IPFS in the Fast Lane: Accelerating Record Storage with Optimistic Provide

Dennis Trautwein*†, Yiluo Wei‡, Yiannis Psaras†, Moritz Schubotz§, Ignacio Castro¶, Bela Gipp*, Gareth Tyson‡¶
*University of Göttingen, †Protocol Labs Inc., ‡Hong Kong University of Science & Technology (GZ),
§FIZ Karlsruhe – Leibniz Institute for Information Infrastructure, ¶Queen Mary University of London

Abstract—**The centralization of web services has raised concerns about critical single points of failure, such as content hosting, name resolution, and certification. To address these issues, the "Decentralized Web" movement advocates for decentralized alternatives. Distributed Hash Tables (DHTs) have emerged as a key component facilitating this movement, as they offer efficient key/value indexing. The InterPlanetary File System (IPFS) exemplifies this approach by leveraging DHTs for data indexing and distribution. A critical finding of previous studies is that DHT PUT performance for record storage is unacceptably slow, sometimes taking minutes to complete and hindering the adoption of delay-intolerant applications. To address this challenge, this research paper presents three significant contributions. First, we present the design of Optimistic Provide, an approach to accelerate DHT PUT operations in Kademlia-based IPFS networks while maintaining full backward compatibility. Second, we implement and deploy the mechanism and see its usage in the de-facto IPFS deployment, Kubo. Third, we evaluate its effectiveness in the IPFS and Filecoin DHTs. We confirm that we enable sub-second record storage from North America and Europe for 90% of PUT operations while reducing networking overhead by over 40% and maintaining record availability.**

Index Terms—**InterPlanetary FileSystem, IPFS, DHT performance, Kademlia, Filecoin**

I. Introduction

Powerful economies of scale have fueled the growing consolidation of web systems [1], [2]. For example, administrators of websites frequently opt to utilize cloud platforms such as Amazon EC2, third-party domain name system providers such as GoDaddy, content delivery networks such as Akamai, as well as central Certificate Authorities like Let's Encrypt. Although convenient, these constitute single points of technical, organizational, and regulatory failure.

The "*Decentralized Web*" has emerged as a response to this increasing concentration of power. The *Decentralized Web* encompasses a range of technologies designed to shift control away from these dominant entities. These technologies operate through open-source software, attempting to decentralize conventional functions of the web, including name systems [3], [4], object storage [5], [6], and social media [7], [8]. By enabling anyone to utilize and contribute to the software, this endeavor aims to reduce barriers to participation and counteract the prevailing trend of web consolidation [1].

One exemplar is the InterPlanetary FileSystem (IPFS) — a set of peer-to-peer (P2P) protocols that enable the decentralized publication and retrieval of web objects. IPFS is widely recognized as the primary storage layer for the Decentralized Web because of its content-addressed nature, decoupling the content identifier from its storage location. The adoption of IPFS is widespread, with over 120 million IPFS gateway requests and more than 200 k unique peers participating in the P2P network every week [9]. IPFS serves as the underlying technology for various Decentralized Web applications, encompassing social networking and discussion platforms (Discussify, Matters News), data storage solutions (Space, Peergos, Temporal), content search (Almonit, Deece), messaging (Berty), content streaming (Audius, Watchit), and e-commerce (Ethlance, dClimate) [10]. Notably, mainstream browsers like Opera and Brave have integrated support for accessing IPFS, facilitating its widespread and easy adoption.

At the heart of IPFS is a Kademlia DHT, which is used to index available content and facilitate peer discovery. Previous studies have found performance challenges in IPFS's use of the Kademlia DHT [5]. While the `GET` performance is generally satisfactory [5], [9], with record retrieval times under a second, the `PUT` performance is significantly slower, taking several dozen seconds or even minutes. Yet, efficient DHT `PUT` performance is crucial for the delay-sensitive publication of objects in the network. IPFS's content-addressed nature makes it necessary for mutable content to be re-published with every change. As users expect content changes to propagate quickly, low-latency access to data is vital for meeting the demand of delay-intolerant interactions in IPFS [11].

