Mapping the Interplanetary Filesystem

Sebastian Henningsen Martin Florian Sebastian Rust Björn Scheuermann

Weizenbaum-Institute for the Networked Society
/ Humboldt-Universität zu Berlin
Berlin, Germany

***Abstract*—The Interplanetary Filesystem (IPFS) is a distributed data storage service frequently used by blockchain applications and for sharing content in a censorship-resistant manner. Data is hosted by an open set of peers, pointers to which are distributed using a Kademlia-based distributed hash table (DHT). In this paper, we study the structure of the resulting overlay network, as it significantly influences the robustness and performance of IPFS. We monitor and systematically crawl IPFS' DHT towards mapping the IPFS overlay network. Our measurements found an average of 44474 nodes at every given time. At least 52.19 % of these reside behind a NAT and are not reachable from the outside, suggesting that a large share of the network is operated by private individuals on an as-needed basis. Based on our measurements and our analysis of the IPFS code, we observe that IPFS bears similarities to unstructured overlay networks in terms of content retrieval mechanics and overlay topology. While such a structure has benefits for robustness and the resistance against Sybil attacks, it leaves room for improvement in terms of performance and query privacy.**

I. Introduction

The Interplanetary Filesystem [1] is a community-developed peer-to-peer protocol and network providing public data storage services. IPFS is often cited as a fitting data storage solution for blockchain-based applications [2]–[6] and was previously used for mirroring censorship-threatened websites such as Wikipedia[1]. IPFS' design is reminiscent to classical peer-to-peer systems such as filesharing networks [7]. Any Internet-enabled device can participate as an IPFS node and nodes are operated without explicit economic incentives. Unlike blockchain-based systems such as Bitcoin [8], data items (files, folders, ...) are not replicated globally. Instead, each data item is stored by a small set of nodes, that item's *providers*, who make the data item available to other peers. Data items are addressed through immutable, cryptographically-generated names which are resolved to their providers through a distributed hash table based on Kademlia [9].

Given IPFS' reported attractiveness as a building block for decentralized applications and censorship circumvention, the question arises whether the network is actually suited to fulfill this role in terms of robustness and "decentrality". We are particularity interested in the following questions:

- What is possible with regards to mapping and monitoring the IPFS overlay network (assessing its "health")?
- How many and what types of nodes participate in the IPFS network? What kind of churn do they exhibit?
- How "decentralized" is the network – in terms of overlay structure and the geographical distribution of nodes?

In this paper, we present the results of a comprehensive empirical study on the IPFS overlay network. As central components of our methodology, we collected connection data from several self-controlled monitoring nodes and repeatedly crawled the IPFS DHT. We find that, similar to other Kademlia-based systems [7], [10], connections corresponding to DHT routing table entries can be learned through carefully crafted, iterative peer discovery queries. However, through measurements and studies of the IPFS code base, we also uncover a number of differences to both previous systems and the design outlined in the IPFS whitepaper [1]. For example, in contrast to other Kademlia implementations, IPFS establishes a connection with every peer it encounters and maintains a large number of connections that do not correspond to any DHT routing table entries. As a further surprise, and despite whitepaper claims to implementing mechanisms from [11], we find that no noteworthy protection against Sybil attacks is currently implemented in IPFS. The effect on security, however, is limited. Thanks to IPFS' unusual hoarding of overlay connections and the fact that requests are promiscuously sent to all direct peers, content retrieval is possible even if an attacker can place Sybil nodes at key locations in the overlay.

Our contributions are threefold:

- We give an overview on the IPFS system based on white papers, public online discussions, and code. Notably, we describe the actual state of current implementations (go-ipfs v0.4.22 and related packages), and contrast it with the design documents.
- We run monitoring nodes with varying connectivity (fully reachable, behind NAT), some of which modified to accept an unlimited number of overlay connections. Among other things, this allows us to map the quantitative relationship between overlay connections and DHT routing table entries, as influenced by node connectivity.
- We repeatedly crawled the IPFS DHT to obtain its topology, thereby also enumerating all DHT-enabled nodes and their addresses.

II. Related Work

Peer-to-Peer networks have been studied extensively in the past, yielding various results on network crawling and char-

[1] https://github.com/ipfs/distributed-wikipedia-mirror

ISBN 978-3-903176-28-7

acterization [12]–[20] with applications to real-world peer-to-peer systems like BitTorrent [7] and KAD [16]. We extend this line of research by developing and performing a measurement study on IPFS – a highly popular data storage network with various reported applications (see, e.g., [2], [4], [5] for a sample of academic projects).

In their seminal work, Stutzbach *et al.* [15], [20] study requirements and pitfalls with regards to obtaining accurate snapshots of peer-to-peer overlays. Specifically, they find that the duration of crawls should be as small as possible, to avoid distortions in the results due to churn. Steiner *et al.* [16], [19] crawled the KAD network to obtain the number of peers and their geographical distribution as well as inter-session times. Salah *et al.* [10] studied the graph-theoretical properties of KAD and contrasted their results with analytical considerations. Similarly to KAD and other networks, the DHT component of IPFS is also, in principle, based on Kademlia [9]. We therefore build upon previously proposed crawling methods. However, we also find that IPFS differs substantially from more canonical Kademlia implementations, necessitating enhancements to existing measurement approaches. We furthermore explore links in the IPFS overlay that are not used for DHT functionality.

A simple crawler for the IPFS DHT has been made available before[2] that aims at enumerating all nodes in the network. For this paper, we developed a new crawler from scratch to capture the entire overlay topology[3]. It is optimized for short running times to ensure the correctness of snapshots.

The I/O performance of retrieving and storing data on IPFS was studied in [6], [21]–[23], with interesting and partially contrasting results. Ascigil *et al.* [6] report high latencies, low throughput and high redundancy when retrieving data through IPFS. Similarly, Shen *et al.* [21] report high latencies for large files and large variances in the transmission speed. In [22], [23], the authors optimize and evaluate the performance of IPFS in edge computing settings. They report small latencies and high throughput when using the global DHT as little as possible and running IPFS in a private local network. In contrast to these prior works, we focus on grasping the overlay structure and node composition of the public IPFS network. Furthermore, we give a comprehensive, code review-supported overview of IPFS' "network layer", revealing information not previously available in literature or documentation.

