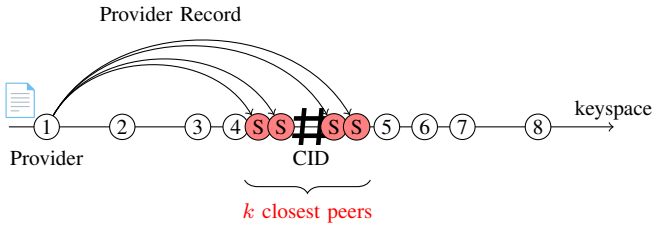

Fig. 2: Illustration of a censorship attack: Attacker places k Sybil peers ($k = 4$ in this example) closer to the CID than any honest peers, so provider records are now only sent to Sybil peers. Similarly, requests to obtain provider records are sent to the Sybil peers, who can ignore them.

Details. The attack proceeds as follows. First, we use the IPFS API to retrieve the current $k = 20$ closest peer IDs to the target CID, sort them by distance to the CID, and identify the closest one. We then repeatedly generate random public/private key pairs and compute new peer IDs by hashing the public key. If a generated peer ID is closer to the CID than to the currently closest peer ID, we keep the corresponding key pair, otherwise, we discard it. We repeat the process until we obtain e peer IDs that are closer to the target CID than any of the original 20 closest peer IDs.

For each generated Sybil peer ID, we spawn a custom libp2p DHT node (server) that joins the IPFS network. These Sybil nodes drop received provider records for the target CID and respond with empty messages to received resolution queries for the target CID. The Sybil nodes behave normally for other CIDs, so that our experiments do not affect the rest of the network. The attacker continuously monitors the set of e closest peers to the target CID to make sure it contains only the Sybil nodes. If a new, honest node appears in the set, the attacker reacts by generating additional identifiers to maintain the desired number of Sybil nodes in the set.

B. Attack analysis

We analyze the attack in terms of cost and in terms of effectiveness, including its timing.

Initial costs. The first cost for an attacker is that of generating Sybil identities. As the hash function is pre-image resistant, this process must use a brute force generation of private/public key pairs and associated IDs. The number of attempts it takes for generating e Sybil identities that are closer to the target is denoted $s(e)$. This number naturally depends on e, but also on the distance of the closest honest peer from the target CID, which in turn depends on the number of peers in the network. The closer the honest peer is to the target CID, the more keys the attacker needs to generate to obtain Sybil peer IDs closer to the CID.

The number of attempts further translates into an operational cost which we quantify using public cloud resource

costs. This cost c_{gen} depends on $s(e)$ and the cost of generating one private/public key pair. IPFS and `libp2p` support both the RSA and Edwards-curve Digital Signature Algorithm (EdDSA) cryptosystems. The generation of keys for EdDSA is significantly faster than for RSA, and both are embarrassingly parallel; we evaluate these costs in Section VIII, and choose to target EdDSA in our implementation of the attack

The effectiveness of the attack a_{eff} varies with the number of Sybils e. Theoretically, the attack only requires $e = k = 20$ Sybil identities. However, different DHT nodes do not always discover the same set of $k = 20$ closest peers (we provide experimental evidence in Section VIII, Figure 6). As a result, the attack generally requires more than 20 Sybil identities (as some further honest peers may be discovered). On the other hand, some of the discovered peers may not be online, so the attack may also succeed with fewer than 20 Sybil identities. We empirically studied the effect of the number of Sybil identities on the rate of successfully censoring the target content, and observe that $e = 45$ Sybil identities can censor content with a $a_{\text{eff}} = 99\%$ probability of success (Section VIII, Figure 7). While $e(a_{\text{eff}})$ depends on k, $e(a_{\text{eff}})$ does not depend on whether the content was already provided before the Sybils were launched or not.

Timing. The *warmup time* t_{w}, *i.e.*, the time before the content is effectively censored and becomes undiscoverable, depends on whether the Sybil peers are launched *before* the provider sends its provider records to the network or the Sybil peers are launched *after*.

If Sybil peers are launched before the first provider sends its provider record to the network then, since the Sybil peers are the closest peers to the target CID, providers will most likely send their provider records to only Sybil peers. These Sybil peers simply drop the provider records. As a result, the content never becomes discoverable in the network. Therefore, if all Sybil nodes are launched before the first provider advertised the content, the attack is effective immediately, i.e., $t_{\text{w}} = 0$. We note, however, that this best-case scenario is not likely in all contexts of use of IPFS, as it requires knowing the CID of the content to censor before mounting the attack. This CID depends, indeed, on the *content* of the file which may be known only upon its publication. In certain cases, however, the attacker may know in advance that a specific file will be published and act to prevent its discoverability.

If the provider records were already stored on honest peers before the Sybil peers were launched, a downloader may encounter an honest peer with relevant provider records *before* reaching the Sybil peers, and be able to obtain the provider records this way. By default, however, such provider records expire every 48 hours[3], after which a provider must call PROVIDE(cid) again. As a result, in the worst case, the last provider record on an honest resolver will be removed $t_{\text{w}} = 48$h after launching the Sybil nodes, after which the content becomes censored. It is not desirable to get rid of this limited lifetime of provider records: an unlimited lifetime would result in a gradual overload of long-running peers and open new avenues for DoS attacks.

Overall costs. The overall costs of the attack c_{att} include the initial costs c_{gen} plus the operational costs of running the Sybil nodes at c_{oper} per unit time. The operational cost is incurred during the warmup time t_{w} before the attack is effective and the time during which the attack must *remain* effective t_{eff}. Therefore, $c_{\text{att}} = c_{\text{gen}} + (t_{\text{w}} + t_{\text{eff}}) \times c_{\text{oper}}$.

[3] 24 hours in older versions of `go-libp2p`

VI. CENSORSHIP ATTACK DETECTION

The first step in countering an attack is its reliable detection. It enables activating mitigation techniques only when needed and avoids unnecessary overhead when the network is not under attack. Importantly, the detection cannot simply rely on the unavailability of the provider records as this could be due to other reasons (e.g., the provider records expired and the provider did not renew them). Moreover, the detection method should ideally expose an attack as soon as the Sybil peers are added to the DHT, even though unavailability of the provider records may only occur several hours ($t_{\text{w}} = 48$ hours) later (as described in Section V-B). This would help prevent downtime for the content.

To execute the content censorship attack, the attacker must hijack all PutProvider advertisements for the target CID. To this end, the attacker must operate e Sybil peers whose IDs are closer to the target CID than to any other honest peer. As a result, when an honest node queries the DHT for the target CID, the 20 closest peer IDs it finds are closer to the CID than usual. The node can thus use the observed peer IDs to detect whether the CID is censored or not.

Method Overview. We repurpose a statistical method originally developed by Cholez *et al.* [30] for detecting eclipse attacks. This method first obtains the $k = 20$ closest peer IDs to the CID, using a DHT query, and computes the common prefix length of each peer ID with that CID (*i.e.* the number of leading bits that match both the CID and the peer ID). It then compares the empirical distribution of the 20 common prefix lengths with the 'model' probability distribution that would result if all these peers were honest, *i.e.*, if their peer IDs were chosen randomly and uniformly. We compare the empirical distribution to the model distribution by computing the Kullback-Liebler (KL) divergence between the two distributions (also known as a G-test [53]). A large KL divergence indicates a mismatch between the empirical and model distributions, indicating an attack. On the other hand, a small KL divergence indicates a good match of the distributions, indicating no attack. This KL divergence-based has been shown to perform well under a small number of samples [62], which is our situation with only 20 samples corresponding to the 20 closest peers.

A challenge of implementing this detection method for the IPFS network is that it requires knowing the number of peers in the DHT to compute the model distribution. Cholez *et al.* [30] estimated a model distribution based on measurements of the KAD network (an implementation of Kademlia used by the eDonkey [8] network). However, such an approach does not work when the number of peers in the DHT changes over time, thus changing the model distribution. In this work, we show how to adapt this detection mechanism to a DHT of dynamic size by using a network size estimator. Based on the network size estimate, our detection method computes an estimate of the model distribution, which is then used

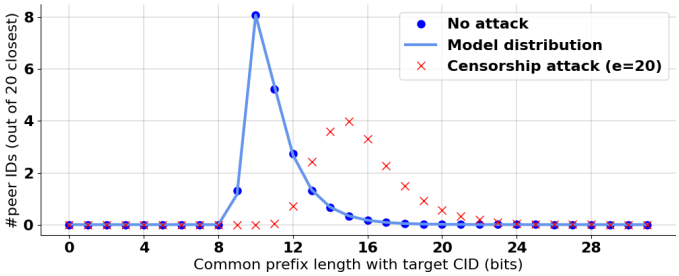

Fig. 3: Probability distribution of common prefix lengths of the target CID with its $k = 20$ closest peer IDs.

for detection. Our detection method does not require any additional communication and requires little local computation.