Although there has been a wealth of prior work focusing on optimizing the `GET` performance of Kademlia-based DHTs, there has been little work optimizing the performance of `PUT` operations. As this is vital for the operation of IPFS, we propose an optimized `PUT` operation for the Kademia DHT in IPFS, which we refer to as *Optimistic Provide*. Through empirical measurement, we show that the long delays in IPFS `PUT` operations are driven primarily by unreachable peers and exacerbated by the termination condition of the existing design. To overcome this, we develop a novel model capable of predicting the likelihood that discovered peers will belong to the eventual target set of peers used to store the record. This allows us to "optimistically" store records on the most likely target peers before all peers have responded. By tuning the query termination condition, we can therefore avoid waiting for unreachable peers to reply. Through this, we achieve sub-second publication times in the IPFS network from North America and Europe for 90 % of `PUT` operations, equating to a speed-up of over one order of magnitude. Our contributions are:

1) We design the *Optimistic Provide* mechanism — an approach to accelerate DHT `PUT` operations in Kademlia-based IPFS networks while maintaining full backward

compatibility.

2) We implement and deploy our Optimistic Provide mechanism in the `go-libp2p-kad-dht`[1] library, and see its usage in the de-facto IPFS deployment, Kubo.
3) We evaluate the effectiveness of our approach. We examine its performance, reliability, and overhead across two of the most important IPFS DHTs. We confirm that we improve performance by over one order of magnitude while maintaining data availability and reducing networking overhead by over 40 %. This enables sub-second record storage in North America and Europe for 90 % of the `PUT` operations.

By addressing the performance challenges associated with the `PUT` operation in IPFS' Kademlia DHT, this research contributes to the overall improvement and scalability of decentralized web systems, facilitating their adoption and practical usability.

II. Background

We provide a brief overview of IPFS and its use of the Kademlia DHT protocol. For full details, we redirect interested readers to [5] and [12], respectively.

A. The InterPlanetary File System (IPFS)

Overview. The InterPlanetary File System (IPFS) is a set of protocols that facilitate decentralized content-addressable media object storage and retrieval. At its core is a content-based addressing scheme using unique **Content Identifiers (CIDs)**. These CIDs are hash-based, immutable, and self-certifying. This means any peer can serve the content requested via a CID, and the receiver can verify that the data served matches the identifier requested using hashing. In order to know which node hosts the data for a certain CID, IPFS uses Distributed Hash Tables (DHTs) to maintain a mapping between a CID and the physical node that can serve it. Peers can then use this DHT to publish and retrieve objects without requiring any central authority. The ecosystem consists of multiple such DHTs facilitating content and peer discovery (discussed later).

Content Addressing. When new content is added to IPFS, it is first divided into smaller chunks, usually 256 kB in size [13]. Each chunk is then assigned a CID, generated by hashing the content and then attaching metadata. This unique identifier is used for publishing and retrieving the unique object.

Peer Addressing. Every peer within IPFS also has a unique identifier, known as a PeerID, which is created by hashing its public key. Within IPFS, anyone can search (via the DHT) for a specific PeerID to retrieve its associated network addresses (*e.g.,* IP address plus port). The network addresses can then be utilized to establish a connection with that peer.

Content Publication. To make content available in the IPFS network, two records are written to the DHT: (*i*) a *provider record* mapping a CID to a PeerID, and (*ii*) a *peer record* mapping the PeerID to its network addresses. Both processes require locating the 20 closest peers to the CID or PeerID, respectively, and storing the record with all of them [12]. Record replication on 20 peers ensures availability in the presence of churn.

Content Retrieval. To find the provider network addresses for a CID, a peer (*i*) looks up the provider record to find the PeerID that claims to provide the content on the DHT, and (*ii*) looks up the peer record to identify its physical address. Once this has been obtained, the content request is sent to the host.

Filecoin. IPFS involves peers hosting each others' content. Naturally, this requires an incentive model to encourage peers to contribute resources. To achieve this, IPFS uses *Filecoin*, an incentivized storage layer building on top of IPFS[2]. The Filecoin protocol ensures, through monetary incentives and mathematical proofs, data replication and availability over time. As part of this, **Filecoin relies on a separate DHT** for peer discovery. Full details can be found in [14].