III. The Interplanetary Filesystem

In the following, we describe key aspects of IPFS' design and discuss particularities relevant to conducting a measurement study and interpreting its results. Most of IPFS' networking functionality is handled by the reusable networking library *libp2p*[4] (originally part of the IPFS project). IPFS is still the main use case of libp2p. To ease understanding, we will therefore refer to both IPFS and libp2p as simply IPFS. It is worth noting that the development of IPFS and libp2p is ongoing, so that details of the design may change over time. Here, we focus on inherent conceptual properties that change rarely in deployed protocols.

[2]https://github.com/vyzo/ipfs-crawl

[3]https://github.com/wiberlin/ipfs-crawler

[4]https://libp2p.io/

A. In a Nutshell

As a broad overview, the design of IPFS can be summarized in the following way:

- Data items are stored and served by data providers.
- References to data providers are stored in a DHT.
- In *addition* to DHT-based provider discovery, data items are requested from *all* connected overlay neighbors.
- DHT: Kademlia over TCP (and other reliable transports like QUIC), $k = 20$, no eviction mechanism and replacement buffers.
- In contrast to information in the IPFS white paper [1], no proposals of S/Kademlia [11] are implemented.
- Overlay connections can correspond to DHT routing table (bucket) entries, but do not have to.
- By crawling the DHT we obtain a subset of all connections; we estimate the size of that subset in Sec. V-B.

B. Node Identities and S/Kademlia

Anyone can join the IPFS overlay network, i. e., it is an open (permissionless) system with weak identities. Nodes are identified by the hash of their public key, $H(k_{pub})$. To ensure flexibility, IPFS uses so-called "multi-hashes": a container format capable of supporting different hash functions. A multi-hash adds meta information about the hash function and the digest length to a hash. By default, IPFS uses RSA2048 key pairs and SHA256 hashes.

Creating a new IPFS identity is as simple as generating a new RSA key pair – making the network highly susceptible to Sybil attacks [24]. Towards increasing the cost of Sybil attacks, the IPFS white paper [1] suggests that the Proof-of-Work-based ID generation approach of S/Kademlia [11] is in use. However, based on our careful review of the IPFS codebase, this is *not* the case at the moment. IPFS currently implements no restriction on the generation of node IDs, neither are DHT lookups in IPFS' Kademlia implementation carried out through multiple disjoint paths, as proposed in S/Kademlia. IP address-based Sybil protection measures, such as limiting the number of connections to nodes from the same /24 subnet, are currently also *not* in use.

On the network layer, IPFS uses a concept of so-called "multi-addresses". Similar to multi-hashes, these multi-addresses are capable of encoding a multitude of network and transport layer protocols. Through these multi-addresses, a peer announces its connection capabilities, e. g., IPv4, IPv6, etc., and the addresses it can be reached from to the network.

C. Data Storage: Self-Certifying Filesystem

IPFS uses a form of Self-Certifying Filesystem (SFS) (originally introduced in [25]) to ensure the integrity of data throughout its delivery. In IPFS, each data item d is assigned a unique immutable address that is the hash of its content, i. e. $\mathrm{addr}(d) = H(d)$. Technically, folders and files are organized

in a Merkle tree-like structure. For example, a folder entry contains the hashes of all files in the folder. In the end, the receiver can recognize whether received data was tampered with by comparing its hash with the requested address.

D. Content retrieval

Data items are usually stored at multiple nodes in the network. Nodes store content because they are its original author or because they have recently retrieved it themselves. Nodes normally *serve* the data items they store, upon request. The nodes that store a given data item are consequently referred to as that data item's *providers*.

When an IPFS node v wishes to retrieve data item d (e. g., based on a user request), it follows two strategies *in parallel*:

1) Look up providers $P(d)$ for d in the DHT (cf. Sec. III-E), then request d from members of $P(d)$.
2) Ask *all* nodes it is currently connected to for d, using the *Bitswap* subprotocol (cf. Sec. III-F)

E. Content Retrieval: Kademlia DHT

IPFS uses a Kademlia-based DHT. Kademlia offers mechanisms for efficiently locating data records in a peer-to-peer network [9]. Each node in the IPFS overlay network is uniquely identified by its node ID—the SHA-2 hash of its public key. Data records are indexed and located using *keys*. Records in the IPFS DHT are lists of data providers for data items stored in IPFS, and the keys to these records are consequently the addresses of data items ($H(d)$ for a data item d). An example scenario illustrating this relationship is presented in Fig. 1. Nodes can be part of the IPFS overlay

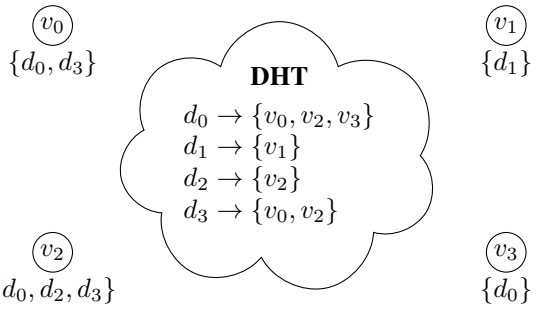

Figure 1: Interplay between data items, data providers and the DHT: for each data item d_i, the DHT stores a pointer to peers which possess d_i.

network but choose to not participate in the DHT, by acting as so-called *client nodes*. Therefore, we distinguish between the *IPFS overlay* which entails all nodes, including clients, and the *overlay without clients* which consists of nodes that take part in the DHT protocol. When no ambiguity may occur, we simply use the term overlay for both.

Node IDs and keys share the same representation and are treated identically as *IDs* ($\in [0, 2^{256}]$). Records in a Kademlia DHT are stored at nodes whose node ID is "close" to the record's key. Closeness is defined via the bitwise XOR of two IDs, taken as an integer value, i. e., $\mathrm{dist}(x, y) = x \oplus y$. A node stores its known neighbors in so-called k-*buckets* which partition the known overlay network (without clients) based on the local node's ID. Every k-bucket (or simply bucket) stores up to k neighbors. For each possible distance n there is exactly one bucket storing nodes with $\mathrm{dist}(x, y) = n$, where x is the local node's ID and y is any node ID from the respective bucket. In IPFS' DHT, $k = 20$, i. e., each bucket stores up to 20 entries. Bucket i stores nodes whose distance is in $[2^i, 2^{i+1})$, which effectively corresponds to the length of the common prefix shared between two node IDs.

