Retrieval Success Rate with Optimistic Provide in IPFS

Fotis Bistas

Advisor: George Xylomenos

Athens University of Economics and Business

March 2023

Contents

Abstract

The aim of this report is to examine the retrievability of the provider records that were provided to the IPFS network by using the Optimistic Provide algorithm [1] and assess whether it is comparable to that of the standard IPFS algorithm. This is evaluated by executing a series of experiments with the ipfs-cid-hoarder tool [2] tool, which has been expanded with a new set of features, contained in the following pull request [3].

Chapter 1

Introduction

1.1 The IPFS network and CIDs

The *InterPlanetary File System* (IPFS) is a peer-to-peer protocol for storing and sharing hypermedia in the form of a *distributed file system*. The *IPFS network* is an instantiation of IPFS. IPFS was initially designed by Juan Benet, and is now an open-source project. It aims to make the web faster, safer, and more open by replacing the traditional, centralized model of the web with a decentralized system. Along with that, IPFS aims to offer a distributed alternative to the existing, centralized, content routing systems.

In IPFS, each file and all of the blocks within it are given a unique fingerprint called a *cryptographic hash*; these *cryptographic hashes* are referred to as *CIDs* (content identifiers) and are used to match files and blocks with IPFS nodes. As a result, IPFS stores and retrieves files and blocks based on their content (which is used to calculate the CIDs), rather than their location. This allows for a more resilient and permanent web, as the data are distributed across multiple nodes and can be accessed even if some of the nodes are down. It is important to emphasize that by data, we do not mean the actual content, but rather *links to the content and the peer node that provides it.*

CIDs use the SHA-256 hashing algorithm by default, although many other algorithms are supported. This ensures that any changes in the content will produce different CIDs; therefore, if the same content is added to two different nodes with the same settings, it will product the same CID. Conversely, if the content is changed, its CID changes along with it [4].

1.2 Content provision and provider records

In a distributed peer-to-peer system like the IPFS network, communication between nodes occurs in a more elegant and decentralized manner compared to a centralized network where a central server facilitates communication. It is crucial for peers in a distributed network to have the ability to locate each other, as new nodes join and existing nodes leave the system. To facilitate the exchange of data between nodes, IPFS utilizes a *Distributed Hash Table* (DHT). The DHT functions as both a catalog and a navigation system for the IPFS nodes, allowing them to map keys to values and, eventually, nodes.

Type	Purpose	Used by
Provider records	Map a data identifier (i.e., a multihash) to a peer that has advertised that they have that content and are willing to provide it to you.	- IPFS to find content - IPNS over PubSub to find other members of the pubsub *topic*.
IPNS records	Map an IPNS key (i.e., the hash of a public key) to an IPNS record (i.e., a signed and versioned pointer to a path like `/ipfs/bafyxyz...`)	- IPNS
Peer records	Map a peerID to a set of multiaddresses at which the peer may be reached	- IPFS when we know of a peer with content, but do not know its address. - Manual connections (e.g., `ipfs swarm connect /p2p/Qmxyz...`)

Figure 1.1: The three key-values pairings stored in the IPFS DHT. The IPNS values are not relevant to this study [4].

IPFS uses an implementation of the Kademlia DHT, which ensures:

- The unique identification of peers, using an *address space* of $[0, 2^{256} - 1]$.
- A peer ordering in the address space from smallest to largest. This is achieved using the SHA-256 hash of the peer's ID and interpreting it as an integer between $[0, 2^{256} - 1]$.
- A *projection* that assists in calculating a position in the address space where a *record key* should be stored at. This is done using the SHA-256 hash of the record key.

To help locate peer nodes (and, thus, record keys), each peer maintains a *skip-list* with the peers that are at a distance of *approximately* 1, 2, 4, 8, ..., hops away (inversely common prefix); since the address space is much larger than the number of peers, there is no guarantee that a peer will exist at these *exact* distances. The distance/hops is calculated as a XOR operation between the addresses of two peers. At the same time, due to the *node churn* in the IPFS network, that is, the constant addition and removal of peer nodes, we cannot rely on a single pointer to maintain routing; instead, a peer stores K links for each distance, with $K = 20$.

For example, for the distance of 128, it stores $K = 20$ links to nodes that lie somewhere in the 65 $(2^6 + 1)$ to 128 range. These are called *k-buckets*. As the address space is 2^{256}, each node maintains 256 *k-buckets* [4]. These *K-buckets* are the IPFS *routing table* of each node, containing 256 rows, each corresponding to one *K-bucket.* Peers are inserted into this list when a peer connects to another peer. Deciding which nodes should be inserted to the routing table is left up to the Kademlia implementation.

When a peer desires to provide content into the network, it does a lookup into the DHT to retrieve the K closest peers, measured in XOR distance (this is the Kademlia closeness metric), to the CID of the content. In order for the peer to successfully advertise that it has the content, it must add a *Provider Record* (PR) to these peers. The PR is essentially a pointer to the peer that provides the content.

To add the PR, we use the *ADD_PROVIDER* RPC of the Kademlia DHT. This notifies the remote peer that it is in the set of closest peers to the content provided. If the PR is actually 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 which is crucial for this study, is that they are only kept alive in the IPFS network for 24 hours. This means that after the 24 hour mark, the peers will *no longer share* the PRs. If the provider wants to keep the content accessible, it is suggested that it republishes the PRevery 12 hours, adapting the content to the rapidly changing network conditions [5]. In this manner, peers who are no longer the closest ones to the CID, automatically drop the PR, while the now closest peers will start storing it.

1.3 Hydra boosters

To accelerate content routing, the IPFS network may include some *Hydra nodes.* Hydras are essentially performing a non-malicious Sybil attack on the network, using multiple peer IDs distributed in the address space [6]; each of these peer IDs are one of the Hydra's heads. Hydras influence the way the network behaves; due to their multiple IDs, nodes will likely contact them to complete IPFS functionalities [7].

Hydras receive *ADD_PROVIDER* RPCs from nodes that want to advertise content in the network and they proceed to store them in a large database. The Hydra heads can then access the database, when a *GET_PROVIDERS* RPC is received and serve the RPC; essentially, a Hydra node can make large jumps in the address space due to its multiple Peer IDs. If a Hydra does not contain the PR, it proactively searches the network in order to store it [7].

Currently Hydras have been dialed down in the network, to determine their impact. This implies that the Hydras' database does not serve any requests. The initial provide operation may insert a PR in a Hydra node, but as a result of the dial down, the Hydra nodes will not respond back with that PR. As a result, it is expected that the total online peer average will be higher than the received PR average.

1.4 Optimistic Provide

The process of providing content in the IPFS network is notoriously slow compared to discovering content, since providing content requires locating *all* K peers, while discovering content requires locating just *one* of them. Discovering content in the network in most cases takes about 1.5 s, while providing content can take sometimes up to 60 s. This in turn makes content provision a non-user friendly experience [8].

Locating all the closest peers to a CID is further complicated by the fact that due to *node churn*, some of the peers in the node's routing tables may have left the network, leading to timeouts. Also, peers need to limit the amount of resources they assign to each connection, thus some connection attempts might fail due to the fact that remote peers have reached their resource limit. Additionally, since the peers are sparsely distributed in the ID space, peers close to the XOR distance or those that share high number of common prefix bits, of the peer's own ID will likely not be as available as others, simply because there are not that many stable peers to begin with in close distances [5]. A helpful visualization is given in Figure 1.2.

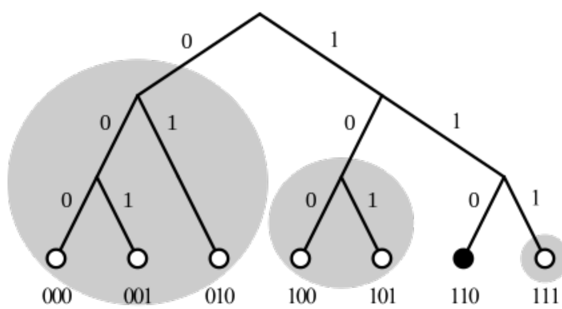
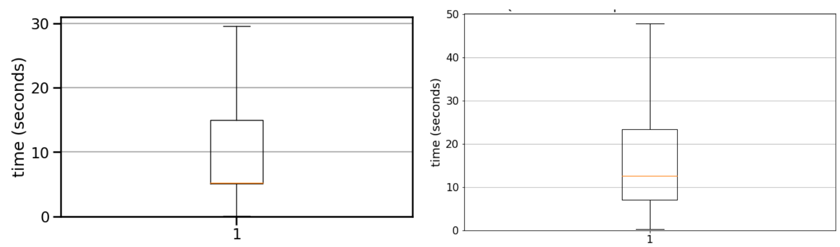


Figure 1.2: A visualization of the available peers in the Kademlia address space. Each grey circle is its own K-bucket and the original node is the black one [9].

Optimistic provide is an algorithm that aims to decrease the time that it takes for content to be published in the IPFS network. Optimistic provide achieves that using an "optimistic" approach when publishing content. Using an *estimate* of the network size and calculating the XOR distance of the CID we want to store to the identifier of a peer that we are already in contact with, we can guess if that peer is appropriate for storing a PR, without performing a detailed search. The goal here is to exploit as far as possible peers that we know about, instead of trying to locate new ones.

To explain how this works, consider a candidate Peer ID P, a CID to store C and an IPFS network with size N. Taking the *XOR* distance of the Peer ID and the CID, normalizing it to the $[0, 1]$ range 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 provider record at peer P, as with high certainty the peer is truly in the list of the K closest peers. The *normed XOR distance* is produced by dividing a distance with the maximum value in the address space, which is $2^{256} - 1$. To understand this, one can think of the *normed XOR distance* as a *percentage*. For example, if the normed XOR distance is $\mu = ||P - C|| = 0.1\%$ and the *network size* N is 7000, following the above formula it is safe to say that around 7 peers are closer to C than P [1]. More in depth analysis of the theoretical modeling can be found in [10]. As Figure 1.3 shows, this can significantly decrease the time to provide content; the maximum time of optimistic provide is around 30 seconds while the max time of the standard provide is around 50 seconds.



(a) Optimistic provide.

(b) Normal provide.

Figure 1.3: Distribution of content provisioning time.

1.5 The Ipfs-cid-hoarder tool

In order to gather metrics about PRs, a tool that pings the providers and analyzes their responses is needed. The *IPFS-cid-hoarder* tool stores relevant information about the content in a database and then proceeds to ping the peers holding each PR. During this process, metrics about the responses (does the peer have a PR, what is the content of the PR, which user agent is used by the peer, etc.) are added to the database for further analysis [11]. The hoarder uses multiple lookup methods in order to gather metrics about the providers. Specifically, it calls concurrently:

- DHT.LookupForProviders(context, CID): this checks if the network can still route the requesting peer to the peer hosting the content, using a full DHT walk. If successful, it assigns a true value to the isRetrievable flag in the database. This flag is used to create the **at least one peer provides** graphs.

```
//returns the providers for a specific CID using a DHT walk
providers, err := p.host.DHT.LookupForProviders(ctxT, c.CID)
// iter through the providers to see if it matches with the
    host's peerID
for _, paddrs := range providers {
    if paddrs.ID == c.Creator {
        isRetrievable = true
    }
}
```

- hoarder.PingPRHolder(individual PR holder): after connecting to a peer, this retrieves all the PRs from the peer. This is not a libp2p library call, but rather a custom function which performs the libp2p call DHT.GetProvidersFromPeer(context, peerID, CID), which allows one to request the PRs of a single peer for a given CID. If we successfully connect to the peer, the isActive flag in the database will be set to true; this flag is used for the **active peer graphs**. If the creator peer of the CID is in the set of providers, the hasRecords flag is set to true; this is used to create the **retrievability graphs**. Note that this scheme does not perform a DHT walk: the individual PR holders are pinged using their peer IDs and the DHT table is not searched to find the PR holders.

```
// if the connection was successful, request whether it has
    the records or not
isActive=true
connError = p2p.NoConnError
provs, _, err := pinger.host.DHT.GetProvidersFromPeer(pinger.
    ctx, pAddr.ID, c.CID.Hash())
// iter through the providers to see if it matches with the
    host's peerID
for _, paddrs := range provs {
    if paddrs.ID == c.Creator {
        hasRecords = true
    }
}
```

- DHT.GetClosestPeers(context, CID): this is the full DHT walk (using the standard lookup scheme) to retrieve the closest peers and, while retrieving them, store some useful additional data (hops etc.). This aids in calculating the very important **In-degree ratio**, which is the number of peers with the isActive flag set to true, that are also in the set of the **K-closest peers** throughout the study:

```
//returns the closest peers currently available to a given CID
closestPeers, lookupMetrics, err := p.host.DHT.GetClosestPeers
    (ctxT, string(c.CID.Hash()))
for _, peer := range closestPeers {
    cidFetchRes.AddClosestPeer(peer)
}
```

To summarize and point out some additional details, the above are used for the following use cases:

- The results of DHT.LookupForProviders are used to ensure that, using a full DHT walk we can retrieve the PRs from at least one peer, implying that at least one peer chosen by optimistic provide is in the K-closest peers set. This is important, as the peers initially chosen might not be in the K-closest peers set, or some new peers might join the network that are closer to the ones that we initially chose. The graphs created by this method (for example, Figure 3.9), exploit the isRetrievable flag.