B. Kademlia in IPFS

Kademlia Protocol. IPFS uses Kademlia for its DHT implementation for both publishing provider records as well as peer discovery. The Kademlia DHT protocol [12] is a decentralized peer-to-peer algorithm designed for efficient retrieval and storage of key-value pairs in a large network of peers. Each peer in Kademlia possesses a unique and random identifier in the form of a fixed-size bit string. The identifier functions not only as a means of identification but also as a routable entity in the network. For that, each peer builds its local routing table and employs XOR distance metric to calculate routing hops.

XOR Metric. The XOR metric is the basis for the Kademlia protocol's key space and distance calculation. It enables a peer to derive the distance between any two node IDs (PeerIDs) or file hashes (CIDs) by performing a bitwise XOR operation on them.

Routing Table. The routing table in Kademlia is a data structure that each peer maintains to facilitate efficient routing and neighbor selection. The routing table is organized as a binary trie, commonly referred to as a k-bucket trie, where each depth level represents a specific prefix length of the PeerIDs. The highest level corresponds to the most significant bits, while the lowest level represents the least significant bits. Each peer's routing table is divided into k-buckets, with each bucket containing k peers that share a specific prefix length of their ID. The number k typically represents a small constant value and is set to 20 in the IPFS DHT network as originally proposed in [12]. Peers within a bucket are sorted in ascending order of their last contact time, allowing for efficient eviction of stale or unresponsive peers.

To maintain an up-to-date routing table, Kademlia employs a refresh mechanism. Peers periodically refresh their buckets by performing lookup operations for random IDs belonging to each bucket. These lookups trigger the exchange of peer

[1] https://github.com/libp2p/go-libp2p-kad-dht

[2] https://filecoin.io

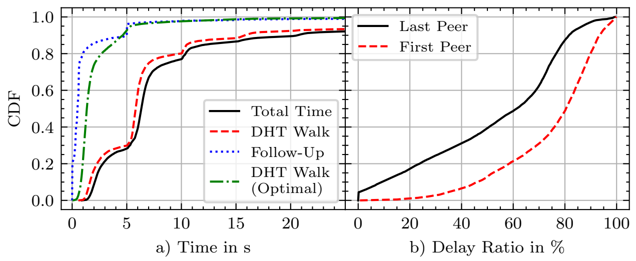

Fig. 1: Left: CDF of total PUT latencies in the IPFS DHT network and CDFs of their comprised DHT walk and follow-up phases. The optimal DHT walk CDF shows the optimization potential. Right: time between locating the first/last of the closest peers and initiating the storage of the provider record to them as a ratio of the total time. The experiment includes $8,503$ measurements between 2023-06-06 and 2023-06-07 from one host in `eu-central-1` AWS region.

information, allowing the peer to update its routing table with fresh data. During each routing table refresh, the peer, in theory, iterates through all of its buckets, which amounts to 256 buckets. However, the IPFS and Filecoin reference implementations Kubo[3] and Lotus[4] only refresh the first 16 buckets in practice. Consequently, each time the peer refreshes its routing table, it performs 16 look-ups to identify the 20 closest peers for 16 randomly selected keys that fall into each of its buckets.

Kademlia Routing. When the user initiates a request to retrieve a key-value pair, the local peer uses its routing table to identify the currently known closest peers to the target key. It then sends requests to the selected peers in parallel with a specific concurrency factor, progressively expanding the set of closer peers until it finds the desired key or exhausts the search depth. Through the XOR metric and routing table, the Kademlia DHT protocol is theoretically highly efficient as peers only need to contact $\log(N)$ peers in the network during a search where N is the total number of peers in the network.

PUT Operation. Storing provider records in the IPFS Kademlia DHT comprises two steps: (*i*) identifying the (network-wide) 20 closest peers to the target key (referred to as the *DHT Walk*), and (*ii*) storing the records with them (the *Follow-Up*).