In IPFS, a node v resolves an ID x to a record or a set of closest nodes on the overlay using multi-round iterative *lookups*. In each lookup round, v sends requests to known nodes with IDs close to x. Nodes reply with either a record corresponding to x or with new node contacts, selected by proximity to x and therefore enabling a subsequent lookup round. Peer-to-peer connections in IPFS are always based on TCP or other reliable transport layer protocols. Consequently, the DHT component of IPFS also uses (TCP-) connections (unlike many other Kademlia implementations, which are datagram based).

In contrast to the original Kademlia proposal [9], IPFS does not ping the last entry of a bucket for a potential eviction, if a new node is about to enter a new bucket. Instead, the new node is simply rejected. Nodes are ejected from buckets only if the connection with them is terminated for other reasons. Similarly, IPFS' DHT does not maintain a replacement buffer [9] with candidate nodes for full buckets.

F. Content Retrieval: Bitswap Data Exchange

Bitswap is a subprotocol of IPFS, used for transfers of actual content data. It is similar to Bittorrent [7] and is used for obtaining data items from connected peers. Interestingly, IPFS overlay nodes (including clients) query *all* of their connected peers for data items they look for (their so called *wantlist*). This happens in parallel to DHT queries for data providers registered in the DHT. While in some aspects questionable with regards to performance, the promiscuous broadcasting of queries results in a key security feature of IPFS: attacks on the DHT have only a limited impact on the availability of stored data. For example, while it is within the means of a dedicated attacker to overwrite the providers record for a file with Sybil nodes, users are able to retrieve it via one of their many (around 900, cf. Sec. VII) directly connected peers.

We verified this behavior experimentally within a private IPFS network. We overwrote the provider lists for a target data item with Sybil entries, effectively removing all correct providers from the DHT. Whereas, as expected, all DHT queries for the data item failed after this intervention, nodes were still able to obtain the data item from overlay neighbors.

We conclude that IPFS enhances DHT lookups with the (one-hop) flooding of requests to all connected neighbors - a technique more natural to unstructured overlay networks.

IV. Understanding the Overlay Structure

IPFS nodes nodes can be separated into two kinds: nodes that participate in the DHT and nodes that are *clients*. We want to disentangle the different kinds of overlays that arise through

this distinction and reason about what information can be measured. We distinguish between the IPFS overlay ($\tilde{G}$), the IPFS overlay without clients (G) and the overlay induced by DHT buckets (G'). We explain the differences in the following.

Overlay networks are commonly modeled as graphs with nodes as vertices and connections between those nodes as edges. Let $\tilde{G} = (\tilde{V}, \tilde{E})$ be an undirected graph (due to the symmetry of TCP-connections), with $\tilde{V}$ representing the nodes in the IPFS overlay and $\tilde{E}$ the connections among them. $\tilde{G}$ consists of *all* nodes, including clients. Since clients do not contribute to the DHT – one of the core pillars of IPFS – we focus most of our analysis on the IPFS overlay without clients. Let $V \subseteq \tilde{V}$ be the set of IPFS nodes that are not clients and $G := \tilde{G}[V]$ the induced subgraph of V, i. e., the graph with vertex set V and all edges from $\tilde{E}$ that have both endpoints in V. V can be, to a large extent, learned by crawling the DHT buckets of each node, whereas it is not straightforward to reason about $\tilde{V}$ as client nodes are not registered anywhere or announced on the network.

What exactly can be learned by crawling each node's buckets? IPFS attempts to reflect every new connection (inbound or outbound) to DHT-enabled nodes (i. e., nodes that are not pure clients) in DHT buckets. When connections are terminated for whatever reason, the corresponding entry is deleted from the respective bucket. Each node's buckets, and hence the DHT as a whole, therefore contains only active connections. *If* buckets were arbitrarily large, nearly all overlay links (E) would (also) be part of the buckets.

However, buckets are limited to $k = 20$ entries without an eviction mechanism. This leads to situations where connections are established but cannot be reflected in buckets. For example, during lookups, IPFS establishes connections to all newly discovered nodes and attempts to add them to buckets. To avoid maintaining a nearly infinite number of connections, IPFS nodes start to randomly terminate connections that are older than 30 s once their total number of connections exceeds 900. If relevant buckets are full at both nodes, the connection exists without being reflected in any bucket at all. Let $G' = (V', E')$ be the bucket-induced subgraph of G, i. e., $E' = \{(v_i, v_j) \in E : v_j \text{ is in a bucket of } v_i\}$ and $V' \subset V$ is the set of all nodes used in E'.

For visualizing the possible configurations, Fig. 2 depicts an example overlay network without clients (G), with the connections that are reflected in buckets (G') redrawn on top. A connection between two nodes can either

1) be reflected in buckets by both nodes (e. g., (v_2, v_3)),
2) be reflected in buckets by only one node (e. g., (v_0, v_1)) or
3) exist independently of bucket entries (e. g., (v_0, v_3)).

In practice, it therefore usually holds that E' is a strict subset of E and G' therefore strict subgraph of G. Unlike G, G' is furthermore a *directed* graph. The direction is important since lookups of data or other nodes *only* consider the nodes stored in buckets.

In the case of a well functioning DHT it is expected that $V' \equiv V$ and $\tilde{V} \setminus V'$ is exactly the set of client nodes.

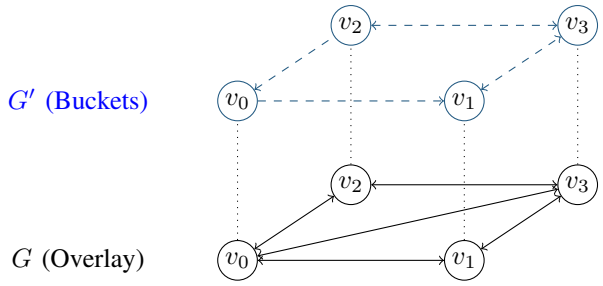

Figure 2: Types of connections: non-client overlay (bottom) and buckets (top). Not all overlay connections are reflected in the buckets, but every entry in a bucket corresponds to an overlay connection.