- The results of the hoarder.PingPRHolder, which are depicted in the retrievability graphs (for example, Figure 3.3) and the online graphs (for example, Figure 3.2), are *not based on the full DHT walk*. This means that even though the hasRecords/isActive flag may be set to true, these peers might not be in the K-closest peers set, so the content cannot be retrieved using a normal DHT walk. To assess whether Optimistic Provide has chosen "good" peers (in the K-closest peers set), the in In-degree ratio is compared to the online average. To ensure that the peers chosen are indeed in the K-closest peers set and the content can be retrieved using a walk, the isRetrievable flag is used to create graphs like Figure 3.10.

- The results of the DHT.GetClosestPeers(context, CID), which is the full DHT walk, are used to create the In-degree ratio graphs (see Figure 3.4) that describe how many of the initially chosen peers remain in the K-closest peers set during the experiments.

1.6 Contributions to Ipfs-cid-hoarder

The tool was initially created to publish purely random CIDs, proceed to ping them and extract information about how the IPFS network treats PRs as a whole. The functionality that was needed to complete this study was to add CIDs to the hoarder that had already been inserted to the IPFS network via the Optimistic Provide algorithm; then, the hoarder continued with the operations mentioned above.

To achieve, this I created two approaches: a HTTP server approach and a JSON file approach.

- The JSON file approach reads the PRs through a JSON file, parses them and inserts them into the database to be further analyzed by the hoarder. The JSON file approach is only used for small sample sizes.

- The HTTP approach publishes the PRs to an HTTP server (running locally), retrieves them from the server, parses them and then inserts into the database to be further analyzed by the hoarder. This is used for large sample sizes.

The goal was to create a general toolkit that can be used for analyzing network utilities/algorithms, that concern PR retrievability, even for future usage. The above methods can indeed be used for future work, but they might need some tuning to be as descriptive and functional as the initial method.

1.7 Goals of the report

This report attempts to answer the following questions.

1. Does the optimistic provide algorithm work properly? We assess this by by checking whether the peers chosen are appropriate peers for the PRs. This can be quantified by pinging them, checking if they actually keep the PRs for a desirable amount of time and then checking the In-degree ratio.

2. What is the percentage of PRs that can be retrieved from the network, compared to the online percentage of the PR holders? This establishes whether the content is retrievable and whether it offers the same level of retrievability as the standard IPFS provide algorithm.

3. Can we always find at least one peer that returns the PRs during the study? This is the basic requirement from IPFS.

Chapter 2

Methodology

We have created a libp2p node which generates CIDs and publishes them using the Optimistic Provide algorithm in the actual IPFS network. Before publishing, the routing table of the libp2p node is refreshed, so as to contain the most recent data from the network. The CIDs are random cryptographic hashes that do not contain any meaningful content. After an ADD_PROVIDER success message is received, which means a PR was successfully added to a remote peer, some properties of the PR are stored in JSON form, as shown in Figure 2.1. These properties include:

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

```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"]
        }
    ],
}
```


Figure 2.1: Example JSON PR created for the hoarder.

These JSON records, which are inserted in the hoarder's database, are not actual PR; they are just a helpful way of representing them. We then extract the peer IDs from the database and ping them to check whether the *actual* PRs can be retrieved. This is achieved in one of two ways:

- If the sample size is small, a JSON file is created with all the records, and the file is later inserted into the hoarder. This means that we cannot start pinging any peers until all PRs are inserted. As insertion takes time, this means that a long time may pass between inserting the first PRs and pinging the peers storing them.

- If the sample size is large, creating a single file would cost pinging rounds and CID "aliveness" in the network. Instead, the CIDs are published in one machine, sent through HTTP to another machine and inserted into the hoarder. This avoids presenting huge amounts of traffic to the local network by publishing and pinging at the same time.

The ipfs-cid-hoarder is configured by the command line tool to create a study lasting *48 hours* with a ping interval of *30 minutes*. Note that the PRs *are not* republished after their initial publication. The 48 hour study accounts for the aliveness functionality of IPFS: after the 24 hour mark, we should observe that peers are not sharing the PR with us, as they have not been refreshed/republished. The 30 minute ping interval was chosen since peers do not accept constant connection requests.

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

- cid_info: contains the basic information about a CID.

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

The peer can respond with the PRs in a variety of ways, containing the multi-address of the peer + the peer ID, containing only the peer ID or not respond at all. The hoarder accounts for that by also keeping a log file.

The PostgreSQL dump file and the log file are sent to a local machine using SCP. Then, the results are visualized using jupyter notebook, by reproducing the database in PG_Admin and querying it. This visualization also includes the analysis of the log files.

Chapter 3

Results

To determine the effectiveness of retrieving PRs in the IPFS network, we need to examine the number of peers that were online during the study and the number of those peers that responded with the PRs. If the online average is similar or close to the retrievability average, it indicates that our goal has been achieved. Furthermore, we must analyze the results of the DHT walk and check whether the PRs can be retrieved using the DHT. Additionally, it is important to consider the number of peers that were successfully saved during the provide process, or how many "put provider" RPCs were successful. Note that only the successful PRs are inserted into the hoarder. This implies that the online average should be around the successful PR holder average.

We performed a total of eight experiments. The first four experiments were designed as pairs: the first and second experiment published 600 CIDs at different times, while the third and fourth published 1000 CIDs at different times, in all cases using the JSON file method to control the experiment. All these experiments were performed in the same period, one after the other.

The next four experiments were performed at a later period (with different IPFS network conditions), increasing the number of CIDs and moving from the JSON file to the HTTP server method. The fifth experiment published 1500 CIDs using the JSON file method, the sixth and seventh experiment published 2000 CIDs, first with the JSON file and then with the HTTP server method, and the eighth experiment published 3500 CIDs with the HTTP server method. We'll only display the JSON file ones.

The graphs are a series of *boxplots*, with each boxplot corresponding to a ping round (30m). The yellow line is the median, 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 distance), and the dots indicate results outside the whiskers.

3.1 First experiment: 600 CIDs/JSON

Figure 3.1 shows the distribution of successful PR holders during the publication process. The largest number of CIDs were stored in 10 nodes, but there were a few CIDs that were stored at less than 5 nodes.

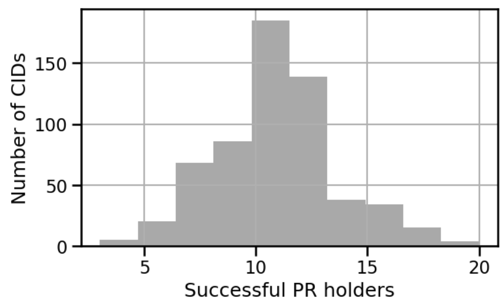


Figure 3.1: Distribution of successful PR holders during publication.

Figure 3.2 shows the average number of online peers over time, while Figure 3.3 shows the average number of peers sharing the PRs. Both these graphs are based on directly pinging the peers known to hold the PRs by the hoarder. To reiterate, this is *not* part of a DHT walk, but rather a direct ping from the database. We see that for an average of 8-10 online peers, 5-7.5 of them will share the PRs, dropping down to 0 around the 24h mark, when the PRs expire.

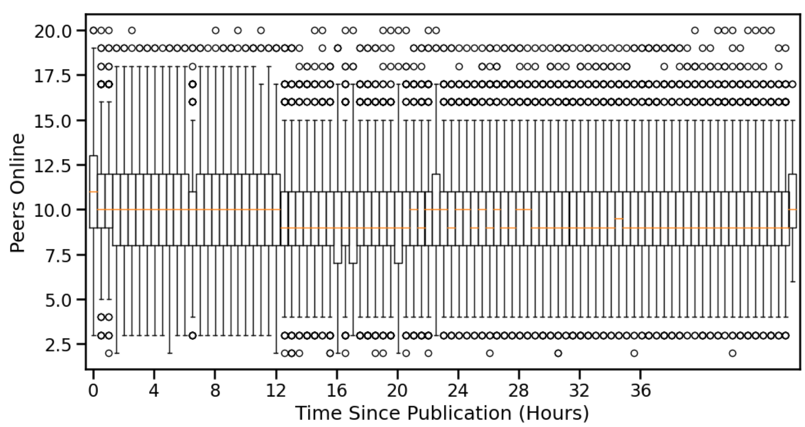


Figure 3.2: Average number of online peers.

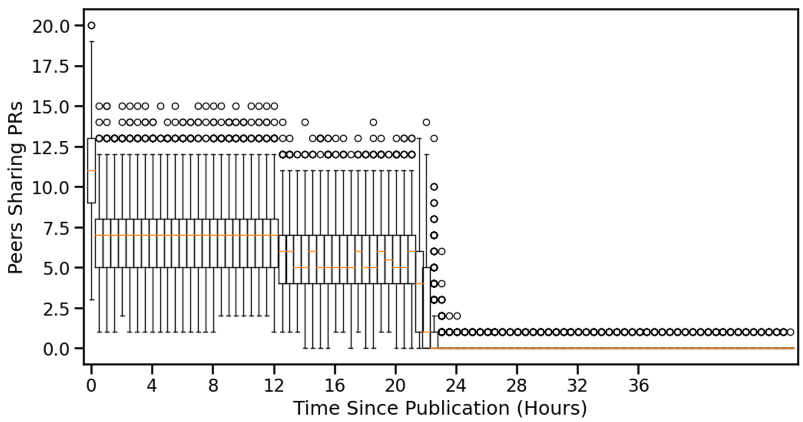


Figure 3.3: Average number of peers sharing the PRs.

To assess how many of these peers remain in the K-closest peers set over the experiment, Figure 3.4 shows the average In-degree ratio per CID, that is, the number of initially chosen peers by the Optimistic Provide process that are among the K closest to the CID. We can observe that 7-10 of those peers remain in the K-closest peers set.

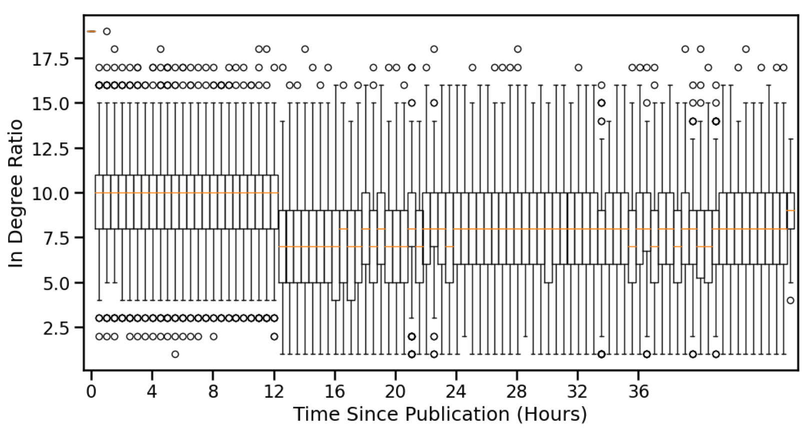


Figure 3.4: Average in-degree ratio.

Figure 3.5 shows the number of non-Hydra peers online, while Figure 3.6 shows the number of those peers sharing the PRs. We can see that we can retrieve almost all provider records from non-Hydra peers.

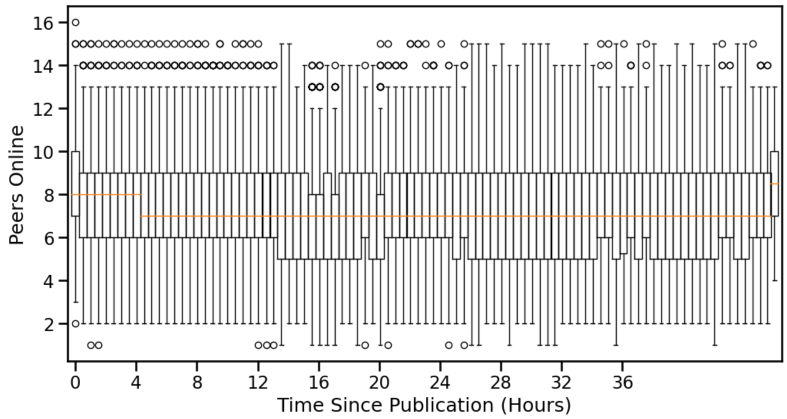


Figure 3.5: Average number of online non-Hydra peers.

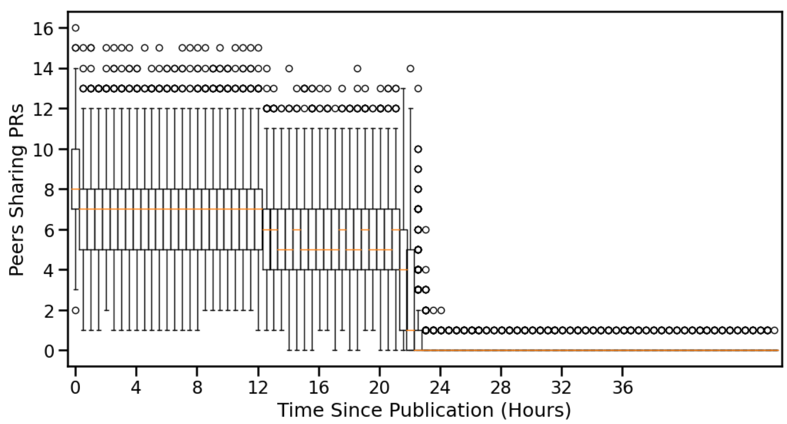


Figure 3.6: Average number of non-Hydra peers sharing the PRs.

