Evaluating IPFS Optimistic Provide in the Wild

Fotios Bistas and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics
Athens University of Economics and Business, Greece
E-mail: {fot.bistas,xgeorge}@aueb.gr

***Abstract*—The InterPlanetary File System (IPFS) network is a very successful peer-to-peer distributed storage system, offering high resilience in the face of significant peer churn which may, however, lead to considerable delays when storing content. The Optimistic Provide (OP) algorithm is a heuristic scheme that reduces IPFS store latency, at the possible cost of making retrieval less reliable. This paper assesses whether content retrieval when the OP algorithm is used is equally reliable to that of plain IPFS, based on an extensive measurement campaign carried out in the actual IPFS network, using tools that we extended for this purpose. Our results indicate that the OP algorithm offers content retrievability that is on par with plain IPFS.**

***Index Terms*—IPFS, Optimistic Provide, Kademlia, DHT.**

I. Introduction

The *InterPlanetary File System* (IPFS) [1] is a peer-to-peer system for storing and sharing content. The *IPFS network* is a realization of the IPFS design, which is present in over 150 countries and more than 2500 autonomous systems. IPFS is built on top of the Kademlia *Distributed Hash Table* (DHT) [2], which, like other DHTs, maps both content and peers to a common identifier space, assigns content to peers in a systematic way and allows looking up content in logarithmic time over the number of peers. Both IPFS and its Kademlia implementation are open source.

The IPFS network is permissionless, so anyone can participate in it at any time. The constant addition and removal of peers in IPFS, also known as *peer churn*, means that the content routing tables that Kademlia maintains are often outdated, containing *dead links*, that is, links to departed peers. As peers leave unannounced, dead links are only discovered when peers do not respond to a query before a timeout expires.

While dead links complicate both content storage and retrieval, they have a disproportionate impact on the former, due to a fundamental performance imbalance. When content is stored, IPFS must locate *multiple* appropriate peers to store pointers to that content for resilience. However, when content is looked up, locating just *one* of these peers is sufficient. As a result, discovering content in plain IPFS takes about 1.5 s, while storing content sometimes takes up to 60 s [1].

To remedy this shortcoming, the *Optimistic Provide* (OP) algorithm was proposed [3]. The main idea behind OP is to first estimate the distribution of peers in the Kademlia identifier space, and then make educated guesses on whether a known (and live) peer is appropriate for a store operation, without exhaustively searching the DHT for the most appropriate peers. This reduces the delays caused by dead links.

The OP algorithm is not perfect: it may store some content pointers at the wrong peers, thus producing errors in future lookups. This paper aims to examine whether content stored using the OP algorithm is as retrievable as in plain IPFS. Unlike the paper proposing the OP algorithm which focuses on its *store* performance [3], we focus on its *lookup* performance and reliability. To achieve this, we ran a measurement campaign in the real IPFS network, using the IPFS-CID-HOARDER tool, which we expanded with additional features. Our results indicate that the lookup performance of the OP algorithm is close enough to plain IPFS for general use. Coupled with the findings of the paper proposing OP, which show OP's superior store performance compared to plain IPFS, OP has since been incorporated in the IPFS code base.

The outline of the rest of this paper is as follows. Section II provides background on IPFS, Kademlia, OP and related work. Section III describes the tools we used and the extensions that we made for this study. Section IV presents the methodology for our experiments, while Section V presents and analyzes the results of our measurements. We summarize our findings and discuss future work in Section VI.

II. Background and Related Work

A. IPFS and Kademlia

The IPFS network operates over a customized version of the Kademlia DHT. Each piece of content is identified by a *Content Identifier* (CID), which is produced by running the content through a cryptographic hash function. In turn, each IPFS peer generates a pair of public-private keys and uses the same hash function over its public key to produce its *Peer Identifier* (PID). The default hash function is SHA-256, which means that both CIDs and PIDs are 256-bits long. The use of cryptographic hash functions for CIDs and PIDs has some interesting implications. First, content is self-certifying, since it must match its CID. Second, content is immutable, as any change to a block leads to a different CID. Third, peers are also self-certifying, since they can prove ownership of their PID via their public-private key pair.