The DHT walk involves *a)* adding the 20 closest peers to the target key from the local routing table to a query queue, *b)* concurrently querying the 10 closest ones from the query queue to ask for even closer candidates, *c)* adding the responses to the query queue, and then *d)* starting over with *b)*. The DHT walk terminates as soon as the initiator has queried and gotten responses from the three closest peers in its query queue. Finally, the *follow-up phase* starts. This phase involves contacting the 20 closest discovered peers from the query queue and pushing the record to them.

III. Measurements & Motivation

We start by exploring the current performance and bottlenecks of the PUT operation in IPFS.

[3] https://github.com/ipfs/kubo [4] https://lotus.filecoin.io/

PUT Performance. To assess the current PUT performance, we repeatedly generate random provider records, store them in the IPFS DHT, and measure the latency. Figure 1a depicts our results (measurement details are in the caption). The solid black line shows the cumulative distribution function (CDF) of the total time to store provider records in the IPFS DHT (DHT walk + follow-up phases). We observe PUT latencies of 6.3 s, 20.6 s, and 49.8 s for the 50th, 90th and 95th percentile, respectively. These latencies severely constrain the viability of delay-intolerant use-cases. For instance, in the RAIL user-centric performance model [15], delays above 1 s or 10 s result in user focus loss and user frustration, respectively.

Optimization Potential. Figure 1a further breaks down the latency CDFs for the DHT walk and follow-up phases. This shows that the DHT walk is the primary contributor to the overall latency, making up over $90\,\%$ of the total time in the majority of cases. We find that there is a significant delay between the discovery of the network-wide 20 closest peers and the follow-up phase. The green dotted-dashed line in Figure 1a shows a CDF of optimal DHT walks. An optimal DHT walk terminates immediately when the last of the 20 closest peers that were eventually chosen to hold the provider record is discovered. The graph shows that in most cases, the optimal DHT walk terminates several seconds earlier.

Figure 1b also shows CDFs of the time elapsed between discovering the first (dashed red line) and last (solid black line) of the 20 closest peers and the follow-up phase as a ratio of the total PUT operation time. The graph shows that in over $50\,\%$ of the PUT operations, the delay between discovering all the 20 closest peers and storing the provider records with them make up over $60\,\%$ of the total time of the operation. The effect is amplified in the delay of the discovery of the first of the 20 closest peers until storing the provider record with it: the delay makes up over $80\,\%$ of the total time in $50\,\%$ of the cases.

Causes of Underperformance. The results show that the main contributing factor for the delay is the DHT walk termination condition. The final peers are often discovered several seconds before they are contacted to hold the record. As mentioned above, the current algorithm requires successful responses from the three currently known closest peers. If any of the three peers is unreachable, their query delays the entire process. The DHT walk then progressively queries peers further away from the target key until the three closest ones it knows have successfully responded. We call this behavior *DHT Walk backtracking*. However, the peers that are more distant from the target key might have been discovered in earlier queries, contributing to the delay. Only $4.3\,\%$ of PUT operations do not experience any delay between discovering all 20 closest peers and the follow-up phase.

IV. Design Overview

The preceding section observed that PUT operations in the IPFS Kademlia DHT exhibit multi-second latencies, even though the median PUT operation discovers the peers with whom the query initiator ultimately stores the provider records already within 1.3 s. In order for IPFS to support a wider spectrum of delay-sensitive use-cases we propose the *IPFS Optimistic Provide* to improve the DHT `PUT` performance.

A. Optimistic Provide

Overview An *Optimistic Provide* query consists of the same phases and steps as a classic `PUT` operation, as described in Section II. The crucial differences are in (*i*) evaluating individual peers' target key proximity for immediate and effective record holding, (*ii*) evaluating the currently known 20 closest peers' target key proximity to decide when to terminate the DHT walk, and (*iii*) returning from the follow-up phase after only a subset of queries have completed.

As it will become clear later, a prerequisite for an *Optimistic Provide* is prior knowledge of the global network size. We will assume this knowledge exists for now and provide a comprehensive explanation in Section IV-B.