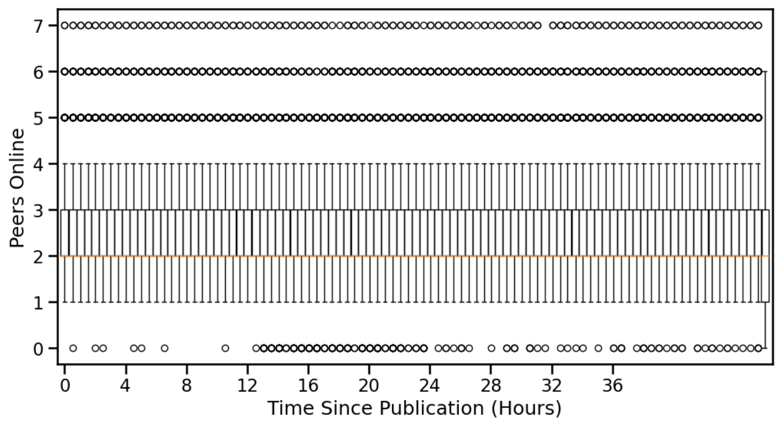


Figure 3.7: Average number of online Hydra peers.

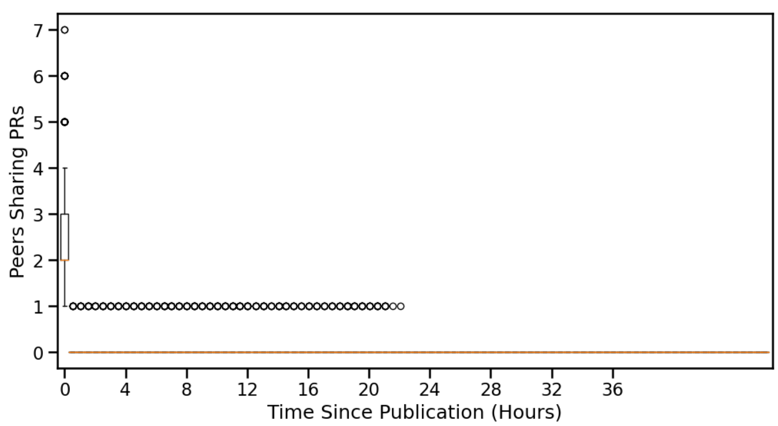


Figure 3.8: Average number of Hydra peers sharing the PRs.

Figure 3.7 shows the average number of Hydra peers online, while Figure 3.8 shows the number of those peers sharing the PRs. We can see that Hydras did not share the PRs with us, which causes the total retrievability average to drop. The sharing during the first hour is attributed to the hoarder initializing data.

We now turn to the basic requirement for IPFS, that at least one peer shares the PRs, before they expire. This is achieved by performing a full DHT walk, which is what an actual retrieval request would do. Figure 3.9 shows that at least one peer shares the PRs with us, during the study before the 24h mark. Note that after the 24h mark the provider records are *still shared with us* until even the 49h mark.

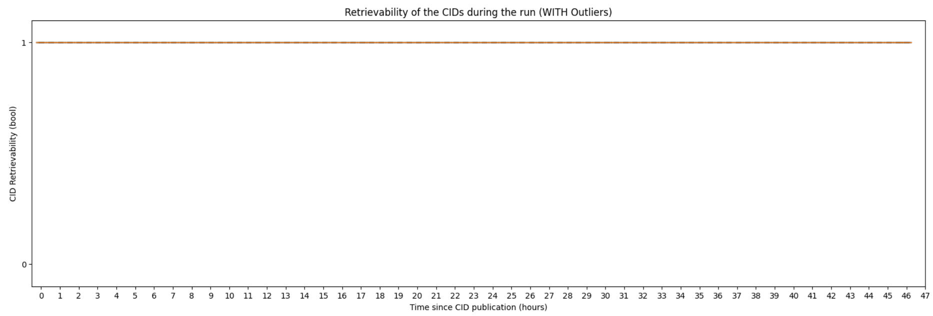


Figure 3.9: Liveness of the PRs.

Figure 3.10 shows the average number of peers responding to a DHT lookup. Note that even after the 24h mark (here the pings are counted, so around 48 pings) some peers might still respond back with the PRs up until 92 pings.

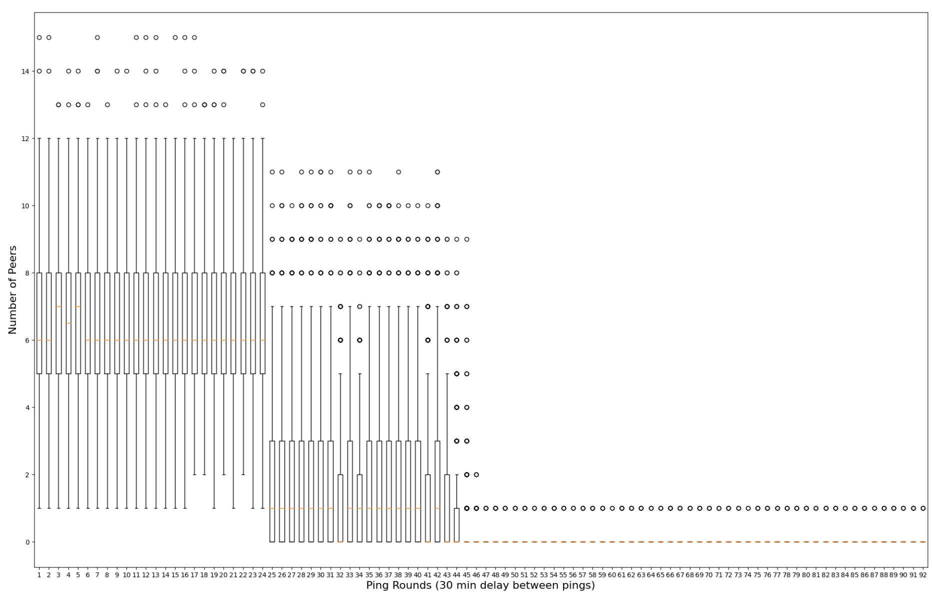


Figure 3.10: Average number of peers responding to a DHT lookup.

Finally, Figure 3.11 shows the results of the DHT lookup. As mentioned before the peers can respond in a variety of ways. It is important to point out that even when the peers hold a PR, they may not respond (e.g., due to the resource manager blocking us out). Again, we can see that even after the 24h mark (here the pings are counted so around 48 pings) some peers might still respond back. In most cases, we either receive only the Peer ID or no response at all.

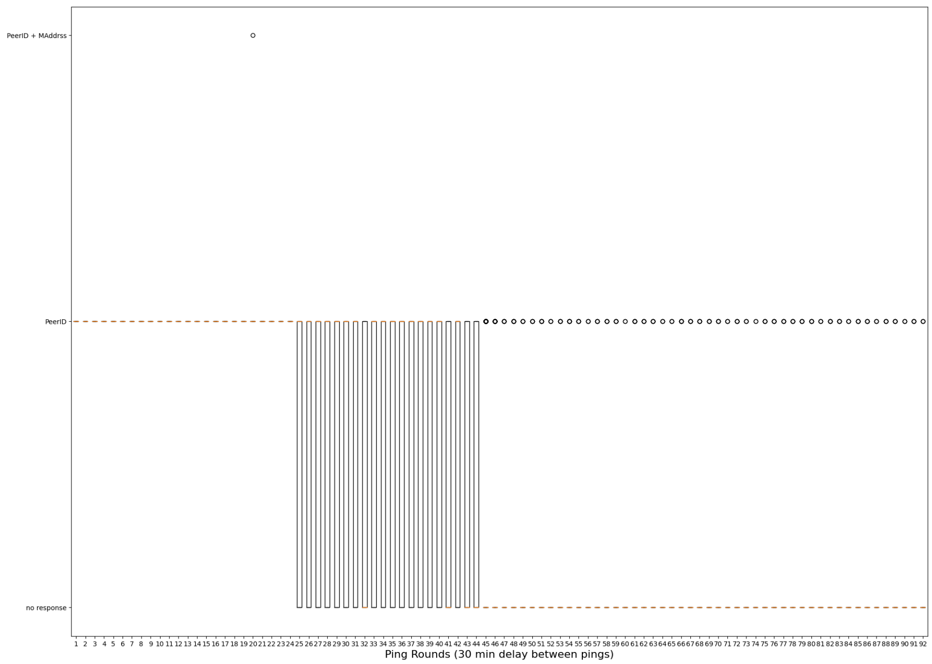


Figure 3.11: Results of a DHT lookup.

3.2 Second experiment: 600 CIDs/JSON

This experiment is a repetition of the first one, at a slightly later date. Figure 3.12 shows the distribution of successful PR holders during the publication process. It can be observed that most of the successful PR holder are in the 8-12 range, but there were a few CIDs that were stored at less than 5 nodes.

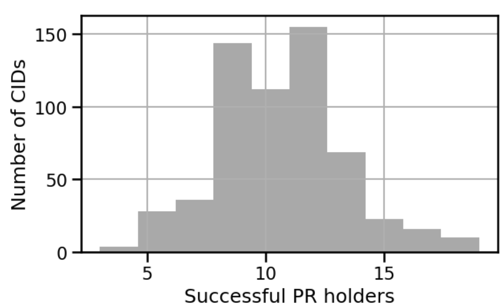


Figure 3.12: Distribution of successful PR holders during publication.

Figure 3.13 shows the average number of online peers over time, while Figure 3.14 shows the average number of peers sharing the PRs. We see that for an average of 8-10 online peers, 5-6 of them will share the PRs, dropping down to 0 around the 24h mark, when the PRs expire; this is slightly lower than in the first experiment.

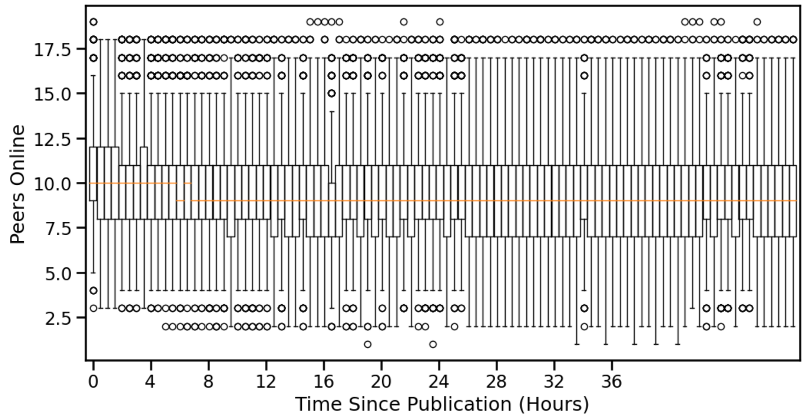


Figure 3.13: Average number of online peers.

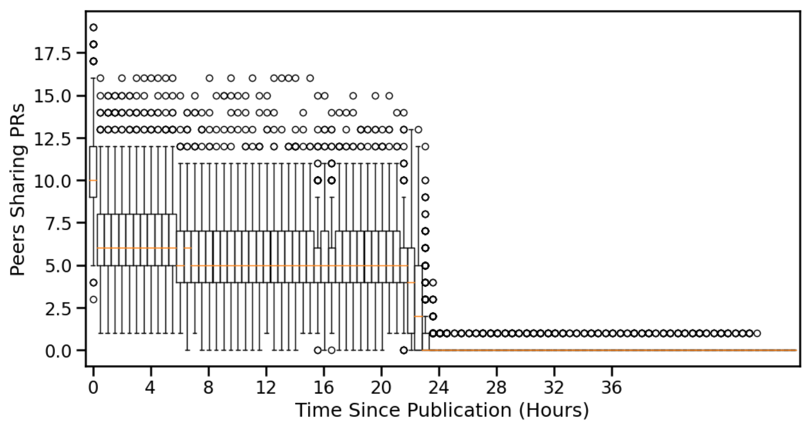


Figure 3.14: Average number of peers sharing the PRs.

To assess how many of these peers remain in the K-closest peers set during the study, Figure 3.15 shows the average (n-degree ratio per CID. We can observe that 6-10 peers remain in the K-closest peers set, similar to the first experiment.

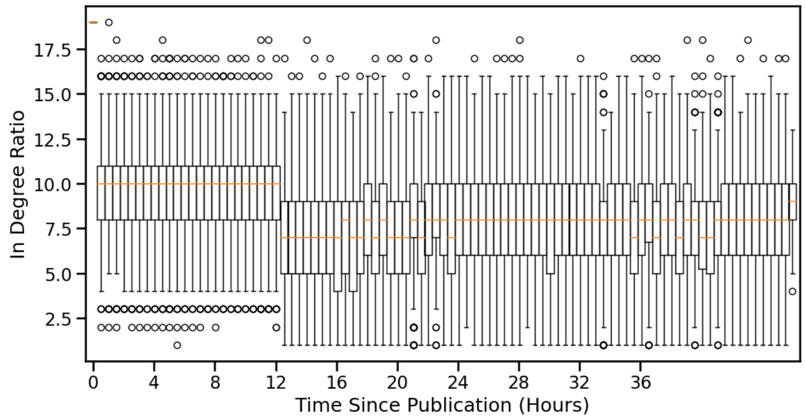


Figure 3.15: Average in-degree ratio.