Kademlia uses the XOR (Exclusive-OR) metric to determine the distance between any two identifiers (IDs): it XORs the two IDs together and considers the result as the distance between them. To navigate the ID space, each peer maintains an array of *K-Buckets*, with each bucket containing K other peers with the following property: the x^{th} K-Bucket stores pointers to peers that share a common PID prefix with that peer of length $x-1$. Since PIDs are 256 bits, a peer maintains

256 K-Buckets each holding pointers to $K = 20$ peers. Using the K-Buckets, Kademlia can locate the XOR-based "closest" peers to any given ID in a logarithmic number of steps.

B. Content provision and provider records

When a peer desires to provide a piece of content via IPFS, it first calculates its CID and then it creates a *Provider Record* (PR), a pointer to the peer that provides the content. Then, it uses the Kademlia DHT to lookup the $K = 20$ *closest* peers, in XOR distance, to the content's CID. Finally, it asks these peers to store the PR for the content. This means that pointers are distributed across multiple peers and can be accessed even if some of the peers are down. To retrieve the content, a peer uses the Kademlia DHT to lookup the desired CID; once a node holding the correct PR is located, the peer can then request the content from its hosting peer.

To ask a peer to store a PR, we use the *ADD_PROVIDER Remote Procedure Call* (RPC) of the Kademlia DHT. This notifies the remote peer that it is in the set of K closest peers to the provided CID. If the PR is successfully inserted into the remote peer, the target peer is added to the *provider store* of the peer providing the content. The provider also locally stores the PR, in case some peer contacts it directly.

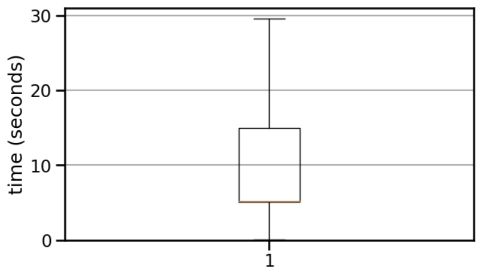
An important property of PRs is that they are automatically removed after a period. During our experiments, this period was 24 hours, as specified by the network protocol.If the provider wanted to keep the content accessible, it should republish the PR before it was removed; the recommended interval for republishing content was 12 hours. As a result, peers who are no longer among the K closest to the CID, automatically drop the PR, while the now K closest peers start storing it. After our study, the deletion interval was extended to 48 hours and the republish interval to 22 hours, reflecting the network's evolving requirements.

C. Hydra boosters

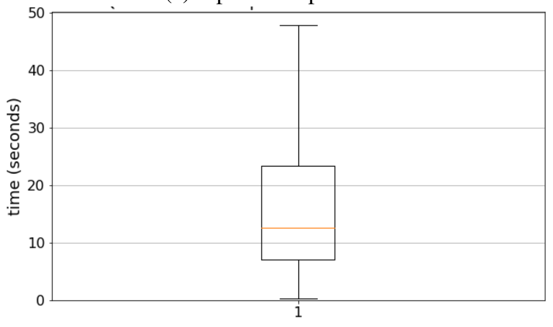
To accelerate content routing, the IPFS network includes *Hydra peers*. Hydras use multiple PIDs distributed in the address space; each of these PIDs are one of the Hydra's *heads*. Due to their multiple PIDs, peers are more likely to contact Hydras than regular peers. When Hydras receive *ADD_PROVIDER* RPCs from peers, they store the PRs in a large, Hydra-wide, database. All Hydra heads can access the database when they receive a *GET_PROVIDERS* RPC. As a result, Hydras can make large jumps in the address space. While our experiments were being conducted, Hydras were being dialed down in the network, to determine their impact. The dial down meant that while a provide operation might have inserted a PR in a Hydra peer, theHydra peer would *not* respond with the PR in a lookup request.

D. The Optimistic Provide algorithm

Providing content in the IPFS network is slower than looking it up, since providing content requires locating *all* K closest peers, while discovering content requires locating just *one* of them. To locate the K closest peers to a CID, a peer starts with the closest peers to that CID from its own K-Buckets and probes them for their own closest peers; this process is repeated recursively, until no more closest peers are returned. Unfortunately, due to *peer churn*, some of the peers in a host's K-Buckets may have left the network, leading to timeouts. In addition, some probes might fail, due to the fact that remote peers may be overloaded. Finally, since the peers are sparsely distributed in the ID space, each peer only has a few other peers close to it. All these factors mean that many messages need to be sent to determine the K closest peers to a CID, many of which go unanswered.



