Crawling the IPFS Network

Sebastian Henningsen Sebastian Rust Martin Florian Björn Scheuermann

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

***Abstract*—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 both are distributed using a Kademlia-based distributed hash table (DHT). In this demo, we present a crawler for the IPFS overlay network (ipfs_crawler) that can be used to study and monitor the network's structure. Therefore, ipfs_crawler is an important building block when assessing the state and health of the network, as the overlay network significantly influences the robustness and performance of IPFS. Specifically, ipfs_crawler systematically traverses the Kademlia DHT of IPFS to enumerate peers in the network and a subset of the connections between peers. Since network communication in IPFS is carried out through the libp2p networking library, ipfs_crawler can easily be adapted to crawl other libp2p-based networks.**

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] 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 [3]. Any Internet-enabled device can participate as an IPFS node and nodes are operated without explicit economic incentives. 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 [4].

In [2], we conduct a measurement study on the IPFS network; contrasting white papers and documentation with the actual code and performing extensive measurements regarding, among others, the number of nodes, their geographic distribution and the graph properties of the resulting overlay.

In this demo, we shed more light on ipfs_crawler[2], the crawler which lies at the heart of our evaluation in [2]. Although we specifically developed ipfs_crawler from scratch to generate snapshots of the IPFS overlay network, its potential applications transcend IPFS. IPFS' network layer is implemented in the libp2p networking library, which originated as part of the IPFS project but was modularized into its own standalone library. Therefore, ipfs_crawler can easily be adapted to crawl other libp2p-based networks. In the following, for easier comprehension and in tune with [2], we focus on crawling the IPFS and will refer to the IPFS node software as a monolith that implicitly contains all libp2p functionality.

We resorted to developing our own crawler instead of reusing existing Kademlia crawlers [5] for two reasons: first, IPFS' protocol and handshake structure is highly complex and best used with the provided API, which would be hard to integrate in existing crawlers. Second, and more importantly, although the crawling *literature* on Kademlia is vast, there are virtually no open source implementations.

Therefore, we made our code public and are cooperating with IPFS/libp2p developers to incorporate the crawler into a periodic monitoring infrastructure, since ipfs_crawler is an effective tool to assess the state and health of the network. Additionally, the obtained data and insights from crawling may be useful for engineers and further research on, e. g., performance and resilience.

II. The Interplanetary Filesystem

IPFS is an open, permissionless system, i. e., anybody can participate, host data and download data from other peers. Data items are stored and served by data providers which announce the data they are serving to the network. These references between data items and their respective providers are stored in a global Kademlia-style DHT.

As in other Kademlia implementations, nodes maintain a routing table of peers, with each peer entry containing a mapping between peer ID and the addresses (e. g., IP addresses) under which the peer can be reached in the underlying network. In contrast to classical Kademlia implementations, DHT-communication in IPFS is carried out over TCP and other connection-oriented transports like QUIC. Therefore, if a node has another peer's routing entry stored in its local table, there exists an active overlay connection between them. Notably, the inverse is not necessarily true as not every established overlay connection is reflected in the DHT routing table (cf. [2] for more details).

III. Crawl Procedure

ipfs_crawler starts by connecting to the IPFS bootstrap nodes, collecting their routing table content and successively trying to connect to every peer it has not tried before. IPFS

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

[2] The code of our ipfs_crawler and evaluation is maintained at https://github.com/wiberlin/ipfs-crawler. We use ipfs_crawler to conduct and visualize periodic measurements at https://trudi.weizenbaum.net.

Annex to ISBN 978-3-903176-28-7

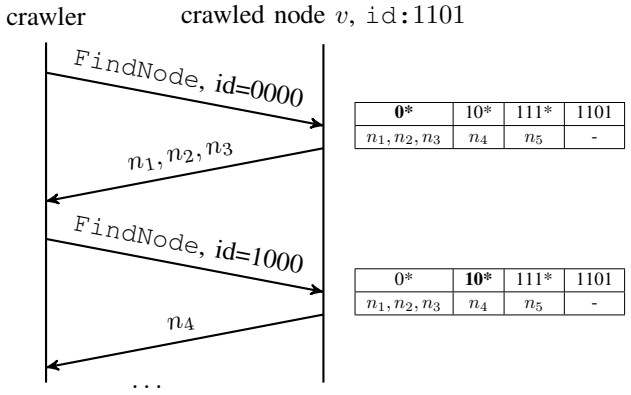

Figure 1: Sequence diagram of the crawl.