Figure 3.16 shows the number of non-Hydra peers online, while Figure 3.17 shows the number of those peers sharing the PRs. Again, we can see that we can retrieve almost all provider records from non-Hydra peers.

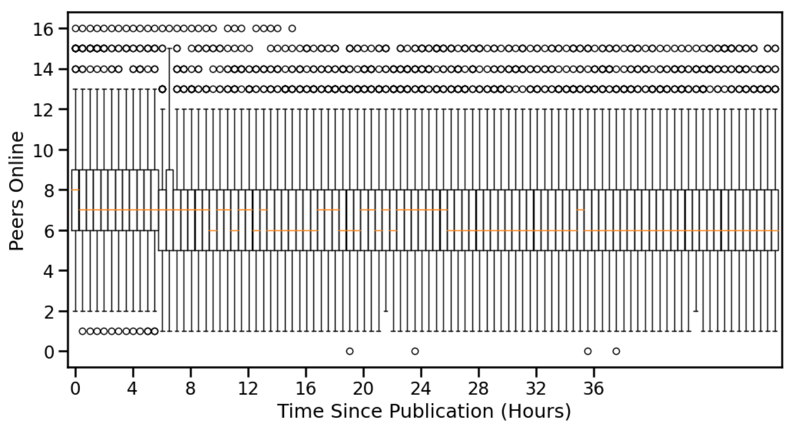


Figure 3.16: Average number of online non-Hydra peers.

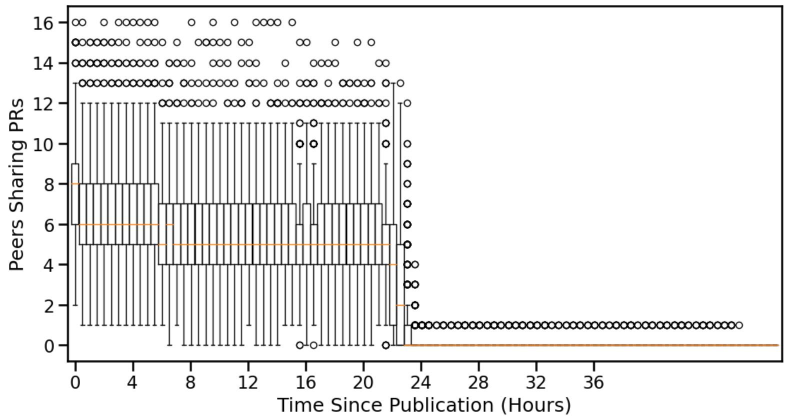


Figure 3.17: Average number of non-Hydra peers sharing the PRs.

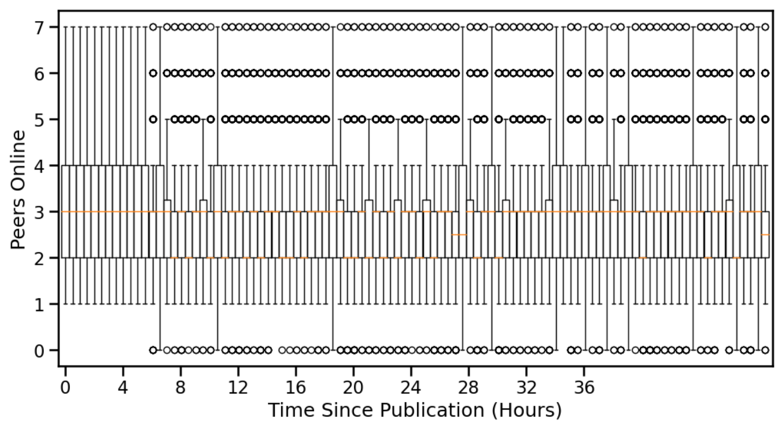


Figure 3.18: Average number of online Hydra peers.

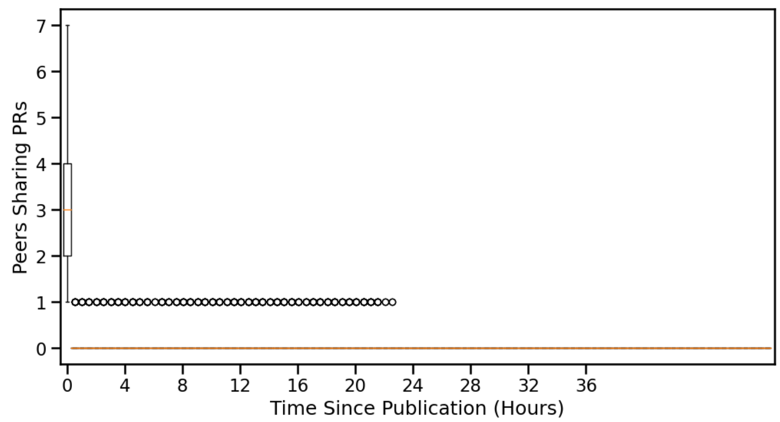


Figure 3.19: Average number of Hydra peers sharing the PRs.

Figure 3.18 shows the average number of Hydra peers online, while Figure 3.19 shows the number of those peers sharing the PRs. We can see that Hydras did not share the PRs with us. Again, sharing during the first hour is attributed to the hoarder initializing data.

We now turn to the basic requirement for IPFS, that at least one peer shares the PRs, before they expire. Figure 3.20 shows that at least one peer shares the PRs with us, during the study before the 24h mark. Note that after the 24h mark the provider records are *still shared with us* until even the 49h mark.

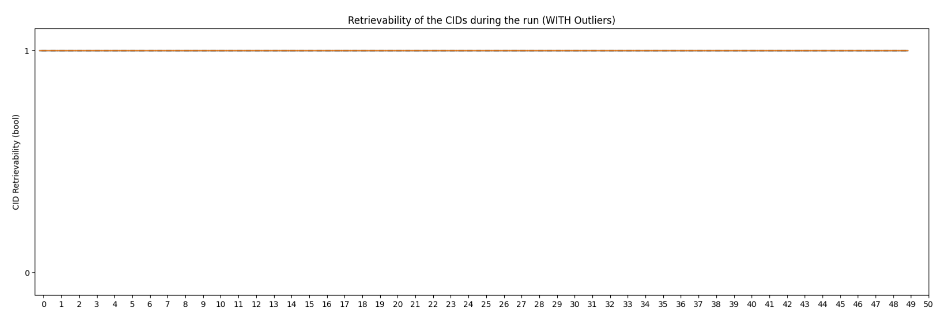


Figure 3.20: Liveness of the PRs.

Figure 3.21 shows the average number of peers responding to a DHT lookup. Note that even after the 24h mark (here the pings are counted, so around 48 pings) some peers might still respond back with the PRs up until 97 pings.

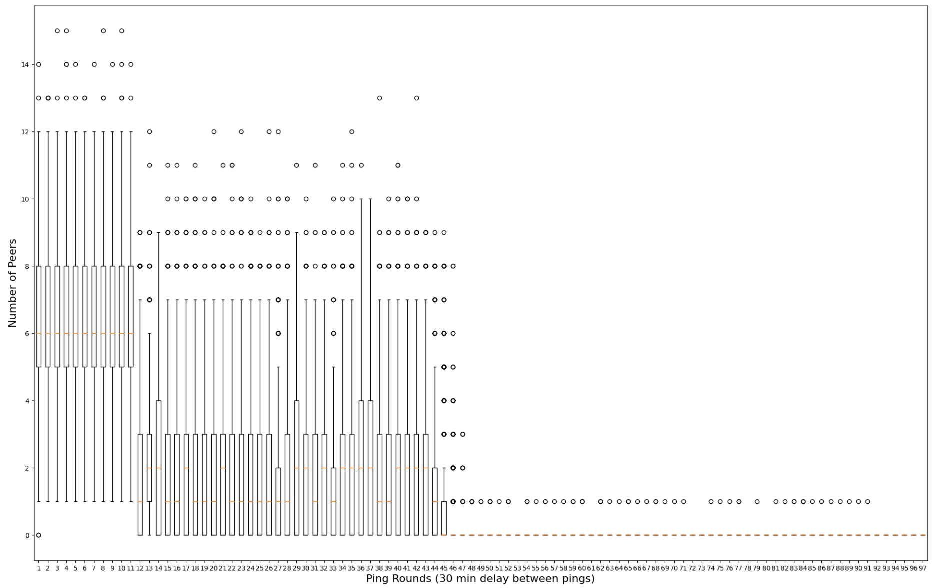


Figure 3.21: Average number of peers responding to a DHT lookup.

Finally, Figure 3.22 shows the results of the DHT lookup. Again, we can see that even after the 24h mark (here the pings are counted so around 48 pings) some peers might still respond back. In most cases, we either receive only the Peer ID or no response at all.

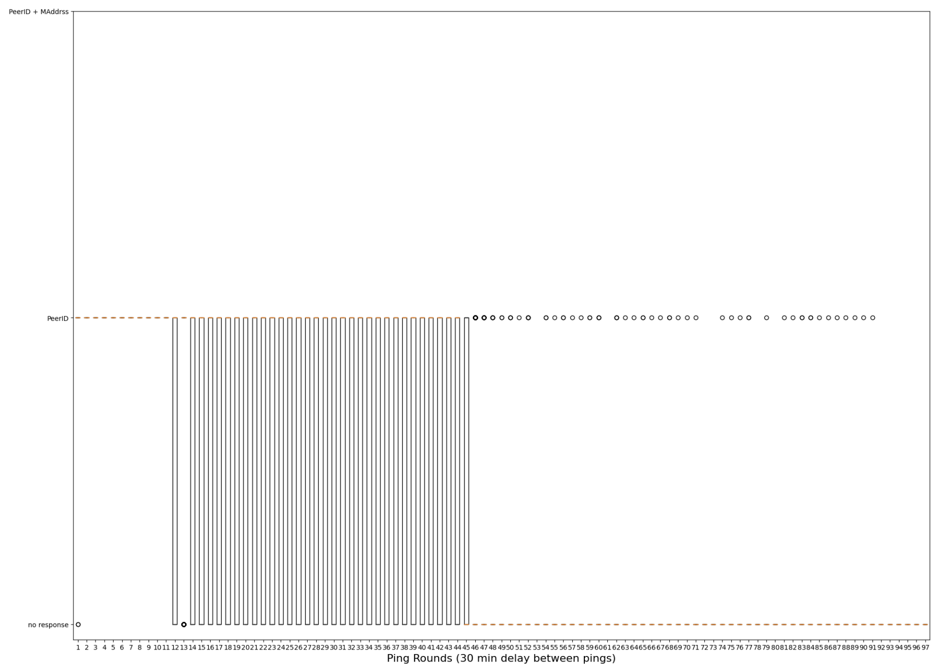


Figure 3.22: Results of a DHT lookup.

3.3 Third experiment: 1000 CIDs/JSON

The third experiment uses the same setup as the previous two, but with a higher number of CIDs (1000 instead of 600). Figure 3.23 shows the distribution of successful PR holders during the publication process. It can be observed that most of the successful PR holders are around 10, but there were a few CIDs that were stored at less than 5 nodes.

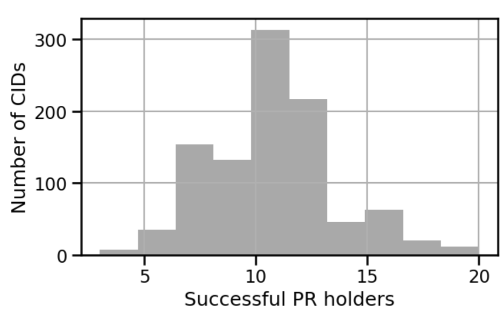


Figure 3.23: Distribution of successful PR holders during publication.

Figure 3.24 shows the average number of online peers over time, while Figure 3.25 shows the average number of peers 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 24h mark, when the PRs expire.

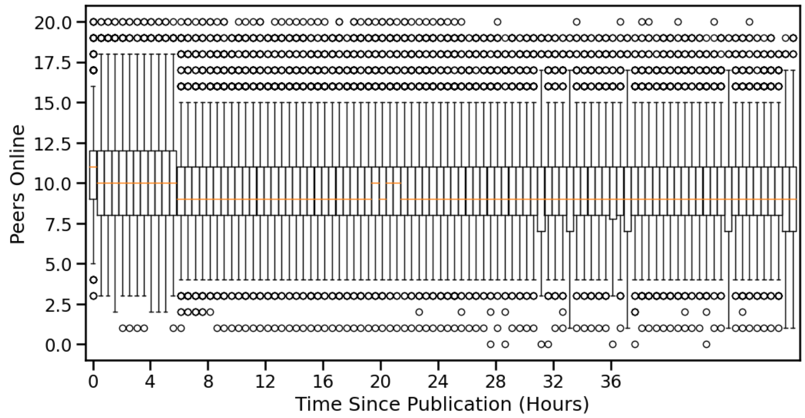


Figure 3.24: Average number of online peers.