(a) Optimistic provide.



(b) Plain IPFS provide.

Fig. 1: Distribution of content provisioning time.

The *Optimistic Provide* (OP) algorithm[1] aims to decrease the time required to publish the content in the IPFS network, by using an "optimistic" approach when publishing content. Using an *estimate* of the network size and calculating the XOR distance between the CID that we want to store and the PID of a known, live peer, we can assess if that peer is likely to be among the K closest to the CID, without performing a full DHT search. The goal is to rely as far as possible on *known live* peers, rather than probing *possibly dead* ones.

To explain how OP works, consider a candidate PID P, a CID C and an IPFS network with size N. Taking the XOR distance of the PID and the CID, normalizing it to $[0, 1]$ (dividing by 2^{256}) and multiplying by the network size we get $\mu = ||P - C|| * N$, which is the number of peers that we expect to lie between P and C. If μ is less than *K = 20* we store the PR at peer P, as its highly likely that the peer is among the K closest peers to the CID. More details on how the network size is estimated can be found in [3].

As our measurements in Figure 1 show (see Section IV for details), this can significantly decrease the time to provide

[1]https://github.com/dennis-tra/optimistic-provide

content: the mean time for content provision using OP is around 5 s, while the mean time to provide content using plain IPFS is around 15 s. The maximum outlier time of OP is around 30 s, while the maximum outlier time of plain IPFS is around 50 s, close to the 60 s reported in previous work [1].

E. Related Work

Recent research indicates that IPFS suffers from high rates of churn: 87.6% of user sessions last for less than 8 hours [1]. This leads to slow content provision, which takes over 33 s for 50% of the provision requests, since the high churn rate leads to inaccurate routing state and timeouts. There is also some work evaluating the performance of the Kademlia DHT as implemented for the BitTorrent network, characterizing the performance penalty for lookups due to peer failures [4], [5].

Many approaches have been considered to improve the performance of DHTs in general, especially as networks grow larger, such as caching [6], network-aware peer selection [7], parallel lookups [8] and hierarchical organization of the DHT based on various criteria [9]. However, none of these works deal with improving the performance of stores. The only work focusing on the optimization of content provision in IPFS that we are aware of, is the paper proposing the OP algorithm [3].

In that paper, the main goal of the evaluation was to assess the latency and the success rate of provision requests; the rest of the evaluation is concerned with the accuracy of estimating network size and distances between peers. In this work, we focus instead on *content retrievability over time*, by looking at how many of the originally selected nodes remain available, and how many can be reached via the DHT, which accounts for the non-perfect selection of peers by OP. In a sense, while the OP paper focuses on provisioning, we focus on retrieval.

III. Measurement Tools

In order to gather data about the behavior of the IPFS network, we used the IPFS-CID-HOARDER tool[2]. This tool probes peers to find which PRs they are holding, gathering various metrics along the way. This data is added to a database for further analysis. The regular operation of IPFS-CID-HOARDER consists of publishing PRs for random CIDs, and then looking them up and extracting information about how the IPFS network treats PRs overall. To perform this study, we modified the hoarder to load in its database a set of CIDs that had *already* been inserted to the IPFS network via the OP algorithm. Additionally, we included the peers at which the PRs were inserted. The hoarder then used three different lookup methods to gather data about the peers.

First, we called DHT.FindXXProvidersOfCID (Context, CID) which uses the DHT to locate any peers that hold PRs for a specific CID. If this call is successful, it means that the DHT can retrieve the PRs from at least one peer, implying that at least one peer chosen by OP remains in the K closest peers for that CID. This is important, as the peers initially chosen by OP might *not* have been among the K closest peers to the CID. Furthermore, regular peer churn may add or remove peers from the K closest set over time, which is one of the reasons that the PR is stored at K peers. If this call succeeds, it sets the isRetrievable flag for that CID in the database, indicating that a PR for the CID can be located via the DHT.