Method Details. Honest peers choose their peer IDs uniformly at random, and thus under no attack, peer IDs in the DHT are distributed uniformly across the hash space. We thus expect the common prefix lengths of peer IDs with the target CID to follow a geometric distribution. That is, given a target CID, the probability that the common prefix length of a randomly chosen peer ID with the target CID equals x is 0.5^{x+1} (effects of the finite length (256 bits) of IDs can be neglected as the probability of a common prefix length of 256 bits is negligible). However, when a node queries the DHT for the target CID, it only obtains the $k = 20$ closest peers to the target CID, and the distribution of their common prefix lengths is not geometric. Particularly, very small common prefixes are not observed as such peer IDs are far from the target ID (see Figure 3). In the work of Cholez *et al.* [30], this effect is modeled by using a geometric distribution conditioned on the common prefix length being greater than L, where L is a parameter that is estimated using measurements from the network. If we adopted this approach, we would have to compute L from a fixed, pre-determined network size estimate. We found that the model distribution computed using this method is highly susceptible to small variations in the network size estimate. Instead, in this work, we compute the model distribution dynamically, *i.e.*, as it varies according to an estimation of the network size.

We will first describe the method assuming that the network size N is known. Then, we will show how to adapt this method using an estimation of the network size. We recall that the common prefix length of a randomly chosen peer ID with the target CID is distributed geometrically. Now, in response to a lookup for the target CID, a node obtains the k largest common prefix lengths (corresponding to the k smallest distances). The cumulative distribution function of the j-th largest prefix length can be calculated as

$$F_j(x) \triangleq \Pr(X^{(j)} \leq x) = \begin{cases} \sum_{i=0}^{j-1} \binom{N}{i} \left(1 - 0.5^{x+1}\right)^{N-i} 0.5^{(x+1)i} & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{3}$$

Since we only receive one sample for the j-th best peer ID for each j, we can compute an average probability mass function of the k best common prefix lengths as

$$p(x) = \frac{1}{k} \sum_{j=1}^{k} (F_j(x) - F_j(x-1)). \tag{4}$$

This model distribution $p(x)$ is shown along with the empirical average distribution of common prefix lengths in Figure 3. Note that the empirical distribution in case of no attack (all peer IDs generated randomly) exactly matches the model distribution, while the distribution under an attack ($e = 20$ Sybil peers closer than honest peer IDs) significantly differs from the model. Importantly, placing more than 20 Sybils drives the ID distribution further away from the model, easing the detection. Since $k = 20$, the sums in eqs. (3) and (4) can be computed efficiently.

The KL divergence is a tool used to quantify the difference between two probability distributions. For two discrete probability distributions $p(x)$ and $q(x)$ on a support set $\mathcal{X}$, the KL divergence from p to q is defined as

$$D\left(q \parallel p\right) \triangleq \sum_{x \in \mathcal{X}} q(x) \ln \left(\frac{q(x)}{p(x)} \right). \tag{5}$$

In our case, X is the random variable denoting the common prefix length between a peer ID and the target CID, $\mathcal{X} = \{0, ..., 256\}$, $p(x)$ is the model distribution and

$$q(x) \triangleq \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\{X_i = x\} \tag{6}$$

is the empirical distribution of common prefix lengths. Note that the support of the empirical distribution $q(x)$ is a subset of the support of the model distribution $p(x)$, therefore the sum in the KL divergence is only computed over values of x for which $q(x) > 0$. Now, a threshold thr must be chosen so that the CID is flagged to be under an attack if and only if $D\left(q \parallel p\right) > \mathsf{thr}$.

Adapting to Dynamic Network Sizes. Computing the model distribution requires knowing the number of peers in the DHT (N) that is unknown to the DHT nodes. Instead, we substitute an estimate $\hat{N}$ of the network size obtained from the network size estimator we described in Section III-C. A node can locally estimate the model distribution by substituting $\hat{N}$ instead of N in eqs. (3) and (4). The complete censorship detection algorithm is specified in Algorithm 1.

It is important to note that the network size estimate does not depend on the closest peer IDs for the CID for which censorship detection is being performed. Instead, it is computed using the closest peer IDs to randomly chosen keys. Therefore, recall from Section III-C that the attacker would require a number of Sybil peers of the order of the network size to bias the estimator. As a result, the detection remains robust under the censorship attack.

Choosing a detection threshold. The detection threshold thr is a per-node constant value. A higher threshold results in more false negatives (some attacks go undetected, hence unmitigated) while a lower threshold results in more false positives (mitigation overhead when there is no attack). In Section VIII, we evaluate the false positive and false negative

Algorithm 1 Censorship Detection Algorithm; thr is a pre-decided detection threshold

```
1: procedure DETECTION(key)
2:     peers ← GET20CLOSESTPEERS(key)
3:     q ← numPeersPerCPL(peers)/20                ▷ eq. (6)
4:     N ← GETNETSIZEESTIMATE()                    ▷ eqs. (1) and (2)
5:     p ← computeModelDist(N)                     ▷ eqs. (3) and (4)
6:     KL ← computeKL(p, q)                        ▷ eq. (5)
7:     return KL > thr                             ▷ true indicates attack
```

rates for different thresholds and recommend a threshold that favors reducing false negatives. Different nodes can however choose different thresholds according to their desired trade-off between the error rates.

VII. Mitigation with Region-Based Queries

Countering Sybil attacks in an open, decentralized system is challenging. Traditionally, this problem is solved by binding identities to valuable resources (*e.g.* using Proof of Work [26], [37]), certificate authorities [28], reputation systems [33], [48], [70], [71], or diversifying the IP addresses of the peers of each node [56]. Proof of Work and IP address restrictions are not sufficient as our attack only requires a few Sybil peers ($e \approx 45$): these measures would only slightly increase the cost of the attack. On the other hand, certificate authorities and reputation systems hamper the decentralization and open participation model of IPFS. Simply increasing the value of the number of closest peers contacted in a DHT query (currently $k = 20$) also does not solve the problem as the attacker only needs to generate more Sybil peers to match the new number. Another naive idea is that the providers modify the content by one bit to modify its CID. However, the new CID must be then advertised to potential downloaders to make the content publicly accessible. The attacker can then simply "follow" the new CIDs and continuously censor the content. Further, modifying the content is unsuitable for immutable Web3.0 content (e.g., NFTs and DIDs) whose hash is already published on a blockchain.

The fundamental problem in countering the content censorship attack lies in the inability to classify resolvers as honest or malicious. When a downloader receives no provider record or an inactive provider record from a resolver (*i.e.*, it is unable to find the referenced provider or the provider does not hold the content), this can be due to several reasons. For instance, the resolver was offline, the record used to be correct but the provider had since left, or there was a network failure. As a result, the downloader cannot draw any conclusions on the resolver's correctness based on the received results, eliminating any attempts to gradually filter out malicious nodes by local scoring systems.

Main idea. The core observation behind our approach is that, while an attacker can spawn additional Sybil identities, it has no way of removing the honest ones from the network. As long as the provider can send its provider record to the initial honest resolvers, and the downloader can communicate with these resolvers, the censorship attack will be mitigated. To maintain communication with the initial honest resolvers even during an attack, we propose *region-based* DHT queries.

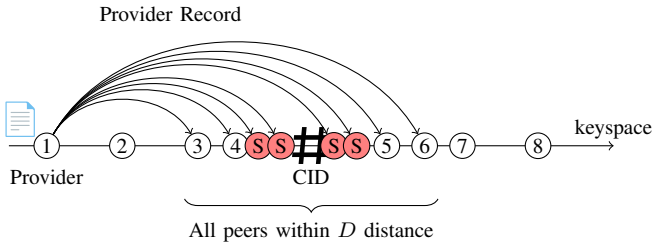

Fig. 4: Illustration of our mitigation: A provider record is sent to all peers within a region that contains k honest peers on average ($k = 4$ in this example), as estimated using the network size. This region contains k honest peers even in the presence of Sybil peers, as honest peers are not removed.