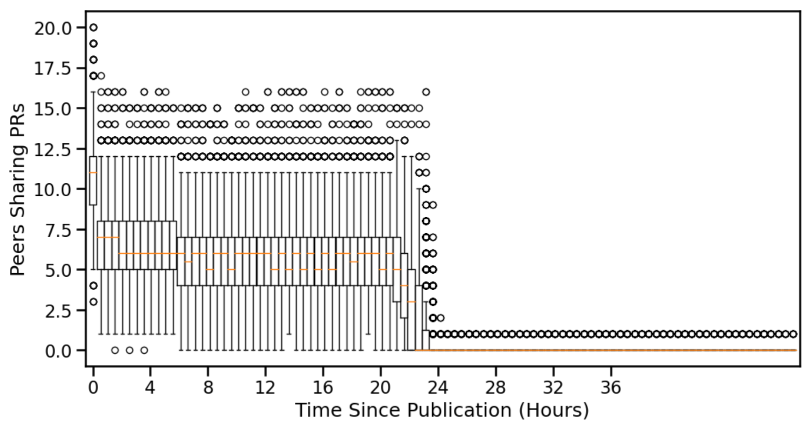


Figure 3.25: Average number of peers sharing the PRs.

To assess how many of these peers remain in the K-closest peers over the duration of the study, Figure 3.26 shows the average In-degree ratio per CID. We can observe that 7-8 peers remain in the K-closest peers set.

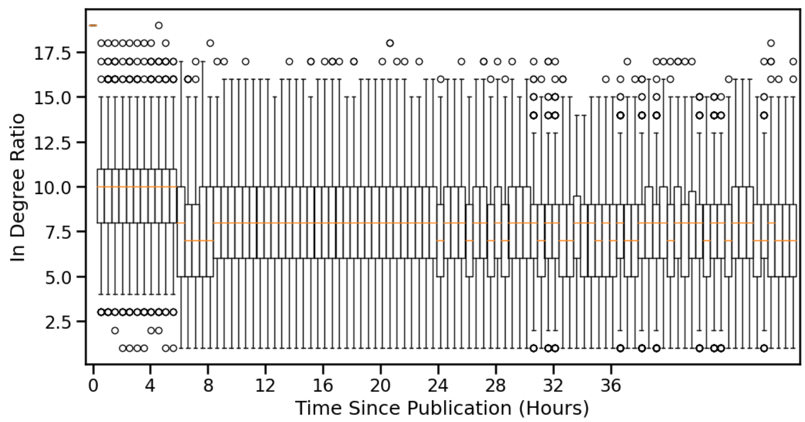


Figure 3.26: Average in-degree ratio.

Figure 3.27 shows the number of non-Hydra peers online, while Figure 3.28 shows the number of those peers sharing the PRs. We can see that we can retrieve almost all provider records from non-Hydra peers. Results are similar to the 600 CID experiments.

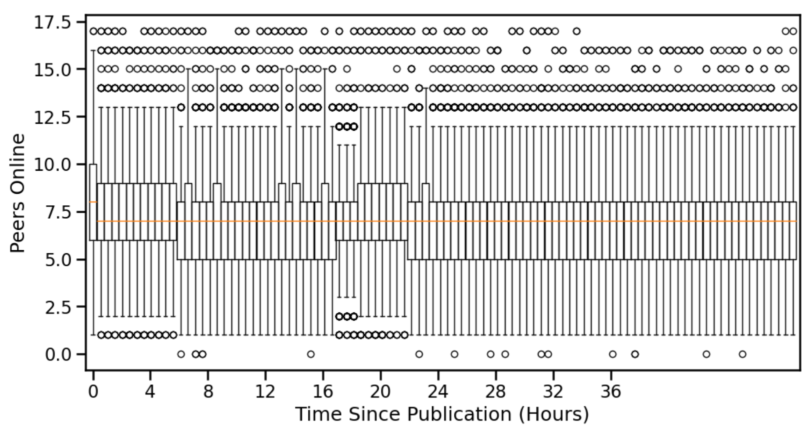


Figure 3.27: Average number of online non-Hydra peers.

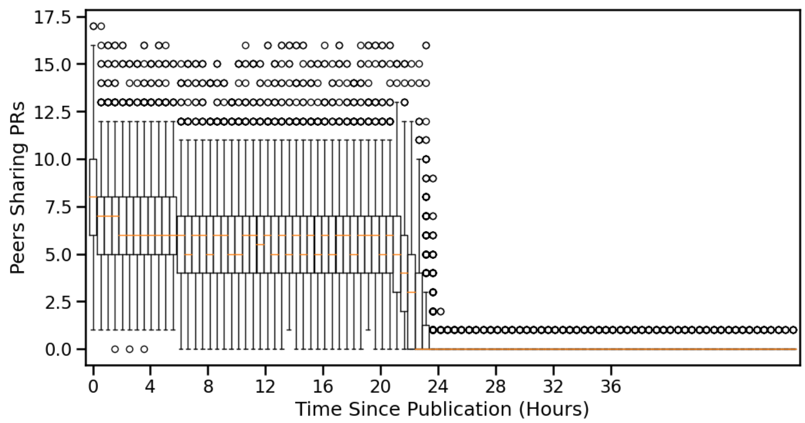


Figure 3.28: Average number of non-Hydra peers sharing the PRs.

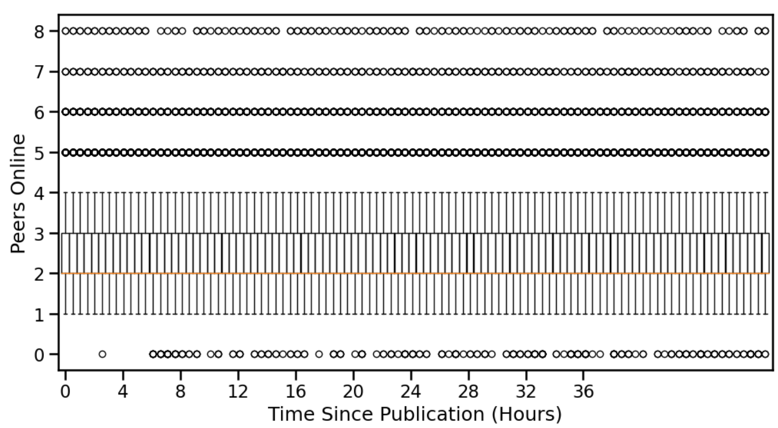


Figure 3.29: Average number of online Hydra peers.

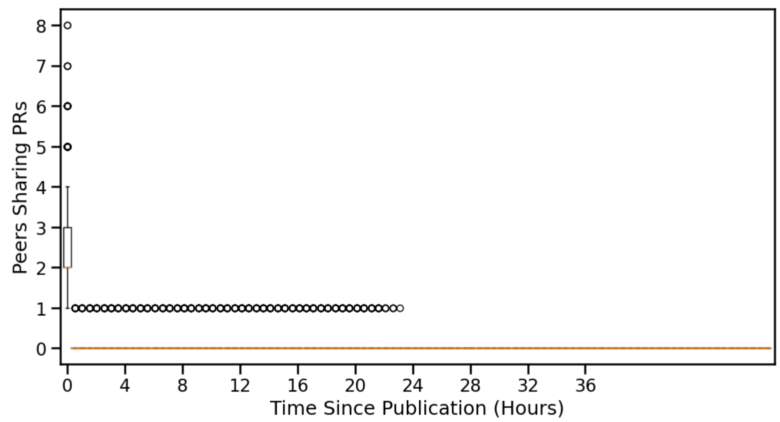


Figure 3.30: Average number of Hydra peers sharing the PRs.

Figure 3.29 shows the average number of Hydra peers online, while Figure 3.30 shows the number of those peers sharing the PRs. Results are again similar to the 600 CID experiments.

We now turn to the basic requirement for IPFS, that at least one peer shares the PRs, before they expire. Figure 3.31 shows that at least one peer shares the PRs with us, during the study before the 24h mark. Note that after the 24h mark the provider records are *still shared with us* until even the 49h mark.

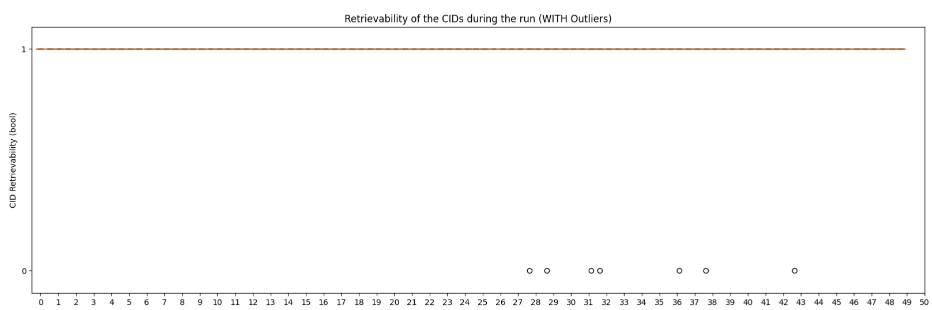


Figure 3.31: Liveness of the PRs.

Figure 3.29 shows the average number of peers responding to a DHT lookup. Note that even after the 24h mark (here the pings are counted, so around 48 pings) some peers might still respond back with the PRs up until 97 pings, 2 days after publication time.

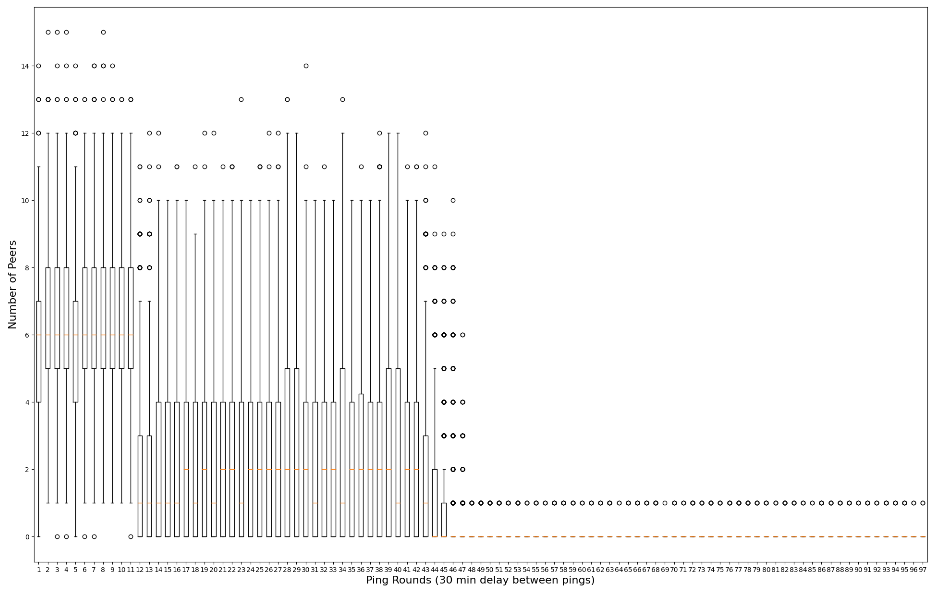


Figure 3.32: Average number of peers responding to a DHT lookup.

Finally, Figure 3.33 shows the results of the DHT lookup. It is important to point out that even when the peers hold a PR, they may not respond (e.g., due to the resource manager blocking us out). Again, we can see that even after the 24h mark (here the pings are counted so around 48 pings) some peers might still respond back.

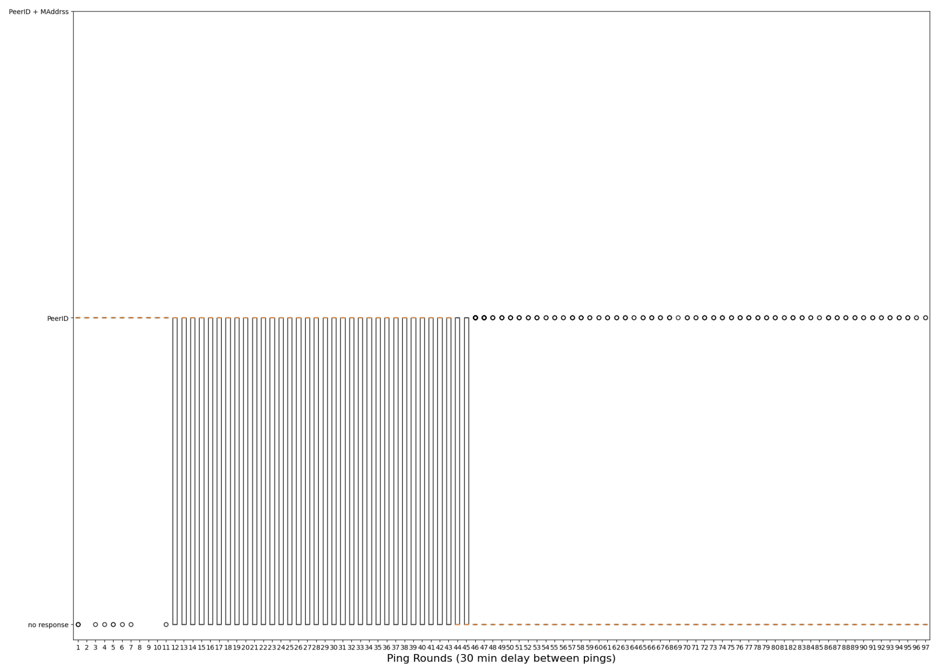


Figure 3.33: Results of a DHT lookup.