[2]https://github.com/cortze/ipfs-cid-hoarder

Then, we called DHT.GetProvidersFromPeer (Context, Peer, CID) for each Peer and CID stored in the database, to request the corresponding PRs. This call does *not* search the DHT; it relies on information already in the database. If we cannot connect to that peer, after a few retries, we assume that the peer is not active in the network. Otherwise, the isActive flag in the database will be set to true. If the peer that created the CID (hence, it holds the content) is in the set of providers according to the PRs retrieved, the hasRecords flag is set to true. Note that even though the isActive and hasRecords flags may be set to true, these peers might not be among the K closest ones. Therefore they are not necessarily retrievable via IPFS. This is the difference with the isRetrievable flag set by the previous method, which uses lookups through the DHT.

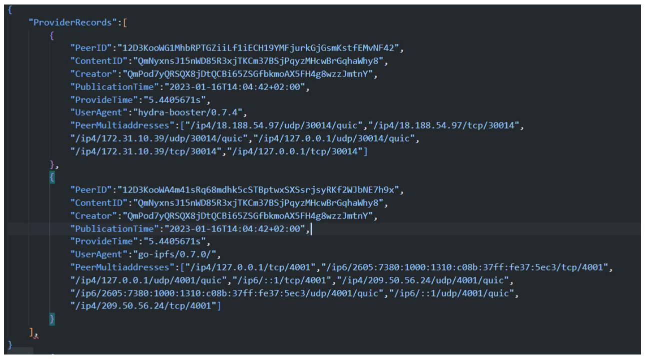
Finally, we called DHT.GetClosestPeersToCid (Context, CID), which used the DHT once again to locate the closest peers to a CID using plain IPFS (not the OP algorithm), and store additional data on the lookups (e.g., hops). We define the *in-degree ratio* for a CID as the number of peers within the K closest set, as returned by this method, that also have the isActive flag set to true, as set by the previous method. Essentially, the in-degree ratio shows how many of the initially chosen peers (by OP) are active and remain among the K closest peers set during the experiments. The reason that we do not instead count the peers with the hasRecords flag set to true, is due to the dial down of Hydra peers: Hydras show up as active but do not respond with PRs.

By combining the results from all these methods, we attempt to answer the following questions:

1) Can we always find at least one peer that returns a PR for a CID? This is the basic requirement from IPFS, and we check this by using the first lookup method.
2) Are the peers chosen by OP reliable? We assess this by probing the peers chosen by OP and checking whether they are alive and store the PRs for a CID for the desired amount of time, using the second lookup method.
3) How well does the OP algorithm operate? We assess this by calculating the *In-degree ratio*, which shows how many of the peers chosen by OP are alive, serve the PR for a CID, and belong to the K closest set. We achieve this via the third lookup method.

IV. Measurement Methodology

To determine the effectiveness of retrieving PRs stored via the OP algorithm, we needed to examine how many of the chosen peers remained online during the study and the number of those peers that responded with the PRs. Furthermore, we needed to assess the results of a DHT search, to verify whether the PRs could be retrieved via the DHT. Finally, it was important to consider the number of PRs that were successfully inserted during the provide process. Note that *only the peers*



```json
{
    "ProviderRecords":[
        {
            "PeerID":"12D3KooWG1MhbRPTGZiiLf1iECH19YMFjurkGjGsmKstfEMvNF42",
            "ContentID":"QmNyxnsJ15nWD85R3xjTKCm37BSjPqyzMHcwBrGqhaWhy8",
            "Creator":"QmPod7yQRSQX8jDtQCBi65ZSGfbkmoAX5FH4g8wzzJmtnY",
            "PublicationTime":"2023-01-16T14:04:42+02:00",
            "ProvideTime":"5.4405671s",
            "UserAgent":"hydra-booster/0.7.4",
            "PeerMultiaddresses":["/ip4/18.188.54.97/udp/30014/quic","/ip4/18.188.54.97/tcp/30014",
            "/ip4/172.31.10.39/udp/30014/quic","/ip4/127.0.0.1/udp/30014/quic",
            "/ip4/172.31.10.39/tcp/30014","/ip4/127.0.0.1/tcp/30014"]
        },
        {
            "PeerID":"12D3KooWA4m41sRq68mdhk5cSTBptwxSXSsrjsyRKf2WJbNE7h9x",
            "ContentID":"QmNyxnsJ15nWD85R3xjTKCm37BSjPqyzMHcwBrGqhaWhy8",
            "Creator":"QmPod7yQRSQX8jDtQCBi65ZSGfbkmoAX5FH4g8wzzJmtnY",
            "PublicationTime":"2023-01-16T14:04:42+02:00",
            "ProvideTime":"5.4405671s",
            "UserAgent":"go-ipfs/0.7.0/",
            "PeerMultiaddresses":["/ip4/127.0.0.1/tcp/4001","/ip6/2605:7380:1000:1310:c08b:37ff:fe37:5ec3/tcp/4001",
            "/ip4/127.0.0.1/udp/4001/quic","/ip6/::1/tcp/4001","/ip4/209.50.56.24/udp/4001/quic",
            "/ip6/2605:7380:1000:1310:c08b:37ff:fe37:5ec3/udp/4001/quic","/ip6/::1/udp/4001/quic",
            "/ip4/209.50.56.24/tcp/4001"]
        }
    ],
}
```