Event though it is likely that not all node pairs from E are reflected in E', we expect that G' is strongly connected and that it is therefore, in principle, possible to enumerate all non-client nodes V by crawling the DHT. Our crawling results (cf. Sec. VII-A) support this assumption. Before delving into our crawling methodology and the remaining insights gained through it, we present an empirically founded estimation of the relationship between $\tilde{G}$, G and G'.

V. Measuring the Interplay between $\tilde{G}$, G and G'

In the following, we focus on the default behavior of nodes (with and without NAT), the share of clients in the IPFS overlay (i. e., $\tilde{V} \setminus V$) and how much of the overlay of DHT-enabled nodes (G) is reflected in buckets (G').

A. Empirical Results of Monitoring nodes

We first started an IPFS node, version v0.4.22, with default settings and a public IP address, without any firewall. Every 20 s, we queried the number of overlay connections, as well as the number of nodes in the node's buckets for a total of 2.99 d. For the number of overlay connections we relied on the API that IPFS offers in this case, whereas the number of nodes was extracted through our external crawling tool (discussed in Sec. VI). The results are depicted in Fig. 3, which shows the number of overlay connections (edges from $\tilde{E}$, solid red line), connections with DHT-enabled nodes (edges from E, dashed green line) and the number of connections in the buckets (edges from E', dashed blue line). The measurement was started simultaneously with the IPFS node, hence the short start-up phase in the beginning. It can be seen that the number of connections largely fluctuates around a value, with up to 400 connections established in just a few minutes. This behavior is due to the way IPFS handles its connection limit.

The default number of connections an IPFS node will establish is 900, but it does not cap at that value. Instead, IPFS 1) starts a new connection with every node it encounters when querying data and 2) accepts every incoming connection at first. If the number of connections exceeds the limit, IPFS evicts connections (uniformly) at random that are older than 30 s, until the upper limit is satisfied again. Furthermore, 76.40 % of connections are DHT-enabled, on average, indicating a notable difference between the overlay with clients ($\tilde{G}$) and the one without (G). We performed

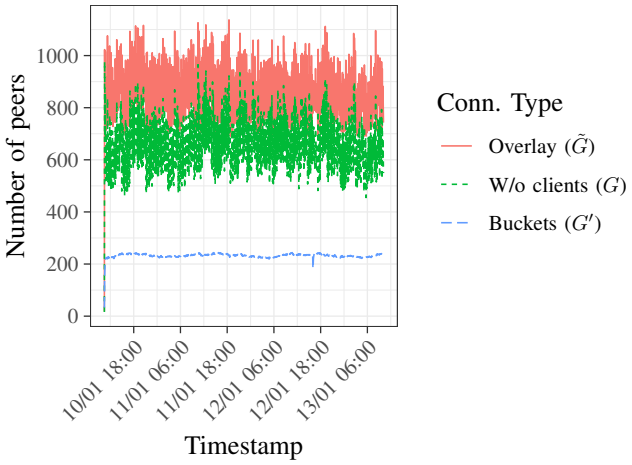

Figure 3: Number of connections of a default-settings IPFS node that is *not* behind NAT. We distinguish between all connections, connections not involving clients and connections corresponding to DHT bucket entries.

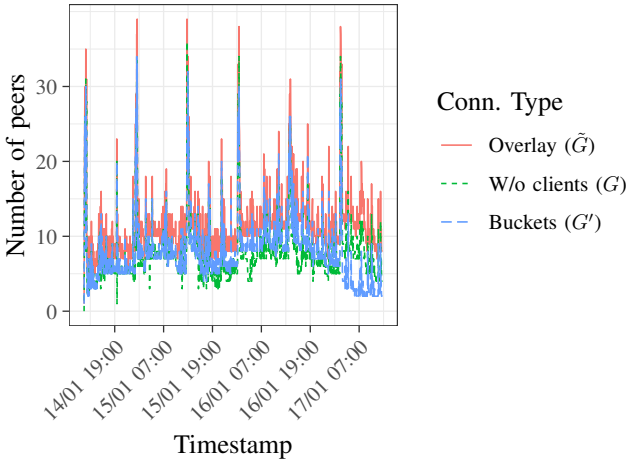

Figure 4: Number of connections of a default-settings IPFS node behind NAT.

the same experiment for a node behind a firewall for a duration of 3.05 d; the results are shown in Fig. 4. Several properties can be observed from both experiments. Firstly, a node behind a NAT has almost two orders of magnitude less connections than its non-NATed counterpart. Secondly, most connections of the non-NATed node are *inbound* connections, i. e., established from another peer on the network. Since our node was idle and not searching for content or other peers, it has no external trigger to start outbound connections. The NATed node cannot accept incoming connections, hence the low number of connections. Note that the presented number of DHT-enabled connections is occasionally smaller than the number of connections corresponding to DHT bucket entries. The former value is obtained via IPFS API functions that may lag behind in reporting the protocols of peers.

Last but not least, we are interested in a more holistic

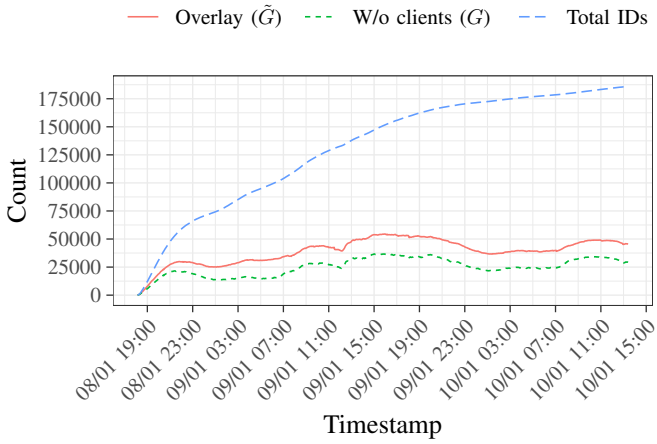

Figure 5: Number of IDs seen, overlay connections and connections to DHT-enabled peers of a node that has no connection limit.

perspective on the different overlay types and number of nodes in the network. To this end we started an IPFS node (v0.5.0-dev) with a public IP and no connection limit. Similar to the other experiments, we logged the connections every 20 s for a total of 1.80 d. The results are depicted in Fig. 5, which shows the number of connections over time, the number of DHT connections and the total number of node IDs seen. On average, the node had 38903 connections, 63.64 % of which were DHT connections. Again, the vast majority of these connections is inbound, since our node was idle. The number of node IDs is steadily increasing, whereas the number of connections is not, which could be due to a one-shot usage or people deleting the configuration directory which holds the private key to solve errors.