Step 1: Evaluate Target Key Proximity: Individual Peers. First, a peer launches a DHT walk to discover which peers to store its record with. During the DHT walk, each time the initiator encounters a peer, it verifies whether this peer is already sufficiently close to the target key to host the record. To determine if a peer is sufficiently close to a target key, it requires knowledge of the likelihood that there exist no 20 peers closer to the target key than the peer it is currently examining. Calculating this requires an estimate of the global network size. We later show that, by assuming uniform peer identifier distribution and possessing knowledge of the network's size, the initiator can predict the probability that the peer indeed belongs to the set of the 20 closest peers based on its proximity to the target key (details on how this is obtained in Section IV-B).

Step 2: Evaluate Target Key Proximity: Set of Peers. Next, upon each query, the initiator assesses the set of the 20 closest peers it currently knows. If it is highly probable that this set constitutes the actual 20 closest peers, the peer promptly terminates the DHT walk without awaiting final confirmation from the 3 closest peers (as done in the original algorithm). We later show how the initiator can predict the probability that the set of peers indeed represents the final set based on their aggregate proximity to the target key.

Step 3: Prematurely Return from the Follow-Up Phase. Recall, the follow-up phase involves the initiator pushing the record to the chosen 20 peers. However, premature DHT walk termination skips the previously mentioned backtracking behavior which negatively affects the latencies of the follow-up phase. Timeouts from contacting unreachable peers previously experienced during the DHT walk now increasingly manifest in the follow-up phase. Thus, a single peer could delay the whole follow-up phase because the initiator would wait for the request to time out. To address this, the Optimistic Provide algorithm returns control back to the user after k' peers with $k' < 20$ have responded. $20 - k'$ additional queries remain in-flight asynchronously. Although configurable, we choose $k' = 5$ based on previous studies where it was found that decreasing k to 15 in the IPFS network would have a negligible impact on record availability [16]. This ensures immediate record availability after control has been handed back to the user. We emphasize that the remaining queries are not canceled. We call waiting for all 20 responses the *done* strategy and only waiting for k' peers, the *return* strategy. We will refer to both in the following as *Control Flow Strategies*.

B. Network Size Estimation

To implement Optimistic Provide, it is necessary for a peer initiating a `PUT` to know the size of the global DHT. One seemingly intuitive solution might involve crawling the entire network to derive a number of participating peers. Yet, this approach proves highly impractical due to the excessive overhead it introduces. Distributing the task to a subset of peers that share the information would require trust in their honesty, which is a challenging proposition in a permissionless network. We therefore next present a lightweight technique that allows individual peers to locally estimate the global network size by briefly describing the peer to target key proximity model according to [17].

Proximity Model. Let the DHT network consist of N peers $P_1,\ P_2,\ \dots P_N$, where each peer is identified by its PeerID, which we assume to be random uniformly distributed identifiers. Each identifier is a bitstring of length L which means the address space spans $[0, 2^L)$. Recall, Distances in a Kademlia network are computed using the XOR $\oplus$ metric. If we choose a random address A in the address space, then the distance D to another address X would be $D(X) = X \oplus A$. When we take the N peers P_i with $i = 1, \dots, N$ and calculate their distances D_i to address A we end up with a set D of distances with $D = \{D_1, D_2, \dots, D_N\}$ where $D_1 = P_1 \oplus A$, $D_2 = P_2 \oplus A, \dots, D_n = P_N \oplus A$. If we assume, without loss of generality, $D_1 < D_2 < \dots < D_n$ then $D_i = D_{(i)}$ where $D_{(i)}$ is called the i-th order statistic. In figures, we normalize the address space of $[0, 2^L)$ to $[0, 1)$ for simplicity.

Now, we assume continuous random variables instead of discrete ones, which we justify with the large address space of size 2^L where $L = 256$ in the case of IPFS. Then, we can apply the probability distributions for order statistics sampled from a uniform distribution. These probability distribution functions are given by Beta distributions $B(\alpha, \beta)$ with varying values for their shape parameters α and β. In our case, $\alpha = i$ where i corresponds to the i-th order statistic in the range $i = 1, \dots, N$ and $\beta = N - i + 1$ [18, p.63].