Fig. 2: Example JSON PR created for the hoarder.

that successfully stored the PRs were added to the hoarder's database for further lookups.

To achieve these goals, we created a peer that generated CIDs and published them using the OP algorithm in the actual IPFS network. Before publishing, the routing table of the peer was refreshed, to contain the most recent data. The CIDs were random cryptographic hashes that did not correspond to any meaningful content. After an ADD_PROVIDER success message was received, meaning that a PR was successfully added to a remote peer, some properties of the PR were stored in JSON form, as shown in Figure 2. These properties include:

- The PR's multiaddresses (the addresses of the PR holder).
- The peer's (PR holder) unique identifier.
- The peer's (PR holder) agent type (hydra, go-ipfs etc.).
- The creator (peer that published the content) of the PR.
- The time it took to provide the CID (provide time).
- The publication timestamp of the CID.
- The CID that the PR is saved for.

We then went through these records and probed the chosen peers to check whether the *actual* PRs could be retrieved. Then, a JSON file was created with all the PRs and responsive nodes, to be later loaded into the hoarder's database.

The IPFS-CID-HOARDER tool was configured to create a study lasting *48 hours* with a ping interval of *30 minutes*; the PRs were *not* republished after their initial publication. The 48 hour study accounted for the aliveness period of IPFS at the timeframe when the experiments were conducted: after the 24 hour mark, we should observe that peers are no longer sharing the PR, as they have not been refreshed/republished; recall that the PR retention period has since been changed to 48 hours. The 30 minute ping interval was chosen since peers did not accept constant connection requests, to avoid overflowing the resource handler.

The hoarder gathers results in a PostgreSQL database consisting of the following tables:

- cid_info: basic information about a CID.
- k_closest_peers: the K-closest peers for each CID and for each ping round.
- fetch_results: summary of all the requests done for a given CID on a fetch round.
- peer_info: basic info of a peer chosen as a PR Holder
- ping_results: result of an individual ping of a PR Holder.
- pr_holders: helper table connecting different tables.

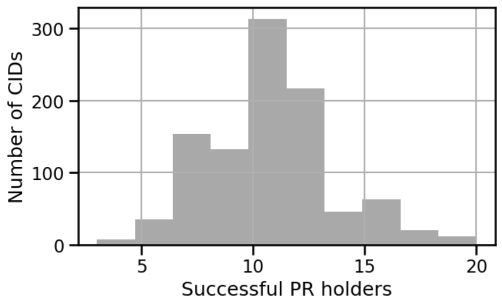


Fig. 3: Distribution of PR holders after publication.

A peer can respond with the PRs in a variety of ways (containing both the multiaddress of the peer and the peer ID, containing only the peer ID, etc.), or may not respond at all. The hoarder accounts for that by also keeping a log file. The PostgreSQL dump file and the log file were transferred to a local machine and visualized using the Jupyter Notebook[3]. This visualization also included an analysis of the log files.

V. RESULTS

We conducted four experiments, but for brevity, we will only present results from one of them, since the results were similar in all of our experiments; the full details can be found in [10]. All experiments were performed in the same period, one after the other, with the same parameters. Note that since these experiments were performed at an earlier period on the real IPFS network, if we were to repeat them now, the numbers would probably be different, due to changes in the IPFS algorithms (e.g., the PR retention period and the Hydra behavior). More generally, the results of *any* experiments on the IPFS network largely depend on the prevailing network conditions. For example, if there are multiple peers not serving the PRs, such as the Hydras, the experiment will be influenced. Our goal was to check whether the OP algorithm satisfies the basic requirements of the plain IPFS algorithm, so assessing all the metrics during the same period was essential.