B. Analysis of the Difference between E and E'

In Fig. 3 it is clearly visible that the buckets (E') store 22.16 % of all connections between DHT-enabled nodes (E), due to the buckets' limited capacity of $k = 20$ nodes (cf. Sec. IV). The connections in E that are not reflected in buckets can, therefore, *not* be found by crawling the DHT nor be obtained through passive measurements alone. This raises the question: why is the gap between connections stored in a node's buckets (E') and all overlay connections between non-client nodes (E) so significant?

Analytically, we are interested in the following quantity: Given $N := |V|$ non-client nodes in the overlay, what is the expected number of nodes stored in the buckets? The distribution of nodes to buckets is highly skewed, since the first bucket is "responsible" for one half of the ID space, the second bucket for one fourth etc. [26], [27].

The expected number of nodes in each bucket i is therefore $\min\{20, N \cdot p_i\}$, with $p_i := 2^{-i}$. Although we do not encounter every peer equally likely, this reasoning still holds: During bootstrap, an IPFS node (in particular the DHT) performs lookups for its own node ID as well as random targets tailored

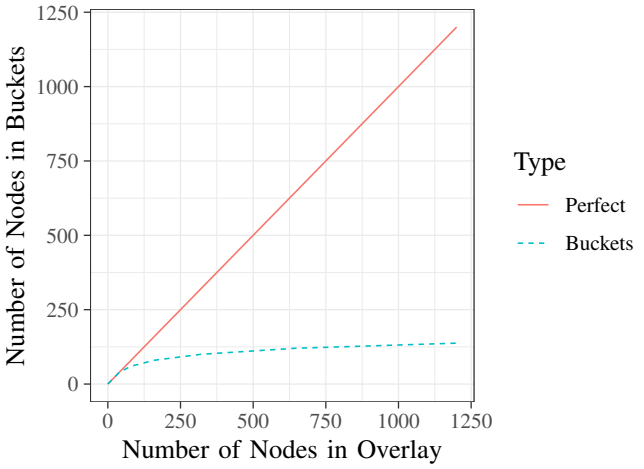

Figure 6: Analytical approximation: number of overlay connections vs. the expected number of nodes actually stored in the buckets. If buckets were infinitely large, every node would have an entry (red solid line). However, due to the structure and limited capacity of buckets we expect to see a logarithmic curve in reality (blue dashed line).

for each bucket. The former ensures that it knows *all* nodes in its direct neighborhood, partially filling the smallest, non-empty bucket. The latter ensures that it knows some nodes from each other bucket, filling them completely with high probability. Which bucket will be the smallest non-empty bucket therefore only depends on the total number of non-client nodes in the overlay.

Abusing notation, this yields:

$$\mathbb{E}[\#\text{ nodes in buckets}|N\text{ nodes in overlay}] \tag{1}$$

$$= \sum_{i=1}^{256} \mathbb{E}[\#\text{ nodes in bucket } i|N\text{ nodes in overlay}] \tag{2}$$

$$= \sum_{i=1}^{256} \min\{20, N \cdot p_i\}. \tag{3}$$

The result of this analytical consideration are depicted in Fig. 6. The solid red line corresponds to a "perfect" setting, where each overlay connection would be stored in a bucket, whereas the dashed blue line is the result of eq. (3). Plugging the empirically found number of nodes from Sec. VII into this formula yields an expected number of bucket entries between 232 and 247, which coincides with the measured average number of entries of 232.47 (max. 245).

So far, we've seen some indications on the relationship between the IPFS overlay $\tilde{G}$, the overlay without clients G and what is stored in buckets G'. In the following, we are interested in obtaining measurements of G' to learn more about the topology of the network.

VI. Crawling the Kademlia DHT

A. Crawl Procedure

As described in Sec. III, nodes in the network are identified through a (multi-)hash of a public key. The crawler used new key pairs in every run to thwart potential biases due to re-occurring IDs. Moreover, an IPFS node will not store our crawling nodes in its buckets, since our nodes are marked as clients who do not actively participate in the Kademlia communication[5].

Nodes can be found by issuing FindNode packets for some target ID, which can be a key or a node ID (=hash of a public key in the case of nodes). To completely crawl a node, one has to send a FindNode packet for each possible bucket. This is due to the fact that a node returns its k closest neighbors to the target provided in a FindNode packet. The closest neighbors to the target are the ones in the bucket the target falls into. If the bucket in question is not full (i. e., less than k entries), the closest neighbors are the ones in the target bucket and the two buckets surrounding it. Since the bucket utilization of remote nodes is unknown, we do not know in advance how many requests per node we have to send for obtaining all of its DHT neighbors. We therefore send FindNode packets to targets with increasing common prefix lengths and stop once no new nodes are learned. This results in a significant speed improvement as no requests are sent for buckets that are nearly certainly empty (since the number of potential bucket entries decreases exponentially with the common prefix length). Faster crawls, in turn, enable us to capture more accurate snapshots of the dynamically changing network.

B. Hash Pre-Image Computation

Unlike more canonical Kademlia implementations, whenever an IPFS node receives a FindNode packet, it hashes the provided target and searches for the nearest neighbors to that hash. Therefore, to know which target to send for which bucket, we need to compute pre-images that, when hashed, yield the desired common prefix length between FindNode target and the node ID of the node we are crawling. To that end, we generated pre-images for every possible prefix of 24-bit length. In other words, we computed 2^{24} pre-images such that, for each possible prefix of the form $\{0,1\}^{24}$, there is a hash starting with that prefix.

Equipped with this table, one lookup operation is sufficient to pick a pre-image that, when hashed by the receiver, will yield the desired bucket. Note that this table can be used for an arbitrary number of crawls, hence the computation only had to be performed *once*.