Deriving the Network Size. To derive a network size estimation from this model, we focus on the expected values $\mathrm{E}[X]$ of the above parameterized Beta distributions, which is given by $\mathrm{E}[X] = \alpha/(\alpha + \beta)$ where X is a random variable [18]. In our case, the random variable is an order statistic $D_{(i)}$, which lets us rewrite the expected value with values for α and β from above to

$$\mathrm{E}[D_{(i)}] = \frac{1}{N+1} \cdot i \ . \tag{1}$$

This equation shows that the expected value for each order statistic (i) is proportional to $(N+1)^{-1}$ where, again, N is the network size we are searching for. This equation also confirms the intuition that a larger network yields a denser keyspace and vice-versa. Our strategy is to perform multiple look-ups, calculate the i-th-closest peers' average distances to the target keys, do a linear fit $f(x) = mx + b$ through the average values, and use the slope of the fit to arrive at a network size estimation. In our case, the linear fit must cross the origin, hence $b = 0$. This aggregation technique deviates from [17] because we find ours to yield network size estimations with at least $6.7\,\%$ lower variance.

Sampling the Keyspace. In order to obtain sufficient statistical information to carry out the above network size estimation, the above algorithm requires look-ups for random keys in the network. We naturally wish to avoid introducing additional overhead by initiating excessive queries. Therefore, we rely exclusively on information from the existing routing table refresh maintenance queries (see Section II). However, this introduces bias towards the peer's local keyspace density and thus affects the network size estimation. This is because when performing a routing table refresh, a peer searches for random keys within all of its buckets. Consequently, the routing maintenance search tends to uncover peers that are in close proximity to the searching peer. As a result, when dealing with higher bucket numbers, we tend to encounter the same peers repeatedly. This significantly affects the distribution and favors the density in the vicinity of the searching peer's current keyspace location. If the keyspace happens to be dense, the peer will overestimate the current network size, and conversely, the peer will underestimate the network size in a sparse keyspace. In order to address this bias, we employ a bias correction technique.

Correcting Local Keyspace Density Bias. The bias correction technique tries to strike a balance between not discarding valuable data points and correcting the bias sufficiently. Note, we cannot change the routing maintenance task itself and can, instead, select which queries we utilize from its overall set. Our biased sampling works by weighing data points exponentially less if they fall into a non-full bucket. The assumption is that if a bucket is full, the key space that the bucket covers is occupied by enough peers to get a representative sample. We define the weighing function w as follows:

$$w(b) = 2^{l(b)-k_{\text{max}}} \tag{2}$$

where b is the bucket number with $b \in [0, L)$, $l(b)$ is the level of bucket b at the time the closest peers are tracked with $l(b) \in [0, k_{\text{max}}]$, and k_{max} is the maximum level a bucket can have. In the case of IPFS $k_{\text{max}} = 20$. Put simply, non-full buckets will contribute exponentially less to the overall network size estimation with decreasing levels. This bias correction, therefore, alleviates the aforementioned problem of tracking the same peers repeatedly. Levels for high-numbered buckets will be zero, so their data points would contribute a negligible amount to the overall estimation.

For practical implementation, we limit the data collection to a rolling $t_{\text{window}} = 2\,\text{h}$ time window and require a minimum of $S_{\text{min}} = 16$ look-ups and a maximum $S_{\text{max}} = 192$ to bound the memory usage of the algorithm[5]. All three are tunable parameters and can affect the accuracy of the estimation.

C. Calculating Target Key Proximity of Individual Peers

Recall the Optimistic Provide algorithm used the above network size estimation to calculate a maximum distance to a target key that an individual peer must not exceed to be optimistically considered among the 20 closest peers, with a certain probability $p_{\text{individual}}$. This allows the initiator to immediately store records with peers as it learns about them during the DHT walk and thus expediting the process. We next describe how this maximum distance threshold is calculated.