Algorithm 2 Function to find all peers with a Common Prefix Length (CPL) $\geq$ minCPL with key

```
1: procedure FINDBYCPL(key, minCPL)
2:     set ← GETCLOSESTPEERS(key)
3:     CPL ← minCommonPrefixLength(set, key)
4:     while CPL ≥ minCPL do
5:         qkey ← key[:CPL] || key[CPL] || key[CPL + 1:]
6:         set ← set ∪ FINDBYCPL(qkey, CPL + 1)
7:         CPL ← CPL − 1
8:     removeItemsWithPrefixLessThan(set, minCPL)
9:     return set
```

Rather than communicating with the $k = 20$ closest peers to a CID, that an attacker can easily control, we communicate with all the nodes in the hash space region that $k = 20$ uniformly distributed peer IDs should cover (Figure 4). The size of this region is calculated using the network size estimate and using the assumption that honest peer IDs are distributed uniformly over the key space. Such an approach ensures that regardless of the number of Sybil nodes placed by an attacker, the provider can store provider records on ≈ 20 honest resolvers, and the downloader also communicates with $\approx$ 20 honest resolvers to reliably retrieve the correct provider records. To prevent additional overhead when there is no attack, we run the region-based queries only when an attack is detected using the detection mechanism that we detailed in Section VI.

Algorithm. The region-based query algorithm is described in Algorithm 2, with a sample execution in Figure 5. The goal of this algorithm is to find all peer IDs that share a common prefix of at least minCPL bits with key. Note that any two keys $\mathsf{k}_1, \mathsf{k}_2$ have a common prefix length (CPL) of at least l iff the XOR distance between k_1 and k_2 is less than 2^{256-l}. Therefore, the common prefix requirement specifies a region of the key space with a distance $2^{256-\mathsf{minCPL}}$ from key. To keep our mitigation compatible with the current version of `libp2p` DHT nodes, we use the same RPCs that the DHT nodes currently use. Therefore, we build the algorithm using only calls to GETCLOSESTPEERS(key) which obtains the 20 peer IDs that are the closest to key, which is already available in `go-libp2p-kad-dht` [15]. We start with this primitive and then compute the common prefix length shared by key and all of its 20 closest peer IDs, which we note as CPL.

Since we have found at least one peer ID with a common

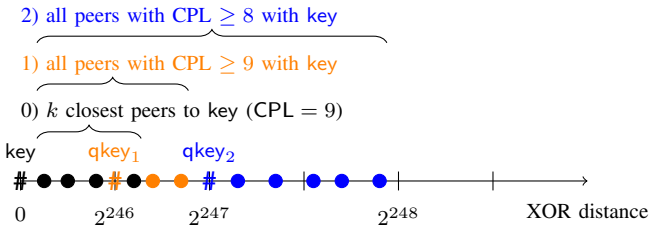

Fig. 5: An example illustration of FINDBYCPL(key, 8) (see algorithm 2). Note that any two keys k_1, k_2 have common prefix length (CPL) at least l iff the XOR distance between k_1 and k_2 is less than 2^{256-l}. Step 0: Find the k (4 in this example) closest peers to key (shown in black). Calculate their minimum common prefix length with key (CPL $= 9$ in this example). Step 1: FINDBYCPL(qkey$_1$, 10) returns the green peers with CPL ≥ 10 with qkey$_1$. Now we have found all peers with CPL ≥ 9 with key. Step 2: FINDBYCPL(qkey$_2$, 9) returns the blue peers. Now we have found all peers with CPL ≥ 8 with key, and the algorithm terminates.

prefix length CPL, we must have found all peer IDs with common prefix length $\geq$ CPL $+ 1$, as the latter are closer (in XOR distance) to key than the former (see step 0 in Figure 5). In the next step, we would like to find all peer IDs with common prefix $\geq$ CPL with key. Since we have already found all peer IDs with the prefix key[:CPL$+1$] (*i.e.*, the first CPL$+1$ bits match key), we only need to find all peers IDs with the prefix key[:CPL] $\|$ $\overline{\text{key[CPL]}}$ (*i.e.* the first CPL bits match key and the (CPL $+ 1$)-th bit is different). This is done recursively using our algorithm. This step is repeated until all peer IDs with common prefix length $\geq$ minCPL with key have been found.

While using this region-based query algorithm, we choose the value of minCPL such that a region of the key space with common prefix length at least minCPL with key $=$ cid contains at least $k = 20$ honest peer IDs with high probability. Suppose that there are a total of N peer IDs in the DHT, distributed uniformly across the hash space. A common prefix length of at least minCPL corresponds to a XOR distance $< 2^{256-\text{minCPL}}$. Then, the expected number of peer IDs in this region is $2^{\text{minCPL}} \times N$. By setting minCPL $= \lceil \log_2 \left(\frac{N}{k}\right) \rceil$, we have a region that contains k honest peer IDs on average. Given an estimate $\hat{N}$ of the network size (as described in section III-C), we substitute $\hat{N}$ for N to calculate the region size. By a simple probabilistic bound, we can also extend this region to contain k honest peer IDs with high probability.

Cost Analysis. By default, both providers and downloaders do one DHT lookup (using GETCLOSESTPEERS(key)) to obtain the list of $k = 20$ closest peer IDs to key. When a provider or downloader uses a region-based query, it does multiple lookups using GETCLOSESTPEERS$(\cdot)$. The number of lookups required increases sub-linearly in the number of Sybil identities placed by an attacker (shown experimentally in Figure 15). Importantly, operating a Sybil identity requires participating in the DHT routing and responding to keep-alive messages. As a result, the cost for the attacker increases linearly with the number of Sybil identities. When the target CID is not under attack, using the region-based query would still use more than one GETCLOSESTPEERS$(\cdot)$ lookups, because honest peer IDs are distributed randomly, and therefore the chosen region might contain more than 20 peer IDs. To avoid this overhead when there is no attack, we run the region-based lookup only when the detection mechanism (Section VI) detects an attack, and use the default lookup otherwise. We evaluate the provider's and downloader's cost of the region-based queries (number of lookups and latency) and the attacker's cost in Section VIII.

Correctness Analysis. We prove that Algorithm 2 indeed finds all peer IDs with a common prefix length of at least minCPL with key.

Theorem 1. *Assuming that* GETCLOSESTPEERS(key) *returns the* 20 *closest peer IDs to* key, FINDBYCPL(key, minCPL) *returns all peer IDs with a common prefix of at least* minCPL *bits with* key.

Proof: We prove this claim through induction. For the base case, if there are $<$ 20 peer IDs with a common prefix length of at least minCPL with key, then GETCLOSESTPEERS(key) must return at least one peer with a common prefix length $<$ minCPL with key. Therefore, CPL $<$ minCPL, hence the function returns all peer IDs that have a common prefix length of at least minCPL.

Otherwise, CPL $\geq$ minCPL. Since we have found at least one peer with a common prefix length CPL, we have found all peers with a common prefix length $\geq$ CPL$+1$, that is all peers with the prefix key[: CPL$+1$]. Thus, we create a new key qkey which has the prefix key[: CPL] $\|$ $\overline{\text{key[CPL]}}$. By induction, we assume that FINDBYCPL(qkey, CPL$+1$) returns all peers with prefix key[:CPL] $\|$ $\overline{\text{key[CPL]}}$. Together, we now have all peers with the prefix key[:CPL]. After subtracting 1 from CPL, we maintain the invariant that we have found all peers with prefix key[:CPL $+ 1$]. If the loop doesn't quit, then we continue to find peers with one more bit in the common prefix in every iteration. If the loop quits, this means that CPL$+1 \leq$ minCPL, therefore we have found all peers with common prefix length of at least minCPL as promised. ■

Even if GETCLOSESTPEERS(key) does not return *all* of the 20 closest peer IDs to key, the algorithm will still terminate (because CPL decreases at every iteration) but may not find all the peers with a common prefix length of at least minCPL. Since we restrict ourselves to build the region-based lookup using GETCLOSESTPEERS(key), our method is accurate only in the cases when GETCLOSESTPEERS(key) is accurate.