3.4 Fourth experiment: 1000 CIDs/JSON

The fourth experiment is the same as the third one, but performed at slightly later time. Figure 3.34 shows the distribution of successful PR holders during the publication process. It can be observed that most of the successful PR holder are in range of 8-12, but there were a few CIDs that were stored at less than 5 nodes.

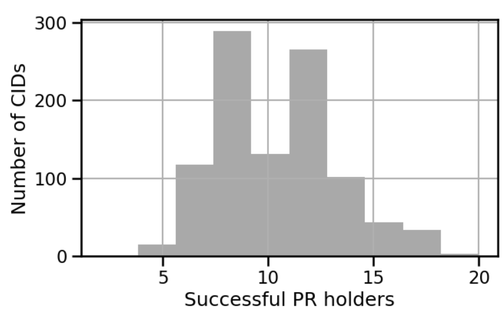


Figure 3.34: Distribution of successful PR holders during publication.

Figure 3.35 shows the average number of online peers over time, while Figure 3.36 shows the average number of peers sharing the PRs. Both these graphs are based on directly pinging the peers known to hold the PRs by the hoarder. We see that for an average of 9-10 online peers, 5-6 of them will share the PRs, dropping down to 0 around the 24h mark, when the PRs expire.

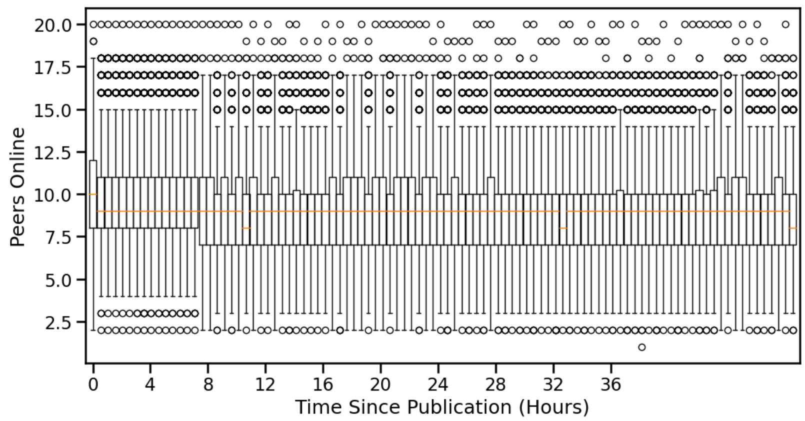


Figure 3.35: Average number of online peers.

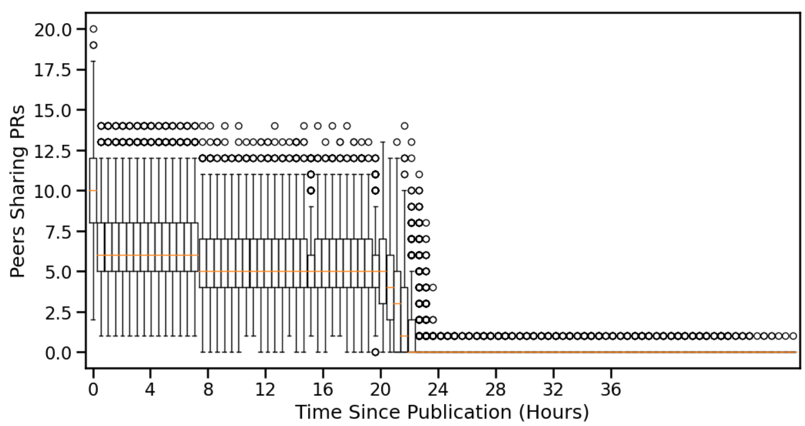


Figure 3.36: Average number of peers sharing the PRs.

To assess how many of these peers remain in the K-closest peers set over the duration of the study, Figure 3.37 shows the average In-degree ratio per CID, that is, the number of initially chosen peers by the optimistic provide process that are among the k closest to the CID. We can observe that a 7-8 peers remain in the K-closest peers set.

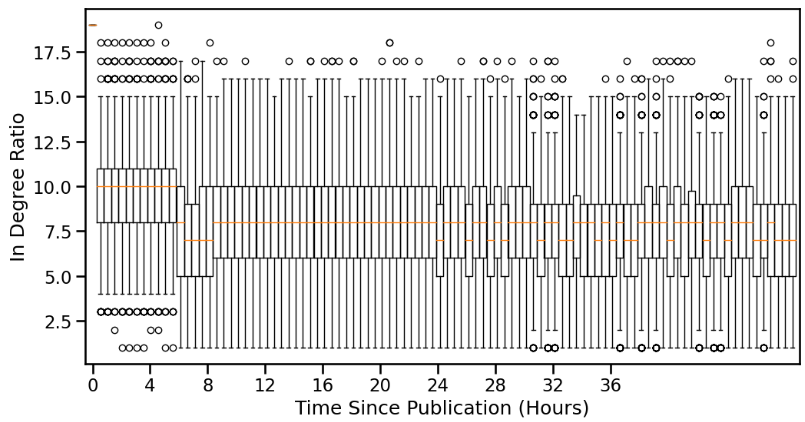


Figure 3.37: Average in-degree ratio.

Figure 3.38 shows the number of non-Hydra peers online, while Figure 3.39 shows the number of those peers sharing the PRs. Results are similar to the previous experiments.

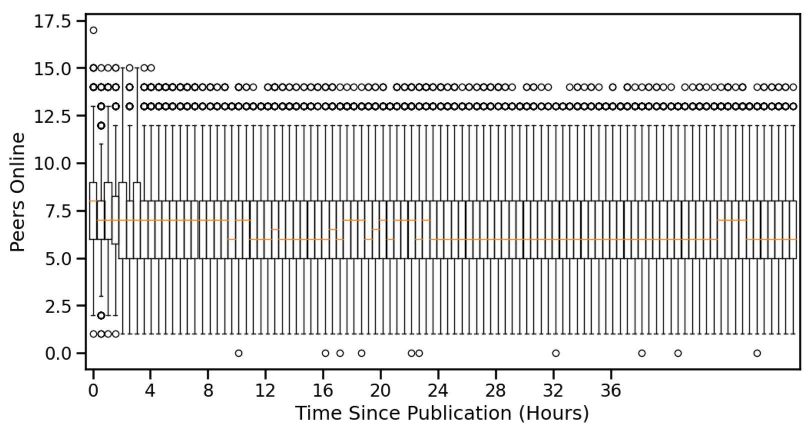


Figure 3.38: Average number of online non-Hydra peers.

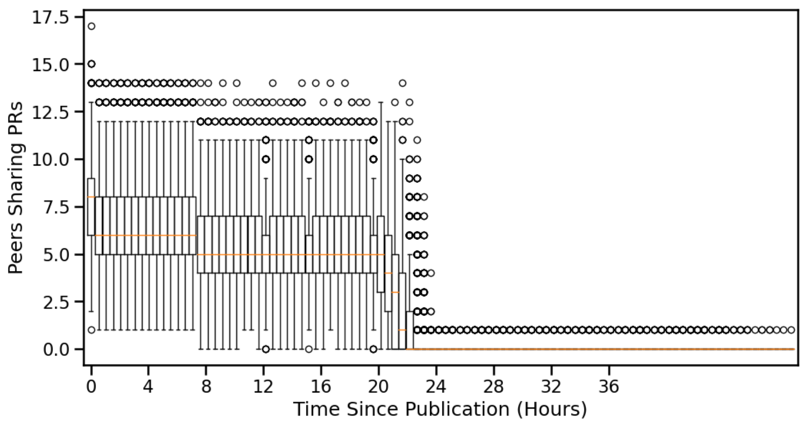


Figure 3.39: Average number of non-Hydra peers sharing the PRs.

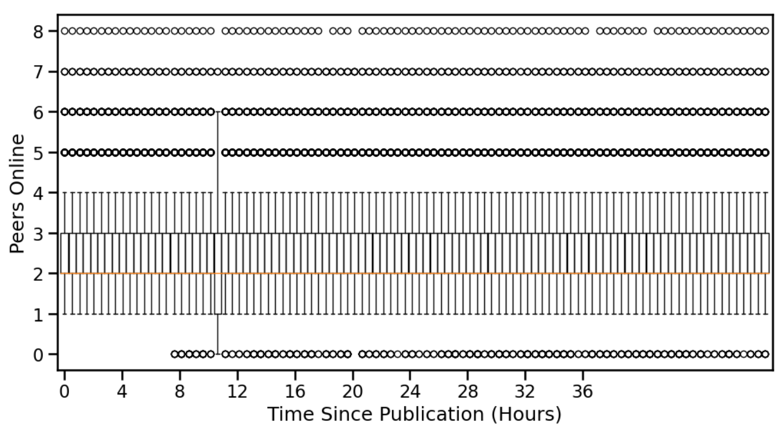


Figure 3.40: Average number of online Hydra peers.

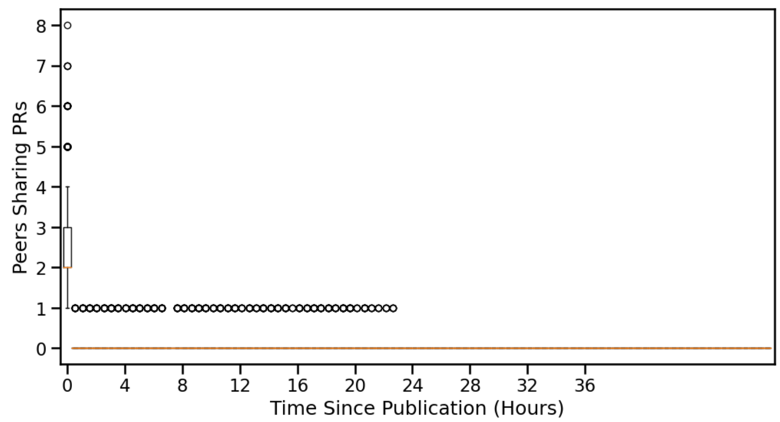


Figure 3.41: Average number of Hydra peers sharing the PRs.

Figure 3.40 shows the average number of Hydra peers online, while Figure 3.41 shows the number of those peers sharing the PRs. Results are again similar to the previous experiments.

We now turn to the basic requirement for IPFS, that at least one peer shares the PRs, before they expire. Figure 3.42 shows that at least one peer shares the PRs with us, during the study before the 24h mark. Note that after the 24h mark the provider records are *still shared with us* until even the 49h mark.

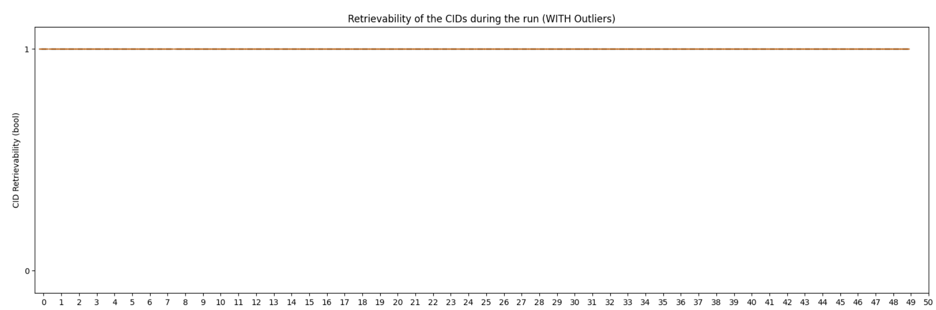


Figure 3.42: Liveness of the PRs.

Figure 3.40 shows the average number of peers responding to a DHT lookup. Note that even after the 24h mark (here the pings are counted, so around 48 pings) some peers might still respond back with the PRs up until 97 pings, 2 days after publication time.

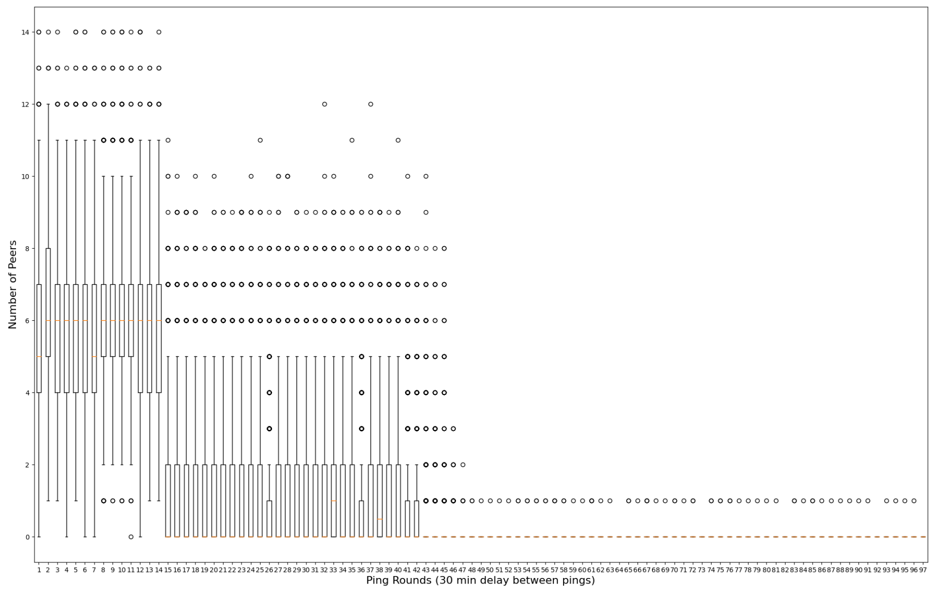


Figure 3.43: Average number of peers responding to a DHT lookup.