C. Crawl Procedure

Each crawl commenced with the four IPFS default bootnodes and continued from there by sending FindNode packets for each common prefix length (and therefore bucket) until no more new nodes were learned. Therefore, if the IPFS network were static and nodes replied to requests deterministically, a crawl would always yield the same results. Due to the inherent dynamics in the overlay this is not the case: repeated crawls allow us to observe changes in the overlay over time.

[5]Nodes announce the list of protocols they can serve to each other.

Session Duration	Percentage	Number of sessions
5 minutes	56.9273	2188153
10 minutes	26.1066	1003478
30 minutes	2.6809	103049
1 hour	0.4289	16486
1 day	8e-04	32
6 days	1e-04	2

Table I: Inverse cumulative session lengths: each row gives the number of sessions (and total percentage) that were *longer* than the given duration.

VII. Crawling Results

We repeatedly crawled the IPFS network from 2019-11-26 14:44:57 until 2019-12-03 14:12:02 for a total duration of 6.98 d. Crawls were started one at a time and back to back, in the sense that as soon as a crawl was finished, a new one was started. We performed a total of 2400 crawls, with a single crawl taking an average of 4.10 min to complete.

A. Number of Nodes, Reachability and Churn

To get an idea for the size of the network, we first focus on the number of nodes and their session lengths. During our 6.98 d, we found a total of 309404 distinct node IDs, with an average number of 44474 per crawl. This is consistent with the results obtained in Sec. V, hinting that both methods provide an adequate view of V, the set of non-client nodes in the IPFS overlay. Surprisingly, of all the nodes that were queried, the crawler was only able to connect to 6.55 %, on average.

We suspect that most IPFS nodes are run by private people connected through NAT. This hypothesis is supported by our results: about 52.19 % of all nodes report only local IP addresses for other nodes to connect to, which is exactly the behavior of nodes behind symmetric NATs (cf. Sec. VII-B). Furthermore, if most nodes are behind NATs, they are also prone to short uptimes, since these are probably used as client nodes which are shut down after use.

Another indicator for the client-node-hypothesis are the session lengths, which are depicted in Table I. We define a session as the time difference between the crawl, when we were able to reach the node and when it became unreachable again. The table depicts the inverse cumulative session lengths: each row yields the number of sessions (and their percentage) that were longer than the given duration. For example, roughly 56 % of all sessions were longer than 5 min, or equivalently, 44 % of all sessions were *shorter* than 5 min. This indicates that participation in IPFS is dynamic and rather short-lived. We also observed a periodic pattern in the number of nodes found through crawling, as shown in Fig. 7. The figure distinguishes between all nodes and nodes that were reachable, i. e., the crawler was able to establish a connection to these nodes. The significant increase in the number of nodes at the end of the measurement period could stem from other researchers' nodes or even an attack on the network. A daily fluctuation of node numbers, peeking at around 7pm UTC each day, can clearly be identified. The (comparatively small) number of *reachable* nodes, on the other hand, does not fluctuate as much. These observations support the hypothesis that most nodes are

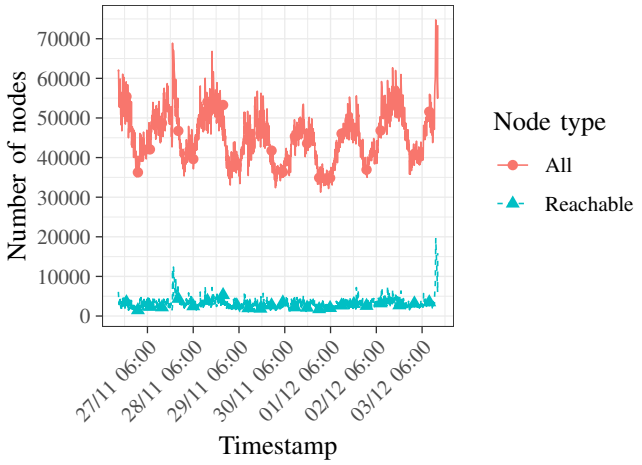

Figure 7: Number of nodes over time, distinguished by all and reachable (=answered to our query) nodes. Times are in UTC.

operated by private users behind NAT and that these users run the IPFS node software on an as-needed basis.

One could argue that private users behind NAT will tend to use their nodes in their spare time, e. g., in the evening. A usage pattern with rising usage numbers in the UTC afternoon would imply that many users reside in Asia, since UTC afternoon corresponds to evening times in, e. g., China. This conclusion is in tune with the distribution of nodes over countries we observe.

B. Node Distribution over Countries and Protocol Usage

All			Reachable		
Country	Count	Conf. Int.	Country	Count	Conf. Int.
LocalIP	23973.61	±173.85	US	1721.08	±26.67
CN	5631.37	±11.27	FR	635.71	±17.65
DE	4224.76	±33.37	DE	569.23	±8.82
US	4091.64	±54.13	CA	119.01	±2.32
FR	1390.2	±33.33	PL	73.63	±2.09
CA	978.17	±19.87	CN	58.61	±0.49
PL	693.08	±17.97	GB	26.67	±0.23
IL	321.07	±2.63	SG	20.98	±0.15
GB	171.08	±0.91	IL	15.09	±0.35
HK	168.09	±1.96	NL	12.95	±0.12

Table II: The top ten countries with the most nodes per crawl, differentiated by all discovered nodes and nodes that were reachable. Depicted is the average count per country per crawl with .95 confidence intervals (student-t).

Protocol	Perc. of peers	Abs. count
ip4	99.9893	309369
ip6	80.4862	249026
p2p-circuit	1.5313	4738
ipfs	1.2737	3941
dns4	0.0669	207
dns6	0.0301	93
dnsaddr	0.0039	12
onion3	3e-04	1

Table III: Protocol capabilities of nodes: over all crawls, we noted the protocols supported by each peer. The vast majority of nodes support IPv4 and IPv6, whereas other protocols are barely used.

Table II depicts the top ten countries, both for all discovered nodes and for nodes that were reachable by our crawler[6] These ten countries contain $91.04\,\%$ ($93.67\,\%$ in the case of reachable nodes) of the whole network, already hinting at centralization tendencies regarding the spatial distribution of nodes.