VIII. EVALUATION

In this section, we evaluate the cost of the attack and the effectiveness of our detection and mitigation methods.

A. Setup

We perform our experiments on the live IPFS networks. All our attacks targeted only content that we created. No content provided by other users was affected. Although we did not require any special permissions from Protocol Labs to execute our attacks, we informed them for responsible disclosure.

In the evaluation, we use three types of DHT nodes: *(i)* the malicious Sybil nodes, *(ii)* a provider node hosting the content

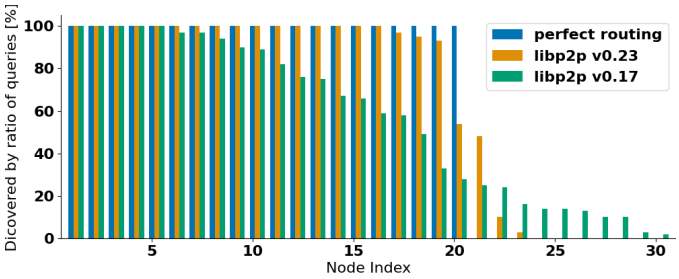

Fig. 6: Distribution of nodes discovered during a DHT query by different versions of IPFS clients, across 100 experiments.

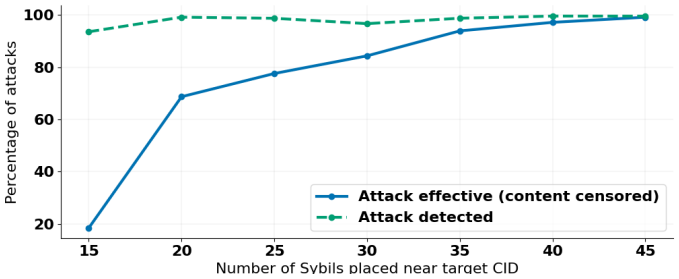

Fig. 7: Percentage of attacks in which *i)* the attack was effective and *ii)* the detection algorithm detected the attack.

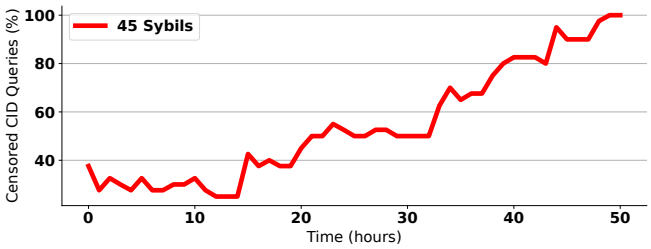

Fig. 8: Rate of censored CID queries (%) over time, from the start of launching the attack (when content has been provided before adding Sybil nodes).

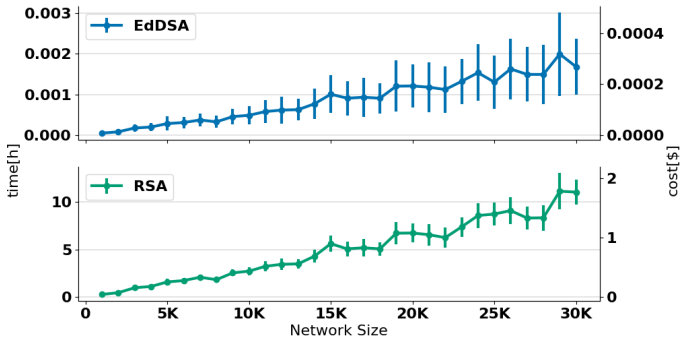

Fig. 9: Average time and cost required to generate necessary Sybil identities.

that we created, and *(iii)* a downloader node that attempts to resolve the target CID and fetch its content. All these nodes were hosted on a single AWS t3.xlarge instance with 4 vCPUs and 16 GiB memory. The attacker node is implemented as a custom DHT client using `libp2p` [14]. Our mitigation and detection methods are also implemented on top of `libp2p`. Except for the experiment of Figure 8, the provider sends the provider record (*i.e.*, it provides the content) after the Sybils are launched so that we avoid waiting for $t_{\text{w}} = 48$ hours for the attack to take effect.

B. DHT Lookup Accuracy

The DHT query GETCLOSESTPEERS(key) is ideally expected to return the $k = 20$ closest peer IDs to the queried key. Figure 6 shows the number of GETCLOSESTPEERS(key) queries in which each peer close to the queried key is discovered. The x-axis shows the index of nodes ordered by their distance to the queried key (1 is the closest node). The y-axis shows the fraction of queries in which each node was discovered. The results are averaged over 100 experiments. The set of peers obtained is compared with the true 20 closest peers ('perfect routing') that were obtained from a network crawler. We observed that when different nodes perform this query, and when the same node does it multiple times, the set of 20 peers received in response is not always the same. This effect is present in both versions of `libp2p` considered here but the query responses are more consistent in the newer version. To mitigate this effect, the attacker has to use a larger number of Sybils $e > 20$ for high attack effectiveness $a_{\text{eff}} \to 100\%$.

C. Attack

We follow by determining the number of Sybil identities e required to achieve high attack effectiveness a_{eff}. Since the DHT queries do not consistently return the same set of peers, downloaders may discover some honest resolvers when there are only $e = 20$ Sybils. Figure 7 shows the success rate of the attack as the number of Sybil peers varies. Even when $e < 20$, the attack sometime succeeds because too few honest resolvers are contacted, and they may be offline. Involving more Sybil peers increases the chance of a successful attack but also proportionally increases the cost for the attacker. With $e = 45$ Sybils, the attack succeeds with 99% probability. We use this value for the rest of the experiments.

When the Sybil nodes are placed before the target content is added to the network, the attack takes effect immediately ($t_{\text{w}} = 0$). However, we also explore a more realistic timeline in which content has been provided before launching the attack. Figure 8 presents the evolution of the attack effectiveness a_{eff} over time. For each censored CID, we spawn 5 downloader nodes per hour. We stop the experiment when all queries have been unsuccessful for at least 3 hours. Immediately after starting the attack, the effectiveness reaches $a_{\text{eff}} = 30\%$. This is caused by a portion of the downloaders not encountering any honest resolvers on their path toward the attacked region. The effectiveness steadily increases over time. Multiple spikes (*e.g.* after 12h) are caused by a portion of resolvers (running older IPFS versions) dropping the provider records. The attack takes full effect after 48 hours, which is when resolvers drop the provider record as per the current IPFS version. Based on this result, we assume a maximum warmup time $t_{\text{w}} = 48$h in the cost calculations that follow.

Finally, we explore the *cost* of performing the attack. The AWS instances running the Sybils cost $c_{\text{oper}} = 0.16$\$ per hour. Figure 9 presents the time and monetary cost of generating

$e = 45$ Sybil identities for both cryptosystems present in IPFS. For readability, we omit the number of iterations s, as it is the same for both methods and proportional to the cost/time. The monetary cost c_{gen} and the generation time increase linearly with the network size. In larger networks, the distance between the closest honest resolver and the target CID decreases and the generation algorithm requires more iterations. EdDSA is significantly faster than RSA and the generation time remains below 12s translating into 0.0005$, even for the largest evaluated network with $n = 30,000$ nodes. Generating Sybils using RSA, while slower, is feasible even for moderately resourceful attackers.

The peak CPU utilization of 30% occurred only during the generation of Sybil private keys. The maximum bandwidth utilization of the machine hosting $e = 45$ Sybils was 4.67 Mbps (both inbound and outbound) when no requests were made for the censored CID. Taking the cost of generating Sybil identities $c_{\text{gen}} = 0.0005$\$, the longest warmup time $t_{\text{w}} = 48$h, and the duration of the attack t_{eff} hours, the total cost of the attack using EdDSA is given by $c_{\text{att}} = c_{\text{gen}} + (t_{\text{w}} + t_{\text{eff}}) \times c_{\text{oper}} = 7.68 + t_{\text{eff}} \times 0.16$\$.