Most of the graphs we show contain a series of *boxplots*, where the x-axis represents time; each boxplot corresponds to a ping round (30 minutes). The orange line is the median of the metric assessed, the boxes extend between the 1st and 3rd quartile of the distribution (25% to 75% of the results), the whiskers extend to 1.5 times the IQR (inter quartile range), and the dots indicate outliers, that is, results outside the whiskers.

In each experiment, we created a large number of CIDs (1000) that were stored in peers using the OP algorithm. Figure 3 shows the distribution of PRs that were successfully stored during this publication process. We observe that most CIDs were stored at around 10 peers, but there were a few CIDs that were stored at less than 5 peers, while some were stored at more than 15 peers; these distributions have a shape similar to those in the OP paper, but with a lower peak, most likely due to adverse network conditions [3].

Figure 4 shows the average number of these peers (where the PRs were successfully stored) that remained online over

[3]https://jupyter.org/

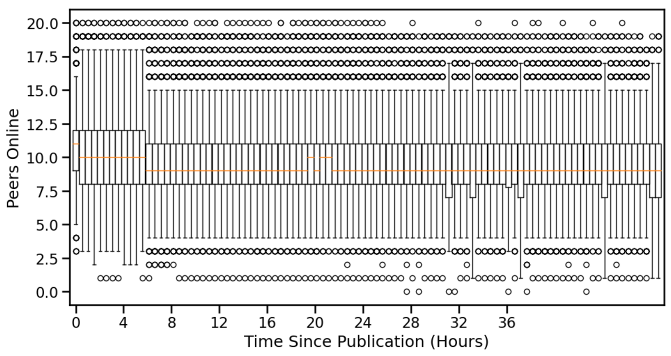


Fig. 4: Average number of online peers.

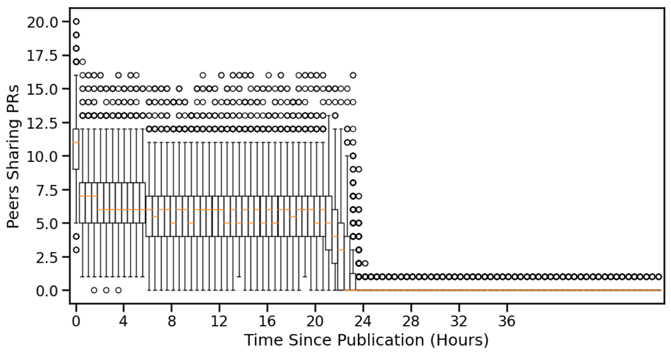


Fig. 5: Average number of peers sharing the PRs.

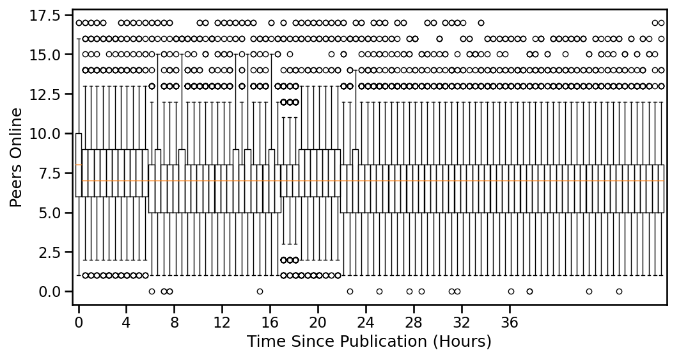


Fig. 6: Average number of online non-Hydra peers.

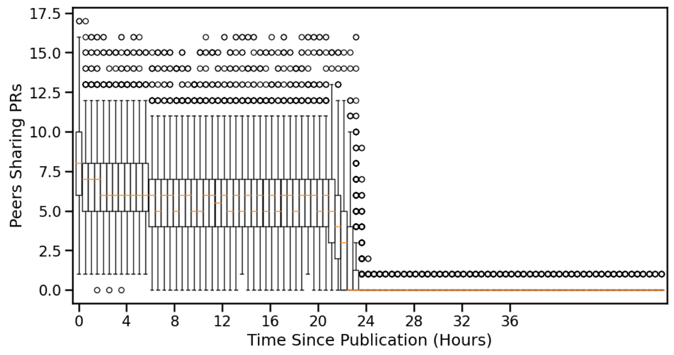


Fig. 7: Average number of non-Hydra peers sharing the PRs.