Again, it can be seen that $52.19\,\%$ of all nodes *only* provide local or private IP addresses, thus making it impossible to connect to them. This is in line with the default behavior of IPFS when operated behind a NAT. When a node first comes online, it does not know its external IP address and therefore advertises the internal IP addresses to peers it connects to. These entries enter the DHT, since IPFS aims to provide support for building private IPFS networks. Over time, a node learns about its external multi-addresses (containing IP-addresses and ports, cf. Sec. III) from its peers. An IPFS considers these observed multi-addresses reachable, if at least four peers have reported the same multi-address in the last 40 minutes and the multi-address has been seen in the last ten minutes. This is never the case for symmetric NATs, which assign a unique external address and port for every connection, yielding a different multi-address for every connected peer.

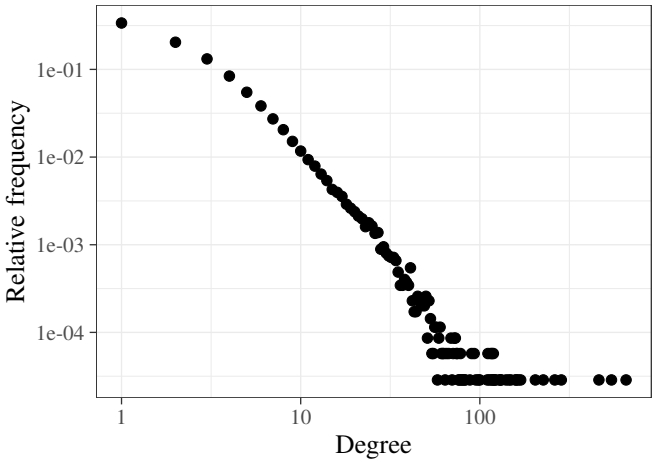

Figure 8: Log-log in-Degree distribution from the first crawl including *all* found nodes. Other crawls yielded similar shapes.

Furthermore, there is a significant difference visible especially between China and the U.S.: although most IPFS nodes are in China, the vast majority of reachable nodes reside in the U.S.

As stated in Sec. III, IPFS supports connections through multiple network layer protocols; Table III shows the prevalence of encountered protocols during our crawls. If a node was reachable through multiple, say IPv4 addresses, we only count it as one occurence of IPv4 to not distort the count. The majority of nodes support connections through IPv4, followed by IPv6, whereas other protocols are barely used at all. The protocols "ipfs" and "p2p-circuit" are connections through IPFS' relay nodes, "dnsaddr", "dns4/6" are DNS-resolvable addresses and "onion3" signals TOR capabilities.

[6]We use the GeoLite2 data from MaxMind for this task, available at https://www.maxmind.com.

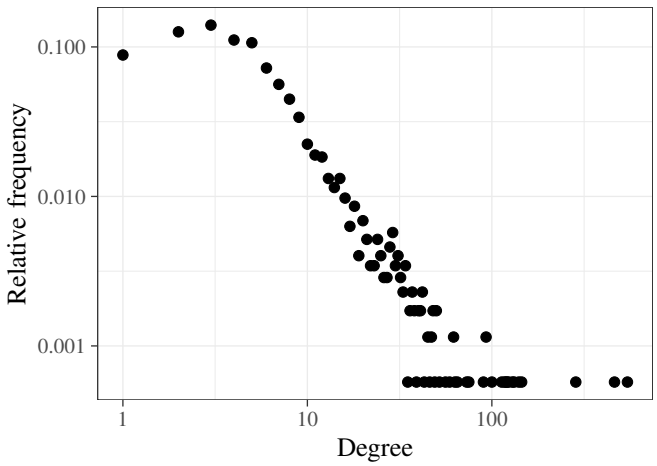

Figure 9: Log-log in-Degree distribution from the first crawl including only *reachable* (i. e. online) nodes. Other crawls yielded similar shapes.

C. Overlay Topology

	Min.	Mean	Median	Max.
total-degree	1	14.32	10.2	942.23
in-degree	1	7.16	3.03	935.48
out-degree	0	7.16	6.48	53.56

Table IV: Degree statistics over all 2400 crawls.

A crawl yields a view on the nodes V' in the buckets $G' = (V', E')$, of non-client nodes, and their connections to each other, i. e. E'. As discussed in Sec. IV, we obtain a measurement of V' (which is approximately V) and E' which is a strict subset of E. Nevertheless, $G' = (V', E')$ is the graph used by IPFS to locate data and nodes through the DHT and therefore provides a lot of insights.

Since there is a huge discrepancy between all discovered and the reachable nodes, and since we cannot measure the properties of unreachable nodes, we distinguish these two classes in the analysis.

Fig.s 8 and 9 depict the log-log in-degree distribution from the first crawl; note that other crawls yielded similar results. We differentiate between all found nodes (Fig. 8) and only reachable nodes (Fig. 9). The (roughly) straight line in the figure indicates a highly skewed distribution where some peers have very high in-degree (up to 1000) whereas most peers have a fairly small in-degree. In other words, the in-degree distribution can be approximated by a power-law. The graph can therefore be classified as scale-free, which is in line with prior measurements on other Kademlia systems [10].

In the following, we specifically focus on the top degree nodes. Top degree nodes were defined as the $0.05\,\%$ of all nodes, yielding 22.2 nodes on average. We refrain from defining a number, say the 20 nodes with the highest degree, since this would weigh crawls with fewer nodes differently than crawls with a higher number of total observed nodes. Fig. 10 depicts a cumulative distribution of how many times the nodes with the highest degree were seen in the crawls.

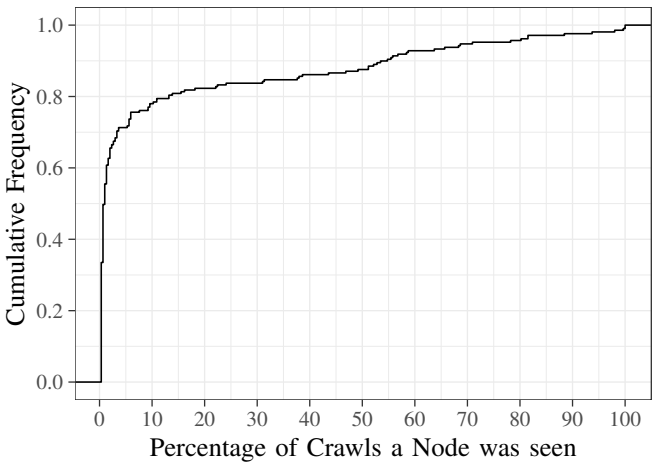

Figure 10: ECDF of how often the same nodes were within the top-degree nodes. This metric indicates whether the network has a "stable core" of infrastructural nodes with a high degree.