D. Detection

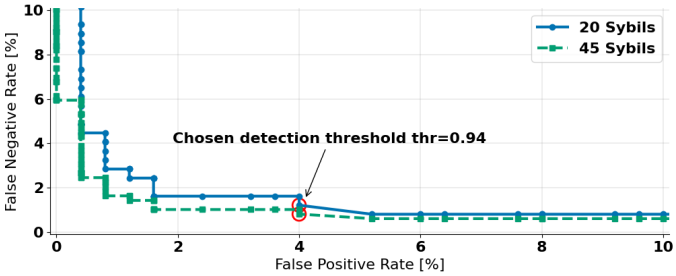

Fig. 10: False positive and false negative rates of the detection method, for different detection thresholds.

We follow by investigating our attack detection mechanism, which calculates the KL divergence between a model and empirical peer ID distributions, and flags a CID as under attack if the divergence is above a certain threshold thr. In Figure 10, we plot the false positive and false negative rates of the detection method for different choices of the detection threshold (the lower the threshold, the more false positives, but the fewer false negatives). We see that increasing the number of Sybils makes the attack easier to detect, with fewer false negatives. Each DHT node can choose its own detection threshold, based on its desired false positive and false negative rates. However, it is reasonable for the default implementation to choose a threshold that favors fewer false negatives, thereby mitigating most attacks, at the cost of a small overhead of running the mitigation even when there is no attack. In the following experiments, we choose a threshold of 0.94 which achieves 4.4% false positives and 0.81% false negatives (circled in Figure 10).

In Figure 11, we show the results of the detection method for 250 experiments each with a different number of Sybil peers. Each point represents the result of a single experiment and the y-coordinate is the observed KL divergence value. The percentage of successful attacks (solid circles), where the

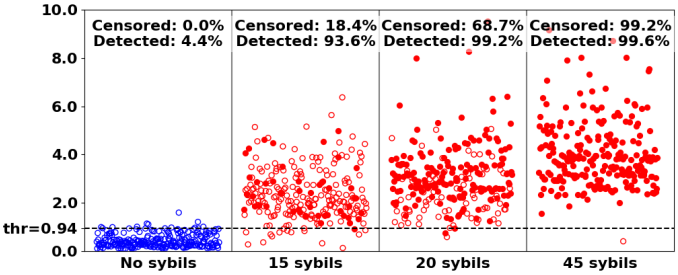

Fig. 11: KL divergence for varying numbers of Sybils e.

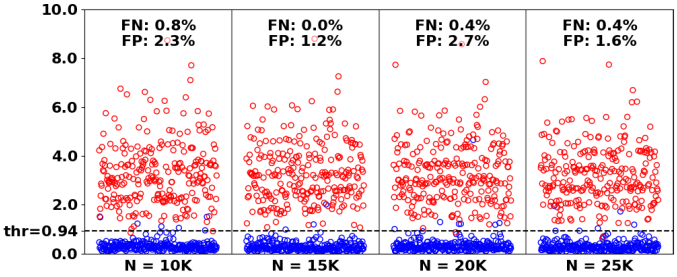

Fig. 12: KL divergence for varying network sizes N, and the false negative and false positive rates.

provider record was not found by the downloader, and the percentage of experiments that were flagged as attacks are indicated. If the number of Sybils launched by the attacker is decreased, more attempted attacks go undetected (false negatives), but we see that these attacks are not successful either. This effect is also summarized in Figure 7.

To run experiments for different network sizes N, we simulate a DHT network by generating N random peer IDs. Requests for a CID are resolved by simply finding the 20 closest peer IDs in this simulated network. Network size estimation and detection are performed using these simulated responses. In Figure 12, we show that the KL divergence metric is robust to changes in the network size. This is because our detection method automatically calculates the model distribution based on the estimated network size. Therefore, a fixed detection threshold can be used even as the network size changes. Note that the KL divergence values in such a simulation tend to be lower than those measured on the real network because lookups in the real network do not always result in the correct 20 closest peers.

E. Mitigation

We evaluate the performance and the overhead of using our mitigation mechanism in the live IPFS network. Mitigation of a censorship attack on cid is successful if its downloaders successfully retrieve at least one valid provider record using a FINDPROVIDERS(cid) operation. Our mitigation mechanism uses region-based queries with a region containing 20 peer IDs on average for both FINDPROVIDERS(cid) and PROVIDE(cid) when an attack is detected.

For different number of Sybils e, we launch censorship attacks on 50 different CIDs. During each attack, we launch e Sybil peers, and after waiting for one minute, we provide

the target CID from a separate DHT instance. We then test the reachability of the content from ten different downloaders. As a baseline comparison, we also provide results for $e = 0$ whenever appropriate.

Mitigation effectiveness. Figure 13 illustrates the mitigation effectiveness m_{eff} for different numbers of Sybils e, and provides results without the mitigation for comparison. Our mechanism mitigates all of the detected attacks for all the evaluated Sybil numbers. Importantly, for $e = 45$, the number of downloaders receiving their content increases from only 0.44% without the mitigation to 100% when it is activated.

To better understand the region-based queries, Figure 14 presents the average size of the following sets of peers: (i) *contacted:* peers encountered by the downloaders during FINDPROVIDERS(cid), (ii) *updated:* resolvers that obtain the provider record for cid, and (iii) *intersection:* non-Sybil peers in the intersection of the two aforementioned sets of peers. We omit unresponsive peers in both sets. The number of contacted peers increases with the increasing number of Sybils. Higher peer density in the target region causes additional lookups that reach honest peers located nearby. For the same reason, the number of updated peers increases as well.

The number of contacted peers is substantially higher than the number of updated peers, because the former includes not only just the peers within the target region but also peers encountered during the DHT walk toward that target region. More importantly, the average size of the non-Sybil intersection of contacted and updated peers oscillates around 30. This is higher than the expected value of 20 (*i.e.*, the region size used by the mitigation mechanism) because the current network size estimation slightly underestimates the size of the IPFS network. Higher intersection size ensures successful mitigation even when multiple honest nodes in the region are offline.

Mitigation overhead. In Figure 15, we assess the overhead of the mitigation mechanism in terms of the number of DHT lookups involved in region-based queries. While the number of lookups is up to 9 times higher than for an un-attacked network, the overhead increases sub-linearly with the number of Sybils, and is incurred only when an attack is detected.

We then measure the latency of the PROVIDE as well as FINDPROVIDERS operations. The average latencies of these operations are shown in Table II when under attack and otherwise, for both the default and the mitigation modes.

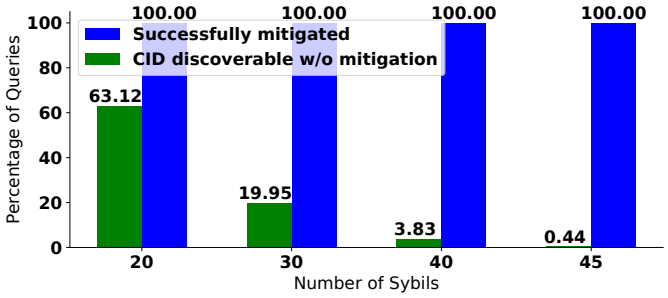

Fig. 13: Percentage of attacks that are mitigated.

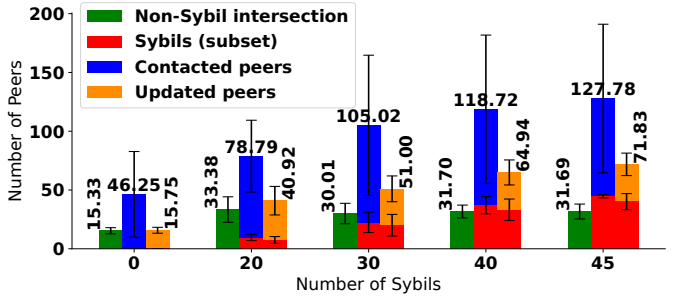

Fig. 14: Number of peers contacted and updated by region-based queries.

		Attack (45 Sybils)	No attack (0 Sybils)
Provide	Default	—	15914 ms
	Mitigation	24780 ms	27099 ms
Find 20 providers	Default	—	26483 ms
	Mitigation	28930 ms	27756 ms
Find 1 providers	Default	—	647 ms
	Mitigation	712 ms	329 ms

TABLE II: Average latency (milliseconds) of Provide and FindProviders operations with and without our mitigation, during attack and no attack. A red dash indicates that the operation was unsuccessful due to the attack.