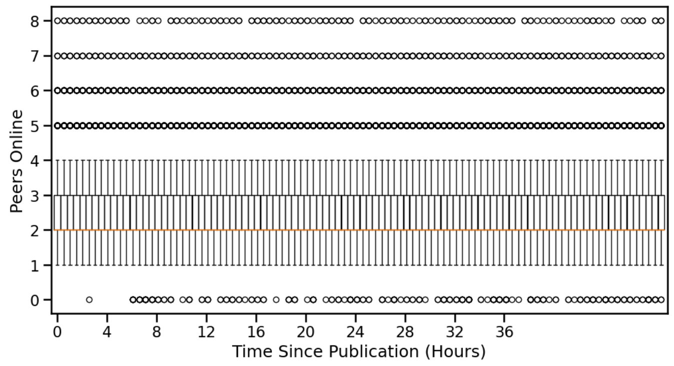


Fig. 8: Average number of online Hydra peers.

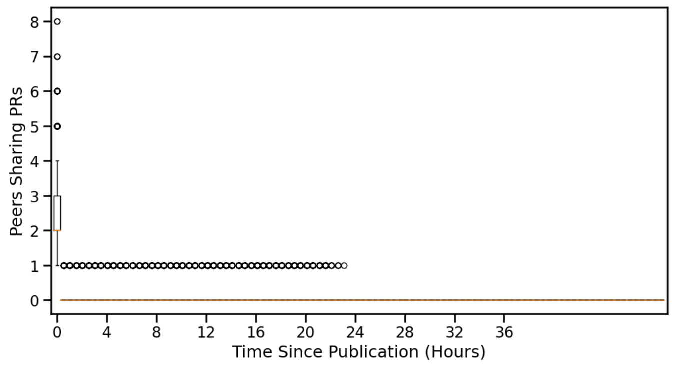


Fig. 9: Average number of Hydra peers sharing the PRs.

time, while Figure 5 shows the average number of these peers actually sharing the PRs. We see that for an average of 9-10 online peers, 5-7 of them will share the PRs, dropping down to 0 around the 24 hour mark, when the PRs expire. There are two reasons for the difference between the two figures: first, Hydra peers do not share the PRs and, second, some peers may not respond due to load limitations.

To assess the influence of Hydra peers on the results due to the dial down process, which meant that they stored PRs but did not respond to queries for them, Figure 6 shows the number of non-Hydra peers online, while Figure 7 shows the number of those peers sharing the PRs. We observe that almost all provider records can be retrieved from non-Hydra peers (compare these figures with Figures 4 and 5). Conversely, Figure 8 shows the average number of Hydra peers that were online, while Figure 9 shows the number of those peers sharing the PRs. Interestingly, one Hydra peer does share some CIDs. This implies that not all Hydras participated in the dial down.

To assess how many of the initially chosen peers remained within the K closest peers over the duration of the study, Figure 10 shows the average *in-degree ratio* across all CIDs over time. We observe that 7-8 peers remain in the K closest peers set, which means that they can be found via a regular DHT lookup. As expected, the *in-degree ratio* is smaller than the number of active peers, since some of them are not in the K closest set, but it is higher than the number of peers sharing the PRs, since some of the chosen peers are Hydras.

We finally turn to the basic requirement for IPFS: that at least one peer discoverable via the DHT shares the PRs. Figure 11 shows that at least one peer shares the PRs with us,

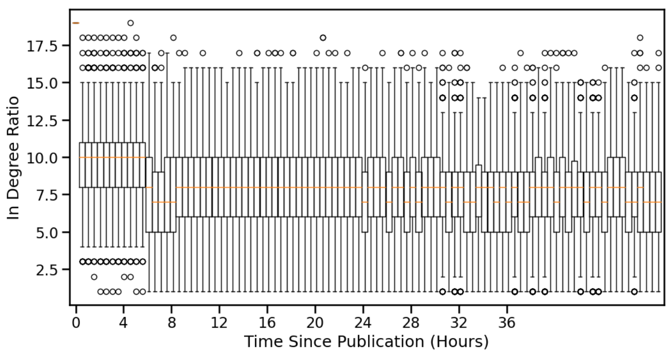


Fig. 10: Average in-degree ratio.