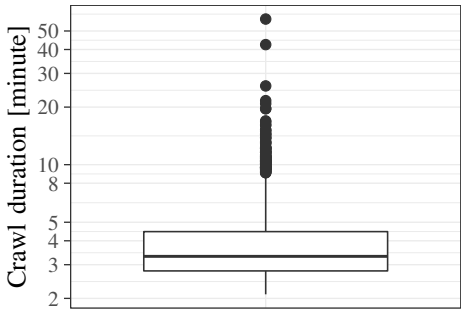

Figure 2: Boxplot of crawl durations for [2]

employs a variant of Kademlia, hence, nodes organize their routing table entries in so-called *buckets* based on the leading zeroes of the xor between their own node ID and the other nodes' IDs. As an example, consider Fig. 1 which depicts the crawling process for one node and said nodes' bucket organization. The receiver's ID is 1101. The first bucket contains only peers whose ID xor 1101 starts with no leading zeroes, hence, node IDs starting with 0. Similarly, for an ID to be stored in the second bucket, the xor has to start with one leading zero, yielding IDs starting with 10, and so on.

ipfs_crawler sends a Kademlia `FindNode`-request with a target ID, to which the receiver will answer with the $k = 20$ routing table entries that are closest to the target. These nodes are exactly the ones in the bucket of the target ID (and surrounding buckets, if the target bucket is not full).

IV. Features

The most important aspect of every crawler is its speed, as it directly influences the accuracy of obtained snapshots [6]. Since the overlay is changing during crawls due to peer churn, the longer a crawl takes, the higher the risk of unwanted artifacts in the snapshots [7]. Therefore, our ipfs_crawler is optimized for small crawl times and is able to crawl 50000 nodes in roughly $4\,\mathrm{min}$, on average. Figure 2 depicts a boxplot summarizing the distribution of crawling times for the data in [2]. The box corresponds to the $25\,\%$ and $75\,\%$ quartiles, respectively, with the median shown as the solid line. It can be seen that $75\,\%$ of crawls took less than $5\,\mathrm{min}$ to complete and only $2.71\,\%$ of crawls took longer than $10\,\mathrm{min}$.

Furthermore, if configured, ipfs_crawler will cache the nodes it has seen. The next crawl will then not only start at the bootstrap nodes but also add all previously reachable nodes to the crawl queue. This caching additionally increases the crawl speed, since it overcomes the bottleneck of finding peers to connect to in the beginning of each crawl. For every peer it encounters, ipfs_crawler saves the following information:

- the peer ID, i. e., the (multi-)hash of a public key,
- all available addresses (e. g., IPv4, IPv6, relay, ...) of the peer and
- whether a connection could be established.

If a connection attempt was successful, it also includes

- the agent version and
- the content of the routing table entries as an edge list.

Taking the example in Fig. 1, the edge list obtained from crawling node v would be $(v, n_1), (v, n_2), \ldots, (v, n_5)$.

ipfs_crawler enumerates the nodes in the network, but without ground truth it is hard to assess the quality and completeness of a crawl. Therefore, it might be desirable to perform a sanity check whether some pre-defined IPFS-nodes are found through crawling. These can be well-known nodes, such as the ipfs.io-gateway, or self-run nodes, specifically started for the purpose of sanity checking the results. If provided, ipfs_crawler checks if it found the respective nodes and notifies the user in case it was not successful.

V. Conclusion

In this demo we present ipfs_crawler, a crawler for the overlay network of IPFS which lies at the heart of our journey towards mapping the IPFS [2]. In particular, ipfs_crawler obtains snapshots of the Kademlia DHT of IPFS, which is implemented in the libp2p networking library. Hence, ipfs_crawler is not limited to IPFS. but could easily be used to crawl other libp2p-based networks.

Towards future work, we are cooperating with IPFS/libp2p developers to integrate ipfs_crawler into a periodic monitoring framework for assessing IPFS' network state and health.

References

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

[2] S. A. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Mapping the interplanetary filesystem," in *Proc. of IFIP Networking*, IFIP, 2020.

[3] 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.

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

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

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

[7] ——, "Evaluating the accuracy of captured snapshots by peer-to-peer crawlers," in *Proc. of PAM*, Springer, 2005.