In general, FINDPROVIDERS waits until it finds 20 distinct providers, or it has contacted all resolvers. This is followed in both the default and mitigation modes. However, the first provider record is obtained much earlier. Although finding one provider record may be enough in the optimistic case, the first provider record that the downloader obtains may not be correct (it may be old or may be sent by a malicious peer) and therefore the downloader might need to wait longer to retrieve the content. Even though the mitigation mode requires several DHT queries while the default mode uses only one, the latency of the mitigation mode is not much higher than the default mode. This is because the subsequent queries are much faster than the first query.

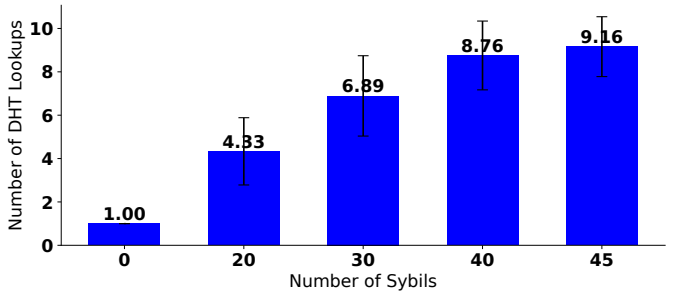

Fig. 15: The number of DHT lookups involved in a region-based query.

IX. Related Work

In this section, we review previous work focusing on solving the problem of attacks based on Sybil identities in decentralized systems and their limitations. For a more complete view of the Sybil attack and countermeasures, we refer the readers to a survey by Urdaneta *et al.* [67].

CFS [31] is a storage system built over the early Chord DHT [63]. CFS uses node ID authentication to prevent a node from taking a specific position in the DHT ring. CFS clients check that the node ID is the result of the hash of its IP address, plus a number from a small range, *e.g.*, 1 to 10. However, this solution is less effective when an attacker has access to a large number of IP addresses (*e.g.* using cloud providers) and it is incompatible with a large number of peers placed behind a NAT, which is the case in IPFS [66]. S-Chord [38] is an extension of Chord that can provide routing guarantees despite the presence of a number of Byzantine nodes in the network but it increases the number of messages and latency for routing by a factor logarithmic in the number of nodes. S/Kademlia [26] proposes Proof-of-Work (PoW) mechanisms to rate-limit the generation of new peer IDs. However, while PoW slows down the attacker, it does not fully mitigate the problem, makes the system less sustainable, and is problematic for constrained devices.

Some mechanisms make additional assumptions on trusted certificate authorities (CAs) to sign peer IDs [28] or use social trust networks [33], [48], [70], [71] to detect or prevent Sybil attacks. Even though most deployed systems do rely on hardcoded bootstrap nodes, relying on CAs to control and certify all memberships would be considered incompatible with the open and decentralized environment of IPFS.

Awerbuch and Scheideler [25] propose that the peer IDs of all honest peers in a DHT be rotated whenever a new peer joins. This can prevent an attacker's peers from concentrating in one region of the key space. Unfortunately, this solution is particularly expensive in a dynamic network where nodes constantly join and leave the system.

Cholez *et al.* [30] introduce a Sybil detection mechanism based on KL-divergence followed by removing peers in densely populated regions from the routing tables. We adopt their detection mechanism but do not remove any peers from routing tables. The removal mechanism could be manipulated by the attacker to progressively remove all honest peers from the network, which further exacerbates the attack.

Recently, Protocol Labs introduced network indexers [11] allowing to resolve a CID to a list of providers in Filecoin [39]. While the usage of cloud infrastructure makes the system highly efficient and resistant to Sybil attacks, the indexer is fully centralized introducing the risks of censorship and can constitute a single point of failure.

Eclipse attacks. Multiple attacks based on node *eclipsing* target decentralized systems. An attacker attempts to control all the neighbors of a specific target node in the overlay. This differs from the content censorship attack we discuss in this paper, which targets a specific entry of a distributed directory.

Eclipse attacks are documented for Bitcoin [40], [60] or Ethereum [42], [51] allowing to *partition* the blockchain network and prevent a miner from participating fairly. The recent Gethlighting attack shows this is possible by only eclipsing a subset of a node's neighborhood [43]. A recent attack targets the IPFS DHT [41], [56] and allows isolating a single node from the network. As a result, the IPFS DHT was augmented with table eviction policies and with rules restricting the number of peers with the same IP address in a routing table. This makes the attack impractical even for a resourceful attacker [56]. Our censorship attack targets content rather than single nodes and works despite these changes. However, we also build upon this past work as we rely on eclipse resistance for our mitigation techniques.

Wang *et al.* [68] is an early example of a content censorship attack, targeting the Kad network, an implementation of Kademlia used in the eDonkey [8] and eMule [9] content-sharing networks. These attacks exploit a vulnerability in the Kad implementation: peers were not authenticated based on their peer IDs, so an attacker could impersonate another peer. This vulnerability does not exist in the IPFS network where peer IDs are derived by hashing the peer's public key, and where messages are signed with the corresponding secret keys. Our attack is much simpler than the one of Wang *et al.* and does not require this vulnerability.

X. Discussion and Future Work

Currently, IPFS does not provide an admission mechanism and resolvers will accept any provider records until they run out of storage. After that, depending on the implementation, the resolvers may crash or flush older, legitimate provider records. The time required for this attack depends on the bandwidth available at each resolver and the amount of free storage. While a deeper analysis is out of the scope of this paper, introducing an admission mechanism based on the diversity of incoming traffic has the potential to eliminate this vulnerability.

While our mitigation technique (Section VII) fully protects against the CID censorship attack, it involves querying the Sybil nodes for provider records. The Sybil nodes may return a large number of fake provider records so that downloaders keep trying them, thereby slowing down the resolution. The impact of such an attack can be reduced if the downloader only tries a single provider record obtained from each resolver and prefers records obtained from resolvers with diverse IP addresses (*i.e.* from different /24 networks). Any attempts to significantly delay the resolution would sharply increase the attacker's cost.

Our mitigation technique relies on region-based DHT queries. For easy integration with the current IPFS network, those queries are built on top of a regular Kademlia DHT that does not natively support them. This results in slightly higher overhead and increases resolution time. Adapting the core internals of the DHT and optimizing them for region-based queries might speed up the process and reduce its overhead. However, such deep changes make incremental deployment and compatibility with the existing version challenging.

During this project, we initially considered an approach where providers register their provider records on all the nodes encountered on the path towards resolvers. Such a solution

increases the chance of an honest downloader receiving a correct provider record before reaching the region with the Sybil nodes. However, the mechanism does not provide resistance for downloaders located close to the CID in the DHT hash space. Furthermore, the on-path registration significantly increases the storage cost of holding provider records for the entire network even when no attack is being conducted.

The IPFS DHT, and thus our mitigation and detection mechanisms, depends on the correctness of the DHT routing. However, a powerful attack may try to disturb DHT operations by deploying a large number of uniformly distributed Sybils that only return other Sybils when queried. While costly, such an attack could be devastating for the entire ecosystem. We advocate for additional future work that improves the DHT resistance to such attacks and is practical to deploy in large-scale networks.

XI. Conclusion

We presented a successful censorship attack on IPFS. We showed that an attacker can easily make any content undiscoverable in the network by strategically placing a small number of Sybil identities in the DHT. The effectiveness of the attack was confirmed by removing multiple, specifically crafted content from the live IPFS network. Importantly, our attack has a constant, negligible cost regardless of the popularity of the target content.

The attack has a significant impact on the IPFS network itself as it threatens the core functionality of the platform. However, it also impacts other systems that rely on the availability of content stored on IPFS. This includes thousands of decentralized applications and oracles deployed on various blockchains. Moreover, the DHT flaw that led to the attack is present in other currently deployed DHT-based systems.

We also presented a robust detection technique allowing us to detect the attack in real time without communication overhead and to activate our proposed mitigation mechanisms when necessary.

Finally, we introduce a practical mitigation technique based on region-based DHT queries. While many others mitigation techniques have been proposed, none of them are practical enough to be deployed in an open decentralized system. Our approach is the first that can be deployed incrementally in a live network without requiring changes to the core DHT protocol. It also does not require additional components and does not incur significant overhead. Importantly, our mitigation technique prevents the attack without blocking any nodes or using unreliable reputation systems. We believe that our mitigation technique can be easily integrated into other DHT-based systems.