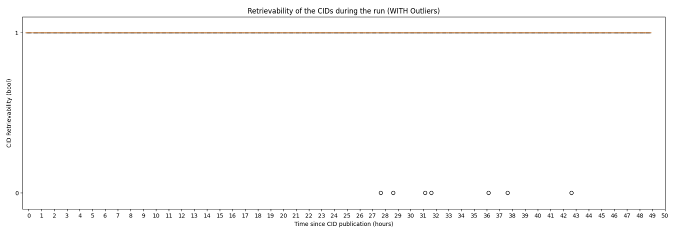


Fig. 11: Retrievability of the PRs over time.

before the 24 hour mark. Note that afterwards, the PRs records are *still shared with us* until the end of the experiment; this is also evident in Figures 5 and 7 where some outlier peers continue responding with the PRs. This indicates that some peers were using the upgraded version of IPFS, where PRs were deleted after 48 hours.

VI. Conclusions and Future Work

The results of our study indicate that the *Optimistic Provide* (OP) algorithm maintains the guarantees of the plain IPFS provide algorithm, while being significantly faster, as shown in in the OP paper [3]. It ensures that the requesting peer can always locate the requested content from at least one peer. Moreover, our analysis indicates that the peers chosen by OP mostly remained in the set of K closest peers, thus making them discoverable by IPFS, ensuring the quick and reliable retrieval of requested data from the network.

Another significant observation was that the Hydra dial down affected the performance of the OP algorithm and the network at large. Hydras were often selected as initial PR holders by the algorithm, and the dial down affected their ability to serve as reliable providers. Therefore, accounting for the impact of Hydra dial down was crucial when implementing and evaluating the performance of the OP algorithm.

It is important to note that if conducted now, the experiments would yield different metrics, as the network has evolved and some parameters have changed. Of course, the study achieved its goal, which was to assess the effectiveness of the OP algorithm. Indeed, the study's findings and insights contributed to the refinement of the OP algorithm, which is now integrated into the IPFS implementation.

An interesting avenue for future work is increasing the number of peers initially selected by the OP algorithm. While the algorithm was capable of retrieving PR holders from the peers, the number of successful peers was lower than with the standard provide algorithm, which averaged around 15. This may enhance the reliability of the network, but it may also increase the time needed to complete store operations, warranting further analysis. Repeating our experiments will also clarify whether the slightly inferior performance of the OP algorithm was due to the prevailing network conditions during the experimentation period, such as peer availability and load, rather than the OP algorithm itself.

Acknowledgements

We would like to thank our colleagues at Probe Lab for their helpful insights and suggestions. In particular, we would like to thank Mikel Cortes and Dennis Trautwein, who provided valuable assistance with the project. Their patience and guidance was most crucial for completing this project.

References

[1] 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 *Proceedings of the ACM SIGCOMM Conference*, 2022, p. 739–752.

[2] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings of the International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[3] D. Trautwein, Y. Wei, Y. Psaras, M. Schubotz, I. Castro, B. Gipp, and G. Tyson, "IPFS in the fast lane: Accelerating record storage with optimistic provide," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2024.

[4] S. A. Crosby and D. S. Wallach, "An analysis of BitTorrent's two Kademlia-based DHTs," Rice University, Tech. Rep., 2007.

[5] S. Wolchok and J. A. Halderman, "Crawling BitTorrent DHTs for fun and profit," in *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, 2010.

[6] O. Saleh and M. Hefeeda, "Modeling and caching of peer-to-peer traffic," in *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, 2006, pp. 249–258.

[7] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, and R. Steinmetz, "Modelling the internet delay space based on geographical locations," in *Proceedings of the Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 301–310.

[8] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed DHT," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2006.

[9] Y. Thomas, N. Fotiou, I. Pittaras, G. Xylomenos, S. Voulgaris, and G. C. Polyzos, "Peer clustering for the InterPlanetary File System," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing*, 2023, pp. 8–14.

[10] F. Bistas, "Retrieval success rate with optimistic provide in IPFS," Athens University of Economics and Business, Tech. Rep., 2023.