Finally, Figure 3.44 shows the results of the DHT lookup. It is important to point out that even when the peers hold a PR, they may not respond (e.g., due to the resource manager blocking us out). Again, we can see that even after the 24h mark (here the pings are counted so around 48 pings) some peers might still respond back.

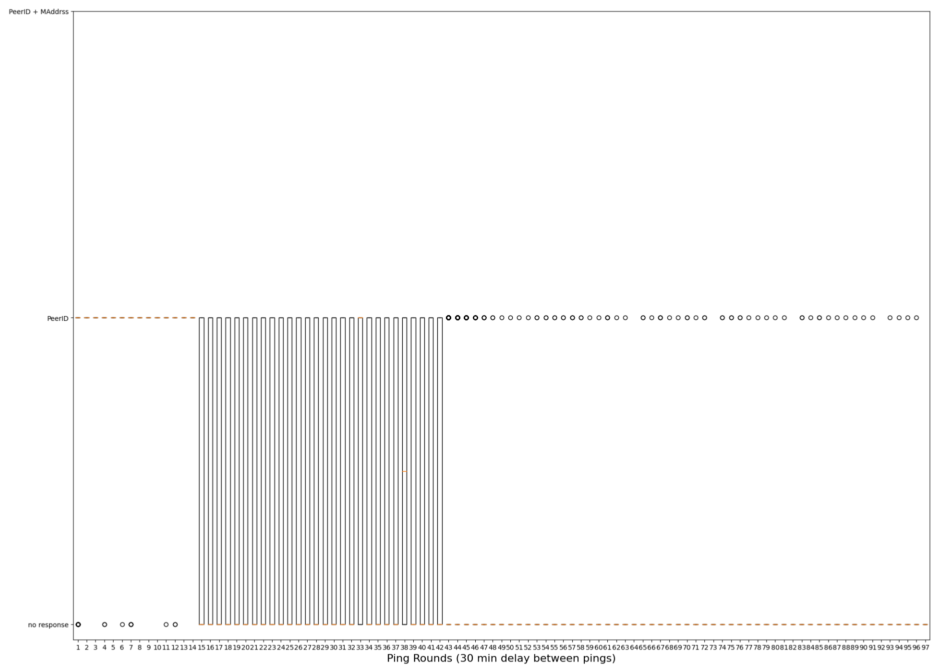


Figure 3.44: Results of a DHT lookup.

3.5 Fifth experiment: 1500 CIDs/JSON

The next four experiments were performed at a later date, when the network seemed to be in a better state than it was when the first four experiments were conducted. The number of CIDs was also gradually increased. For brevity, the graphs involving online nodes, retrievability and in-degree ratio are not shown, as they are similar to the previous ones, but with slightly higher values.

Figure 3.45 shows the distribution of successful PR holders during the publication process. In this (better) network state, there seems to be an increase in the successful put provider RPCs, as the highest numbers of CIDs are stored in 12-15 peers.

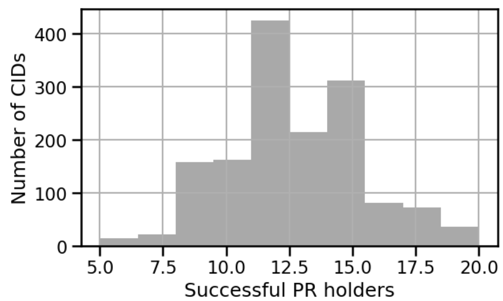


Figure 3.45: Distribution of successful PR holders during publication.

Regarding the basic requirement for IPFS, that at least one peer shares the PRs, before they expire, Figure 3.46 shows that results are almost the same as in the previous experiments, but there are some outliers that are not retrievable before the 24h mark. A peculiar finding is that all of the outliers can be retrieved afterwards, well into 38 hours. This indicates that this is either a hoarder issue or a network issue; for example, the cause might be that the hoarder cannot lookup forever for a specific key, a timeout is set after 2 mins (magic number) and the operation is timed out. This should be further analyzed in future work, in case this assumption is wrong.

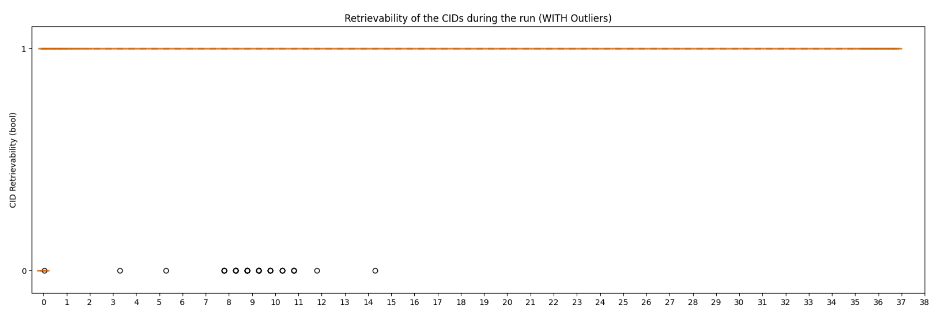


Figure 3.46: Liveness of the PRs.

Running the following query into the database can help further analyze the data:

```
SELECT * FROM fetch_results WHERE is_retrievable = FALSE AND
    ping_round > 0 ORDER BY ping_round DESC;
```

As Figure 3.47 shows, the outliers can be retrieved by subsequent ping rounds, pointing to the fact that specific DHT walks failed and that the CIDs can be indeed retrieved before 24h.

	id [PK] integer	cid_hash text	ping_round integer	fetch_time double precision	fetch_duration double precision	total_hops integer	hops_for_closest integer	holders_ping_duration double precision	find_prov_duration double precision	get_closest_peer_duration double precision	k integer	success_att integer	fail_att integer	is_retrievable boolean
1	48972	QmcZf8Had...	34	1678095903	10229	5	4	10221	982	818	11	10	1	false
2	41490	QmcZf8Had...	29	1678086903	23243	5	4	23236	993	883	11	10	1	false
3	36925	QmVSHGaZs...	27	1678081411	111233	2	2	111233	60000	60001	9	1	8	false
4	37029	QmQT6ko57...	27	1678081360	212376	2	2	212373	60000	60000	13	0	13	false
5	36957	QmcWCN4T...	27	1678081405	123346	2	2	123344	60008	60004	9	0	9	false
6	36945	QmSH5ycRC...	27	1678081399	127034	2	2	127034	60000	60000	9	0	9	false
7	36849	QmV3RDjHSt...	26	1678081320	184799	3	3	184793	60000	60000	9	1	8	false
8	37002	QmSSjU4PN...	26	1678081358	193176	2	2	193176	60000	60000	7	0	7	false
9	36916	QmQ77spw9...	26	1678081409	111963	2	2	111963	60002	60002	11	0	11	false
10	36890	QmRZd5PLW...	26	1678081336	179321	2	2	179317	60000	60000	14	0	14	false
11	36893	QmP8oLaEv...	25	1678081351	165468	3	3	165468	60000	60000	10	0	10	false
12	36899	QmXsyCcH1...	25	1678081395	123215	3	3	123213	60000	60000	12	2	10	false
13	36926	QmXTGHaTr...	25	1678081406	116790	2	2	116787	60000	60000	13	0	13	false
14	36898	QmYVPLYSh...	25	1678081412	105694	2	2	105691	60000	60000	14	0	14	false
15	36838	QmdYcg5vVf...	25	1678081340	153239	2	2	153239	60002	60000	14	0	14	false
16	36860	QmVZHN3q9...	25	1678081330	180879	2	2	180879	60000	60000	8	1	7	false
17	36962	Qmdud9fcpE...	24	1678081315	215118	3	3	215101	60000	60000	15	0	15	false
18	36837	QmWPiGZP6...	24	1678081325	168645	2	2	168645	60000	60000	20	3	17	false
19	36861	QmTDPjr95v...	24	1678081350	160988	2	2	160981	60000	60000	9	0	9	false
20	36873	Qma9AiWkC...	24	1678081343	169552	2	2	169550	60000	60000	9	0	9	false
21	36878	QmY31MPgz...	24	1678081362	151551	2	2	151548	60000	60000	9	1	8	false
22	36889	QmcXuWZSD...	24	1678081355	160271	2	2	160271	60000	60000	10	1	9	false
23	36938	QmYGr5G56...	24	1678081410	114545	2	2	114541	60001	60000	16	0	16	false
24	36894	QmXmY9sey...	23	1678081406	110576	1	1	110567	60003	60000	20	0	20	false
25	36920	QmXXrmvjEt...	23	1678081402	119787	3	3	119787	60000	60000	16	3	13	false
26	36865	QmUkjesXf9i...	23	1678081327	183761	3	3	183761	60019	60009	8	0	8	false
27	36863	QmZgbyFGre...	23	1678081352	159219	2	2	159213	60000	60000	14	1	13	false
28	36954	QmNqzkyUV...	23	1678081333	194733	2	2	194719	60000	60000	15	1	14	false
29	36853	QmXbMzsPE...	23	1678081397	109846	2	2	109844	60008	60009	8	1	7	false
30	36984	QmT293SttK...	23	1678081338	199613	3	3	199607	60000	60000	11	0	11	false
31	36998	QmcNudcP8...	23	1678081362	187816	2	2	187810	60000	60000	10	0	10	false
32	37031	QmZWcjp5u...	23	1678081316	256981	2	2	256977	60000	60000	11	0	11	false
33	36910	QmbtkMUZY...	22	1678081332	188376	2	2	188376	60006	60005	11	1	10	false

Figure 3.47: Output of query.

Running the following query into the database can help determine the percentage of failed DHT walks:

```
SELECT
negatives.cid,
negatives.coun,
total.cid,
total.coun,
(negatives.coun::NUMERIC/total.coun::NUMERIC) as percentage FROM
(
  SELECT cid_hash as cid,count(*) as coun
  FROM fetch_results
  WHERE
   ping_round > 0 AND is_retrievable=False
  GROUP BY(cid)
) AS negatives
JOIN (
  SELECT cid_hash as cid,count(*) as coun
  FROM fetch_results
  WHERE ping_round > 0
  GROUP BY(cid)
) AS total ON negatives.cid = total.cid;
```

	cid text	coun bigint	cid text	coun bigint	percentage numeric
1	QmcZf...	4	QmcZ...	48	0.08333333333333333333
2	QmRZd...	1	QmRZ...	48	0.02083333333333333333
3	Qmdud...	1	Qmdu...	48	0.02083333333333333333
4	QmXb...	1	QmXb...	48	0.02083333333333333333
5	QmSH...	1	QmSH...	48	0.02083333333333333333
6	QmcW...	1	QmcW...	48	0.02083333333333333333
7	QmeHh...	1	QmeH...	48	0.02083333333333333333
8	QmTE4...	1	QmTE...	48	0.02083333333333333333
9	QmXT...	1	QmXT...	48	0.02083333333333333333
10	QmQ7...	1	QmQ7...	48	0.02083333333333333333
11	QmXXr...	1	QmXX...	48	0.02083333333333333333
12	QmTZk...	1	QmTZ...	48	0.02083333333333333333
13	QmQT6...	1	QmQT...	48	0.02083333333333333333
14	QmYtW...	1	QmYt...	48	0.02083333333333333333
15	QmYV...	1	QmYV...	48	0.02083333333333333333
16	QmcNu...	1	QmcN...	48	0.02083333333333333333
17	QmYGr...	1	QmYG...	48	0.02083333333333333333
18	Qmd2R...	1	Qmd2...	48	0.02083333333333333333
19	QmTD...	1	QmTD...	48	0.02083333333333333333
20	QmT29...	1	QmT2...	48	0.02083333333333333333
21	QmWPi...	1	QmW...	48	0.02083333333333333333
22	QmV3R...	1	QmV3...	48	0.02083333333333333333
23	QmXm...	1	QmX...	48	0.02083333333333333333
24	Qma9A...	1	Qma9...	48	0.02083333333333333333
25	QmVA...	1	QmVA...	48	0.02083333333333333333
26	QmcXu...	1	QmcX...	48	0.02083333333333333333
27	QmZW...	1	QmZ...	48	0.02083333333333333333
28	QmQ77...	1	QmQ7...	48	0.02083333333333333333
29	QmP8o...	1	QmP8...	48	0.02083333333333333333
30	QmSSj...	1	QmSSj...	48	0.02083333333333333333
31	QmZgb...	1	QmZg...	48	0.02083333333333333333
32	QmVZ...	1	QmVZ...	48	0.02083333333333333333

Figure 3.48: Output of query.

As Figure 3.48 shows, the percentage of failed DHT walks for each specific CID before the 24h mark.

3.6 Sixth experiment: 2000 CIDs/JSON

Figure 3.49 shows the distribution of successful PR holders during the publication process. In this (better) network state, there seems to be an increase in the successful put provider RPCs, as the highest numbers of CIDs are stored in 12 peers.