Acknowledgements

This work was partly done when Srivatsan Sridhar was consulting for Protocol Labs. At Stanford University, Srivatsan Sridhar's research is funded by a gift from the Ethereum Foundation.

References

[1] Almonit. [Online]. Available: https://almonit.eth.link/#/

[2] Audius. [Online]. Available: https://audius.org

[3] Berty. [Online]. Available: https://berty.tech

[4] The BitTorrent specification. [Online]. Available: https://www.bittorrent.org/beps/bep_0052.html

[5] dClimate. [Online]. Available: https://www.dclimate.net

[6] Deece. [Online]. Available: https://github.com/navinkeizer/Deece

[7] DTube. [Online]. Available: https://d.tube

[8] eDonkey network. [Online]. Available: http://www.edonkey2000.com

[9] eMule network. [Online]. Available: http://www.emule-project.net

[10] Ethlance. [Online]. Available: https://github.com/district0x/ethlance

[11] Introducing the network indexer. [Online]. Available: https://filecoin.io/blog/posts/introducing-the-network-indexer/

[12] IPFS Camp 2022. [Online]. Available: https://2022.ipfs.camp

[13] IPFS ecosystem directory. [Online]. Available: https://ecosystem.ipfs.tech/

[14] libp2p/go-libp2p-kad-dht: A Kademlia DHT implementation on go-libp2p. [Online]. Available: https://github.com/libp2p/go-libp2p-kad-dht

[15] libp2p/go-libp2p-kad-dht: Get closest peers implementation. [Online]. Available: https://github.com/libp2p/go-libp2p-kad-dht/blob/2b85cfc0b1e5372bb36cb3ed4c82321054b1f055/lookup.go#L19

[16] Matters News. [Online]. Available: https://matters.news

[17] Optimistic provide by dennis-tra, pull request 783, libp2p/go-libp2p-kad-dht. [Online]. Available: https://github.com/libp2p/go-libp2p-kad-dht/pull/783

[18] Peergos. [Online]. Available: https://peergos.org

[19] Space. [Online]. Available: https://github.com/ipfs-shipyard/space

[20] Splinterlands. [Online]. Available: https://splinterlands.com/

[21] Temporal. [Online]. Available: https://temporal.cloud

[22] (2019) Discovery overview. The Ethereum Foundation. [Online]. Available: https://github.com/ethereum/devp2p/wiki/Discovery-Overview

[23] (2023) Gala games. [Online]. Available: https://app.gala.games/games

[24] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM computing surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, 2004.

[25] B. Awerbuch and C. Scheideler, "Towards a scalable and robust DHT," *Theory Comput. Syst.*, vol. 45, no. 2, pp. 234–260, 2009.

[26] I. Baumgart and S. Mies, "S/Kademlia: A practicable approach towards secure key-based routing," in *International conference on parallel and distributed systems*, ser. ICPADS. IEEE, 2007.

[27] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of Polkadot and its design considerations," *arXiv preprint arXiv:2005.13456*, 2020.

[28] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *Fifth Symposium on Operating Systems Design and Implementation*, ser. OSDI. USENIX Association, 2002.

[29] Celestia, "The first modular blockchain network," https://celestia.org, 2021.

[30] T. Cholez, I. Chrisment, and O. Festor, "Efficient DHT attack mitigation through peers' ID distribution," in *Workshops of the International Symposium on Parallel & Distributed Processing*, ser. IPDPSW. IEEE, 2010.

[31] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *18th ACM Symposium on Operating Systems Principles*, ser. SOSP, 2001.

[32] G. Dán and N. Carlsson, "Centralized and distributed protocols for tracker-based dynamic swarm management," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 297–310, 2012.

[33] G. Danezis, C. Lesniewski-Laas, M. F. Kaashoek, and R. Anderson, "Sybil-resistant DHT routing," in *10th European Symposium on Research in Computer Security*, ser. ESORICS. Springer, 2005.

[34] G. Danezis and P. Mittal, "SybilInfer: Detecting Sybil nodes using social networks." in *Network and Distributed System Security Symposium*, ser. NDSS, 2009.

[35] "Dat ecosystem," Dat Consortium, 2023. [Online]. Available: https://dat-ecosystem.org/

[36] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with Bitswap: A multi-path file transfer protocol in IPFS and Filecoin," Protocol Labs, Tech. Rep., 2021.

[37] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *12th Annual International Cryptology Conference*, ser. CRYPTO. Springer, 1993.

[38] A. Fiat, J. Saia, and M. Young, "Making chord robust to byzantine attacks," in *13th Annual European Symposium on Algorithms*, ser. ESA. Springer, 2005.

[39] B. Fisch, J. Bonneau, N. Greco, and J. Benet, "Scaling proof-of-replication for Filecoin mining," *Report. Protocol Labs Research*, 2018.

[40] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *24th USENIX Security Symposium*, 2015.

[41] S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Mapping the Interplanetary filesystem," in *IFIP Networking Conference*, 2020.

[42] S. Henningsen, D. Teunis, M. Florian, and B. Scheuermann, "Eclipsing Ethereum peers with false friends," *arXiv preprint arXiv:1908.10141*, 2019.

[43] H. Heo, S. Woo, T. Yoon, M. S. Kang, and S. Shin, "Partitioning Ethereum without eclipsing it," in *Network and Distributed System Security Symposium*, ser. NDSS, 2023.

[44] M. S. Hossan, M. L. Khatun, S. Rahman, S. Reno, and M. Ahmed, "Securing ride-sharing service using IPFS and hyperledger based on private blockchain," in *2021 24th international conference on computer and information technology*, ser. ICCIT. IEEE, 2021.

[45] H.-S. Huang, T.-S. Chang, and J.-Y. Wu, "A secure file sharing system based on IPFS and blockchain," in *2nd International Electronics Communication Conference*, ser. IECC, 2020.

[46] S. Jianjun, L. Ming, and M. Jingang, "Research and application of data sharing platform integrating Ethereum and IPFS technology," in *19th International Symposium on Distributed Computing and Applications for Business Engineering and Science*, ser. DCABES. IEEE, 2020.

[47] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized schemes for size estimation in large and dynamic groups," in *4th IEEE International Symposium on Network Computing and Applications*, ser. NCA. IEEE, 2005.

[48] C. Lesniewski-Laas and M. F. Kaashoek, "Whanau: A sybil-proof distributed hash table," in *7th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI. USENIX Association, 2010.

[49] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[50] G. S. Manku, M. Bawa, P. Raghavan *et al.*, "Symphony: Distributed hashing in a small world." in *USENIX Symposium on Internet Technologies and Systems*, ser. USITS, 2003.

[51] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on Ethereum's peer-to-peer network," *Cryptology ePrint Archive*, 2018.

[52] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *International Workshop on Peer-to-Peer Systems*, ser. IPTPS. Springer, 2002.

[53] J. McDonald, *Handbook of Biological Statistics*, 3rd ed. Sparky House Publishing, 2014.

[54] moxystudio. Discussify. [Online]. Available: https://github.com/ipfs-shipyard/pm-discussify

[55] H. Mukne, P. Pai, S. Raut, and D. Ambawade, "Land record management using Hyperledger Fabric and IPFS," in *10th International Conference on Computing, Communication and Networking Technologies*, ser. ICCCNT. IEEE, 2019.

[56] B. Prünster, A. Marsalek, and T. Zefferer, "Total eclipse of the heart–disrupting the InterPlanetary file system," in *31st USENIX Security Symposium*, 2022.

[57] B. Prünster, D. Ziegler, C. Kollmann, and B. Suzic, "A holistic approach towards peer-to-peer security and why proof of work won't do," in *14th International Conference on Security and Privacy in Communication Networks*, ser. SecureComm. Springer, 2018.

[58] Y. Psaras and D. Dias, "The interplanetary file system and the Filecoin network," in *50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume*, ser. DSN-S. IEEE, 2020.

[59] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *1st international conference on peer-to-peer computing*, ser. P2P. IEEE, 2001.

[60] M. Saad, S. Chen, and D. Mohaisen, "SyncAttack: double-spending in Bitcoin without mining power," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2021.

[61] E. Sohl. A new method for estimating P2P network size. [Online]. Available: https://eli.sohl.com/2020/06/05/dht-size-estimation.html

[62] R. R. Sokal and F. J. Rohlf, *Biometry: the principles and practice of statistics in biological research*, 3rd ed. Freeman, 1994.