If a node was in the set of highest-degree nodes in one run but not in other runs, its "percentage seen" would be $\frac{1}{\#\text{ of crawls}} = \frac{1}{2400} = 0.0033$ or $0.33\,\%$. On the other extreme, if a node was within the highest degree nodes in every crawl, its percentage seen would be a $100\,\%$.

The cumulative distribution in Fig. 10 shows a high churn within the highest degree nodes: approximately $80\,\%$ of nodes were only present in $10\,\%$ of the crawls. Only a few nodes have a high degree in the majority of all crawls; these nodes are the bootstrap nodes along a handful of others.

VIII. Conclusion and Future Work

Founded in an analysis of network-layer mechanisms actually in use in the current IPFS implementation (respectively libp2p), we've presented several approaches towards mapping the IPFS overlay network. Using monitoring nodes and a $6.98\,\mathrm{d}$ crawl of the IPFS DHT, we obtain a holistic view on IPFS' node population as well as indicators about the larger overlay structure. Of the 44474 nodes that we found during each crawl on average, only $6.55\,\%$ responded to our connection attempts. This, among other indicators, hints at the fact that a majority of nodes is operated by private individuals. Lastly, we find that due to broadcast-based content lookups and the maintaining of many non-DHT connections, IPFS can be classified as a hybrid between a structured and an unstructured overlay; implying worse performance but improved robustness when compared to a classical DHT.

Towards future works, we observe that our monitoring nodes can be extended to also record data request patterns (similarly to studies conducted in the Bitcoin peer-to-peer network [28]). As IPFS nodes broadcast each sought data item to *each* of their connected peers (cf. Sec. III-F), a node that is connected to every other node could monitor nearly *all* data requests in the network. While enabling insights into the "filesystem" side of IPFS, the success of such a measurement approach also implies in dire implications for the query privacy of IPFS users.

References

[1] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. arXiv: 1407.3561.

[2] M. Li, J. Weng, A. Yang, *et al.*, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, 2018.

[3] A. Mühle, A. Grüner, T. Gayvoronskaya, and C. Meinel, "A survey on essential components of a self-sovereign identity," *Computer Science Review*, vol. 30, 2018.

[4] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, "Blockchain based data integrity service framework for iot data," in *Proc. of ICWS*, IEEE, 2017.

[5] A. Tenorio-Fornés, V. Jacynycz, D. Llop-Vila, A. Sánchez-Ruiz, and S. Hassan, "Towards a decentralized process for scientific publication and peer review using blockchain and ipfs," in *Proc. of HICSS*, 2019.

[6] O. Ascigil, S. Reñé, M. Król, *et al.*, "Towards peer-to-peer content retrieval markets: Enhancing IPFS with ICN," in *Proc. of ICN*, ACM, 2019.

[7] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips, "The bittorrent P2P file-sharing system: Measurements and analysis," in *Proc. of IPTPS*, Springer, 2005.

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[9] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. of IPTPS*, Springer, 2002.

[10] H. Salah, S. Roos, and T. Strufe, "Characterizing graph-theoretic properties of a large-scale DHT: measurements vs. simulations," in *Proc. of ISCC*, IEEE, 2014.

[11] I. Baumgart and S. Mies, "S/kademlia: A practicable approach towards secure key-based routing," in *Proc of ICPADS*, IEEE, 2007.

[12] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long term study of peer behavior in the KAD DHT," *Trans. Netw.*, vol. 17, no. 5, 2009.

[13] G. Memon, R. Rejaie, Y. Guo, and D. Stutzbach, "Large-scale monitoring of DHT traffic," in *Proc. of IPTPS*, Springer, 2009.

[14] J. Yu and Z. Li, "Active measurement of routing table in kad," in *Proc. of CCNC*, IEEE, 2009.

[15] D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the gnutella network," in *Proc. of INFOCOM*, IEEE, 2006.

[16] M. Steiner, T. En-Najjary, and E. W. Biersack, "A global view of kad," in *Proc. of SIGCOMM*, ACM, 2007.

[17] H. Salah and T. Strufe, "Capturing connectivity graphs of a large-scale p2p overlay network," in *Proc. of ICDCS Workshops*, IEEE, 2013.

[18] K. Junemann, P. Andelfinger, and H. Hartenstein, "Towards a basic dht service: Analyzing network characteristics of a widely deployed dht," in *Proc. of ICCCN*, IEEE, 2011.

[19] M. Steiner, E. W. Biersack, and T. En-Najjary, "Actively monitoring peers in KAD," in *Proc. of IPTPS*, Springer, 2007.

[20] D. Stutzbach and R. Rejaie, "Evaluating the accuracy of captured snapshots by peer-to-peer crawlers," in *Proc. of PAM*, Springer, 2005.

[21] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding I/O performance of IPFS storage: A client's perspective," in *Proc. of IWQoS*, ACM, 2019.

[22] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a fog/edge computing infrastructures," in *Proc. of CloudCom*, IEEE, 2016.

[23] B. Confais, A. Lebre, and B. Parrein, "An object store service for a fog/edge computing infrastructure based on ipfs and a scale-out nas," in *Prof. of ICFEC*, IEEE, IEEE, 2017.

[24] J. R. Douceur, "The sybil attack," in *Peer-to-peer Systems*, Springer, 2002.

[25] D. Mazières, "Self-certifying file system," PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2000.

[26] S. A. Henningsen, D. Teunis, M. Florian, and B. Scheuermann, "Eclipsing ethereum peers with false friends," in *Proc. of EuroS&P Workshops*, IEEE, 2019.

[27] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," *IACR*, vol. 236, 2018.

[28] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *Proc. of UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, IEEE, 2016.