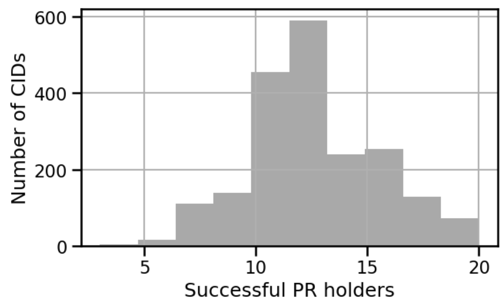


Figure 3.49: Distribution of successful PR holders during publication.

Figure 3.50 shows whether the PRs were retrievable. After the 21h mark some CIDS are not retrievable. This is likely attributed to the fact that the publish time of the CIDs is different and we are inserting the PRs using a JSON file, which may cause us to miss some ping rounds. This is further supported by the query we are doing for the database later. There are some outliers before the 24h mark, but this can attributed to other reasons rather than Optimistic Provide, due to the small number of outliers.

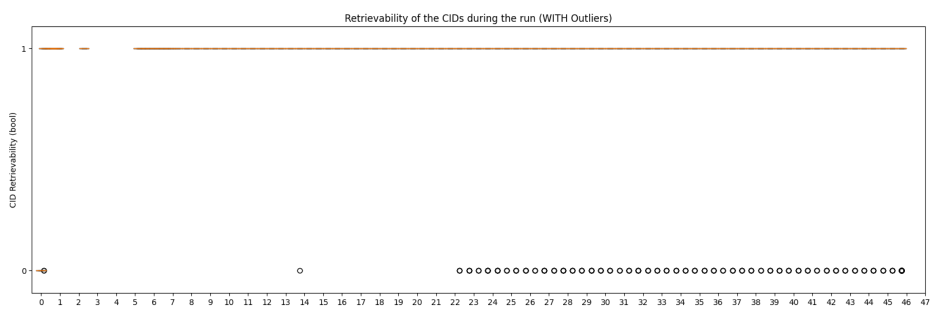


Figure 3.50: Liveness of the PRs.

Running the following query into the database can help further analyze the data:

```
SELECT * FROM fetch_results WHERE is_retrievable = FALSE AND
    ping_round > 0 ORDER BY ping_round DESC;
```

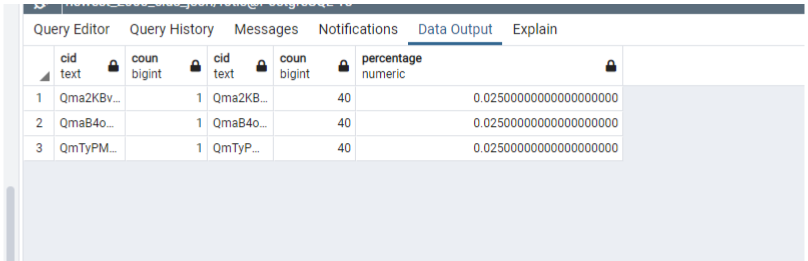
As Figure 3.51 shows, except for some outliers which are retrievable in later rounds, all of the CIDs are retrievable before the 24h mark (ping round 48). For round 1 two CIDs are not retrievable but then are retrievable and for round 32 one CID is not retrievable but then it also becomes retrievable.

	id [PK] integer	cid_hash text	ping_round integer	fetch_time double precision	fetch_duration double precision	total_hops integer	hops_for_closest integer	holders_ping_duration double precision	find_prov_duration double precision	get_closest_peer_duration double precision	k integer	success_att integer	fail_att integer	is_retrievable boolean
1	2237	QmTyPM9to...	1	1677802081	126894	5	5	126845	60006	60007	9	8	1	false
2	2336	Qma2KBv8v...	1	1677802082	153066	4	3	153054	60026	60025	10	6	4	false
3	45634	QmaB4o1pj8...	32	1677847730	65618	5	4	65618	60027	60021	11	11	0	false
4	85992	QmPvAyFB6...	49	1677884123	15347	3	3	15346	15343	10344	11	10	1	false
5	83601	Qmc1anEPM...	49	1677881960	15955	4	3	6680	15954	15954	4	4	0	false
6	91727	QmVt4LEKU...	49	1677889279	15754	4	3	6730	15754	15754	4	4	0	false
7	93742	QmVt4LEKU...	50	1677891079	15760	4	3	6550	15760	15759	4	4	0	false
8	88300	Qmaw26sHr...	50	1677886177	28797	4	4	28786	18418	21317	5	4	1	false
9	87988	QmPvAyFB6...	50	1677885923	15332	3	3	15332	15318	10319	11	10	1	false
10	85600	Qmc1anEPM...	50	1677883760	15733	4	3	6534	15732	15730	4	4	0	false
11	95789	QmVt4LEKU...	51	1677892879	60001	3	3	6548	60000	60000	4	4	0	false
12	87593	Qmc1anEPM...	51	1677885560	11300	4	3	6678	11300	11300	4	4	0	false
13	90495	Qmaw26sHr...	51	1677887977	205027	4	3	205007	60011	60012	5	4	1	false
14	97739	QmVt4LEKU...	52	1677894679	15729	4	3	6678	15728	15728	4	4	0	false
15	89592	Qmc1anEPM...	52	1677887360	15851	4	3	6691	15851	15850	4	4	0	false
16	91982	QmPvAyFB6...	52	1677889523	15674	4	3	15665	10897	10896	11	9	2	false
17	96028	QmeS4xZAL...	52	1677893146	13655	4	4	5408	13329	13643	9	9	0	false
18	92279	Qmaw26sHr...	52	1677889777	15193	3	3	15191	10238	10229	5	4	1	false
19	94299	Qmaw26sHr...	53	1677891577	15252	4	3	15241	10239	10462	5	4	1	false
20	94012	QmPvAyFB6...	53	1677891323	20723	3	3	20721	10724	10724	11	9	2	false
21	91591	Qmc1anEPM...	53	1677889160	15734	4	3	6682	15733	15733	4	4	0	false
22	98086	QmeS4xZAL...	53	1677894946	60001	5	3	5699	59999	60000	9	9	0	false
23	99735	QmVt4LEKU...	53	1677896479	15728	3	3	6671	15728	15717	4	4	0	false
24	93610	Qmc1anEPM...	54	1677890960	11158	3	3	6539	11158	11158	4	4	0	false
25	96298	Qmaw26sHr...	54	1677893377	15438	4	3	15427	10424	10423	5	4	1	false
26	101735	QmVt4LEKU...	54	1677898279	15756	3	3	6737	15754	15755	4	4	0	false
27	98319	Qmaw26sHr...	55	1677895177	44575	4	4	44558	43924	43143	5	4	1	false
28	95606	Qmc1anEPM...	55	1677892760	11161	4	4	6544	11160	11160	4	4	0	false
29	103727	QmVt4LEKU...	55	1677900079	15847	4	4	6542	15847	15847	4	4	0	false
30	98002	QmPvAyFB6...	55	1677894923	15624	4	3	15609	10620	10620	11	9	2	false
31	97603	Qmc1anEPM...	56	1677894560	15730	4	3	6726	15729	15729	4	4	0	false
32	100302	Qmaw26sHr...	56	1677896977	23842	4	4	23833	22844	21851	5	4	1	false
33	105723	QmVt4LEKU...	56	1677901879	15869	4	3	6726	15868	15866	4	4	0	false

Figure 3.51: Output of query.

Running the following query into the database can help determine the percentage of failed DHT walks:

```
SELECT
negatives.cid,
negatives.coun,
total.cid,
total.coun,
(negatives.coun::NUMERIC/total.coun::NUMERIC) as percentage FROM
(
  SELECT cid_hash as cid,count(*) as coun
  FROM fetch_results
  WHERE
   ping_round > 0 AND ping_round <= 48 AND is_retrievable=False
  GROUP BY(cid)
) AS negatives
JOIN (
  SELECT cid_hash as cid,count(*) as coun
  FROM fetch_results
  WHERE ping_round > 0 AND ping_round <= 48
  GROUP BY(cid)
) AS total ON negatives.cid = total.cid ORDER BY percentage DESC;
```



Query Editor Query History Messages Notifications Data Output Explain

	cid text	coun bigint	cid text	coun bigint	percentage numeric
1	Qma2KBv...	1	Qma2KB...	40	0.02500000000000000000
2	QmaB4o...	1	QmaB4o...	40	0.02500000000000000000
3	QmTyPM...	1	QmTyP...	40	0.02500000000000000000

Figure 3.52: Output of query.

As Figure 3.52 shows, the percentage of failed DHT walks for each specific CID before the 24h mark.

Chapter 4

Conclusions

The results of this study indicate that the Optimistic Provide algorithm maintains the initial guarantees of the standard provide algorithm that come along with content provision, while being much faster and lightweight. It ensures that the requesting node still receives the requested content from a sufficient number of peers and at least one peer can always be found, even though the second set of results casts some doubts about this, and requires further study. It therefore seems, especially from the first set of results, that Optimistic Provide is a promising solution for improving the provide time without sacrificing the reliability and robustness of the system.

It is important to note that in our study of optimistic provide we only considered successful put provider RPCs, while other studies using the CID hoarder have looked at both failed put provider RPCs and successful put provider RPCs. This is an important distinction to make, as including failed put provider RPCs would likely lead to higher In-degree ratios and higher online averages.

Our findings suggest that the Optimistic Provide algorithm can be further improved by increasing the number of successful initially selected peers. While the algorithm was able to retrieve PR holders from the peers, we observed that the number of successful peers was not as high as the standard provide algorithm which is around 15. The findings from the second set of experiments, reflecting a different network state, indicate that the algorithm manages to choose on average more successful peers. By increasing the number of successful peers, we believe that the algorithm can achieve even greater success in retrieving PR holders. This improvement can potentially enhance the overall performance of the algorithm and reduce the time it takes to retrieve PR holders from the network. Therefore, it seems that an important future step for optimistic provide would be to increase the number of successful put provider RPCs.

Moreover, our analysis indicates that the Optimistic Provide algorithm maintains a consistently high average In-degree ratio compared to the online average. This is an important finding, as a high In-degree ratio is crucial for ensuring that the requested data can be quickly and reliably retrieved from the network.

Another important fact is that the Hydra dial down has a significant impact on the performance of the Optimistic Provide algorithm and the overall network. This is because Hydras are selected as initial PR holders by the algorithm, and the dial down affects their ability to serve as reliable providers. There are some outlier hydras that seem to share the provider records with us. This is probably, because there are hydras running outside of the known dialed down ones. The dialed down hydras cause the general retrievability average of Optimistic Provide to drop. Therefore, it is crucial to consider the impact of Hydra dial down when implementing and evaluating the performance of the Optimistic Provide algorithm. Further research is needed to explore ways to mitigate this impact and improve the algorithm's ability to select reliable initial providers, and avoid unreliable ones.

A very peculiar finding that is not necessarily related to Optimistic Provide is that some PRs can still be retrieved even after the 24-hour mark, up until the 48-hour mark. This suggests that the information sharing among peers does not completely stop after the initial 24 hours. Some new implementations of DHT have increased the PR deletion up until 48h. What we are observing is probably that most IPFS peers are running the new implementations , while there are others which are running on older versions.This means that the provider record holders keep the PRs even after the 24 hour mark and share them.

Acknowledgements

I would like to extend my deepest appreciation to my thesis advisor, Professor George Xylomenos, for his invaluable guidance, support, and his initiative to introduce me to the Probe Lab team. His knowledge, expertise, and experience have been instrumental in the successful completion of this work. I would also like to thank my colleagues at Probe Lab for their helpful insights and suggestions. In particular, I 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. I hope that in the future, if the circumstances permit it, I will be able to see the completion of the Optimistic Provide algorithm. Furthermore working with the Probe Lab team was a incredible learning experience that made understand how to work in the demanding market of programming / computer science, while actually being rewarding and pleasant to work for.

Bibliography

[1] D. Trautwein. (2022) Optimistic provide. https://github.com/dennis-tra/optimistic-provide.

[2] M. Cortes. (2022) Ipfs cid hoarder. https://github.com/cortze/ipfs-cid-hoarder.

[3] F. Bistas. (2022) Hoarder pull request. https://github.com/cortze/ipfs-cid-hoarder/pull/11.

[4] Anon. (2020) Ipfs documentation. https://docs.ipfs.tech/concepts/.

[5] D. Trautwein, Y. Psaras, M. Cortes, and M. Guillaume. (2022) Rfm17. https://github.com/protocol/network-measurements/blob/master/results/rfm17-provider-record-liveness.md.

[6] Anon. Sybil attack. https://en.wikipedia.org/wiki/Sybil_attack.

[7] D. Trautwein. (2022) Hydra talk. https://www.youtube.com/watch?v=zhzxJGoLTg0&t=281s.

[8] web3-dev team. (2021) Improve ipfs content providing proposal. https://github.com/protocol/web3-dev-team/blob/main/proposals/ipfs-content-providing.md.

[9] Anon. (2022) Kademlia. https://en.wikipedia.org/wiki/Kademlia.

[10] D. Trautwein. (2022) op modeling. https://www.notion.so/pl-strflt/Network-Size-Estimation-4ab2c52083ed4e88968f629d1fa47eb7.

[11] F. Bistas and M. Cortes. (2022) Ipfs cid hoarder fork. https://github.com/FotiosBistas/ipfs-cid-hoarder#readme.