[63] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on networking*, vol. 11, no. 1, pp. 17–32, 2003.

[64] J. P. Timpanaro, T. Cholez, I. Chrisment, and O. Festor, "Evaluation of the anonymous I2P network's design choices against performance and security," in *2015 International Conference on Information Systems Security and Privacy*, ser. ICISSP. IEEE, 2015.

[65] D. Trautwein. Network size estimation. [Online]. Available: https://www.notion.so/Network-Size-Estimation-4ab2c52083ed4e88968f629d1fa47eb7

[66] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of IPFS: a storage layer for the decentralized web," in *ACM SIGCOMM 2022 Conference*, 2022.

[67] G. Urdaneta, G. Pierre, and M. van Steen, "A survey of DHT security techniques," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 8:1–8:49, 2011.

[68] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. F. Kune, N. Hopper, and Y. Kim, "Attacking the Kad network," in *4th international conference on Security and privacy in communication networks*, ser. Securecomm, 2008.

[69] Q. Xu, Z. Song, R. S. M. Goh, and Y. Li, "Building an Ethereum and IPFS-based decentralized social network system," in *24th international conference on parallel and distributed systems*, ser. ICPADS. IEEE, 2018.

[70] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "SybilLimit: A near-optimal social network defense against sybil attacks," *IEEE/ACM Trans. Netw.*, vol. 18, no. 3, pp. 885–898, 2010.

[71] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman, "SybilGuard: defending against Sybil attacks via social networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 576–589, 2008.

[72] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[73] ZorrillosDev. Watchit. [Online]. Available: https://watchit.movie/#/

APPENDIX A
ARTIFACT APPENDIX

In this work, we implement a censorship attack on the IPFS network, a method to detect the attack, and a method to mitigate the attack. Our artifact includes the implementations of these three components and experiments to measure their effectiveness, accuracy, and cost.

Artifact Outline (key aspects):

```
/
├── README.md
├── common
│   ├── go-libp2p-kad-dht
│   │   (detection & mitigation)
│   ├── go-libp2p-kad-dht-Sybil
│   │   (Sybil DHT implementation)
│   ├── Sybil_DHT_Nodes (Sybil node implementation)
│   └── kubo (standard IPFS
│           implementation)
├── experimentCombined (main experiment & results)
│   └── README.md
└── python (plots & other simulations)
    └── README.md
```

A. Description & Requirements

1) How to access: The artifact is available online at the link https://doi.org/10.5281/zenodo.8300034. The detection and mitigation parts are also available on Github: https://github.com/ssrivatsan97/go-libp2p-kad-dht, and are scheduled to be deployed in the official release of `go-libp2p-kad-dht`. Our mitigation will be deployed on a per-client basis (not network-wide) and therefore, the experiments in this artifact are expected to remain reproducible as they attack our own providers and downloaders that do not use the mitigation.

2) Hardware dependencies: Our experiments require a machine with a public IP address which must allow incoming TCP connections on several ports to allow other IPFS peers to connect to our Sybil peers. Our experiments were run on a machine rented from AWS. The recommended instance type is `t3.xlarge` which has a 2nd generation Intel Xeon Scalable Processor (3.1 GHz) with 4 vCPUs, 16 GB memory, and 5 Gbps peak bandwidth. While the compute and memory requirements of our experiments are modest, a high peak bandwidth is required to ensure good connectivity of the Sybils and good attack effectiveness. Our artifact includes instructions in the README on how to run the experiments on any other machine with these requirements.

3) Software dependencies: Recommended operating system: Ubuntu 22.04. Required software: Go v1.19.10, Python 3.10, gcc, Make.

4) Benchmarks: None.

B. Artifact Installation & Configuration

For the artifact evaluation, the setup begins with logging in to our provided AWS machine via SSH using the instructions given in the README file. Detailed instructions for each of the following steps are given in the top-level README file.

Network setup: In firewall settings, incoming TCP connections on ports 4001, 5001, 63800-63850 must be enabled.

Install software requirements: Install Go, Python, gcc, and make. We provide a file requirements.txt to help install all required Python packages.

Build and initialize IPFS: We provide the source code of kubo, the IPFS client written in Go, in our artifact. After compiling the source code, an IPFS node must be initialized. The IPFS node runs in the background during our experiments and is used to obtain information from the network such as the closest peers to the target CID.

C. Experiment Workflow

Our main experiment (E1) has two phases: data collection and plotting.

Data collection: In this phase, we run the attack for different numbers of Sybils, different target CIDs, and different downloader clients. We collect measurements regarding the attack's success, detection results, and mitigation success each time. The code for this step along with instructions is given in the folder `experimentCombined/`.

Data processing and plotting: In this phase, we process the collected measurements and generate the plots in the paper (Figures 7, 10, 11 and 13 to 15). The code and instructions for this step are in the folder `python/`.

D. Major Claims

- (C1): Our censorship attack using $e = 45$ Sybil nodes blocks 99% of users' content requests. This is illustrated in Figure 7.
- (C2): Our detection mechanism achieves a false negative rate of 0.81% and a false positive rate of 4.4% for our chosen detection threshold (Figure 10). Most attacks that were not detected were also not successful in censoring the content, and the detection rate improves as the attack effectiveness increases (Figure 11).
- (C3): Our mitigation leads to successful content discovery 100% of the time (Figure 13) by sending provider records to a constant number of honest peers even as the number of Sybils increases (Figure 15). Moreover, the overhead of our mitigation increases sub-linearly with the number of Sybils (Figure 14). Claims (C1), (C2), and (C3) are proven by experiment (E1).
- (C4): Our censorship attack incurs a low cost for the attacker. The required keys can be generated on commodity hardware within 20 seconds (for EdDSA) or 2 hours (for RSA). This is proven by the experiment (E2) whose results are illustrated in Figure 9.

E. Evaluation

1) Experiment (E1): [Detection and Mitigation] [4 compute-hours]: In a single run of this experiment, we do the following: We create a new file, compute its CID, generate Sybil identifiers based on the CID, and launch Sybil nodes. Then, we launch two DHT nodes, a provider and a downloader. The provider *provides* the file, the downloader attempts to find providers for the file and runs the detection. We record the success of the attack and the detection result. Then, the provider provides the file with the mitigation enabled, the downloader attempts to find providers also with the mitigation

enabled, and we record the mitigation success, the number of DHT lookups performed by the mitigation, and the number of honest and Sybil peers contacted and successfully updated with provider records. This whole process is repeated for 10 different files. This experiment is then repeated for 0, 20, 30, 40, and 45 Sybils. Each run takes 4-5 minutes on average. Hence, we scale down the number of runs for each experiment from 100 to 10 and run it for fewer values of the number of Sybils so that the experiment completes within 4 hours.

[Preparation] None beyond Appendix A-B.

[Execution] Change the working directory to `experimentCombined/`, then build and run the experiment as per instructions in `experimentCombined/README.md`. Running the experiment stores the results in the specified output directory.

[Results] The folder `python/` contains code to generate Figures 7, 10, 11 and 13 to 15 from the experiment results. For reference, we also provide the results corresponding to the plots in the paper in `experimentCombined/detection_results` and `experimentCombined/mitigation_results`.

2) Experiment (E2): [Keys Generation Time] [10 compute-hours]: This experiment generates the required Sybil keys for different sizes of the network and analyzes the generation time.

[Execution] Run two sets of experiments measuring generation time for RSA and EDDSA:

```
cd python/
go run measureGenerateSybilKeys rsa > timing_rsa.csv
go run measureGenerateSybilKeys eddsa > timing_eddsa.csv
```

We provide our results files for reference (generating RSA keys takes a while).

```
./simulation_results/sample_rsa.csv
./simulation_results/sample_eddsa.csv
```

[Results] The commands below will reproduce the graph in Figure 9.

```
python3 plot_key_generation_time.py
```

F. Customization

Experiment (E1) provides arguments to customize the number of Sybils, the number of clients, and the number of CIDs for which to run the experiment.

G. Notes

The key generation time is heavily hardware dependent. The result might thus differ from the ones shown in Figure 9. The experiment time might be shortened by adjusting the number of tries per network size. However, this comes at the price of losing the precision.

In the artifact, we include the code and instructions for generating all other figures in the paper as well.