Threshold Calculation. We have shown above how Beta functions can describe the distribution of peer proximities to target keys. The CDFs of the above Beta distributions can be interpreted as functions that represent the probability of finding the i-th closest peer within a certain distance to a target key. CDFs of Beta distributions are given by normalized incomplete beta functions $I_x(\alpha, \beta)$ with

$$I_x(\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^x t^{\alpha-1}(1-t)^{\beta-1}\text{d}t \tag{3}$$

for $0 \leq x \leq 1$ where Γ is the gamma function [19, (8.17.1)]. α and β are the same shape parameters as above and dependent on N and i. This means the inverse of this function $I_x^{-1}(\alpha, \beta)$ will yield the maximum distance to a target key in which one finds the i-th peer for a certain probability p given the network size N. In this case, we need to compute x such that the equation

$$p = I_x(\alpha, \beta) = I_x(i, N - i + 1) \tag{4}$$

is satisfied. $I_x^{-1}(\alpha, \beta)$ will then be the distance threshold in question. As stated above, $I_x^{-1}(\alpha, \beta)$ yields the maximum distance to a target key in which one finds the i-th peer with probability p. In our case, we want to find the maximum distance until we do not find the 20-th peer with probability $p_{\text{individual}}$. This means conversely, if a peer falls below this distance, it will be among the 20 closest peers with probability $p_{\text{individual}}$. Therefore, we compute x in Eq. 4 for $p = 1 - p_{\text{individual}}$, $i = 20$, and the current network size estimate to derive the distance threshold from I_x^{-1}. In our evaluation below, we chose $p_{\text{individual}} = 0.9$. Thus, we calculate the maximum distance value in such a way that we can be $90\,\%$ certain that any peer within that range is, indeed, among the 20 closest peers, based on our estimated network size. $p_{\text{individual}}$ is a tunable parameter and can be chosen freely.

D. Calculating Target Key Proximity of a Set of Peers

In Section III, we identified that prematuring terminating the DHT walk during a `PUT` operation addresses the main performance bottleneck. Recall that the query initiator can

[5] $S_{\text{min}} \widehat{=}$ one full r.t. refresh, $S_{\text{max}} \widehat{=}$ one full r.t. refresh every $10\,\text{min}$ for $2\,\text{h}$ (see Section II)

leverage the network size estimate to establish a supplementary termination condition based on the proximity of its currently known 20 closest peers to the target key to prematurely end the DHT walk. In this section, we elaborate on this condition and the precise methodology we employ for its computation.

Threshold Calculation. Unlike in the previous section, where we calculate the maximum distance for an individual peer, the DHT walk termination condition now considers the distances of the **set** comprising the 20 closest known peers. As the initiator progresses on its DHT walk and discovers new peers, it continuously verifies whether the average distance of this set of 20 closest peers exceeds a maximum distance. If it does, the initiator prematurely terminates the query. We observe that the expected value of the average distance of the 20 closest peers to the target key is equivalent to the expected distance of the hypothetical 10.5-th closest peer. Based on this insight, we proceed to calculate $I^{-1}x(\alpha, \beta)$ using the latest network size estimate N, $i = 10.5$ and probability p_{set}. This calculation yields the distance within which we can expect to find the 10.5-th peer or the average of the 20 closest peers with a probability of p_{set}. In our evaluation below, we choose $p_{\text{set}} = 0.9$. Note, p_{set} is also a tunable parameter.

V. Evaluation

We have implemented, merged, and released the above algorithms in the `go-libp2p-kad-dht` repository that implements the Kademlia protocol which is used by the IPFS reference implementation, Kubo, and also the Filecoin reference implementation, Lotus. Kubo officially supports our feature since v0.20.0. Using this implementation, we next evaluate our approach on both the IPFS record indexing DHT and the Filecoin DHT.

A. Performance

We first measure our `PUT` performance and compare it with the current implementation as the baseline, which in the following we call the *Classic* approach. We specifically evaluate (*i*) the `PUT` performance with different control flow strategies (see Section IV-A), and (*ii*) the regional performance dependence in a globally distributed study.

PUT Performance. Figure 2 shows `PUT` latencies in the IPFS (left) and Filecoin (right) networks using the classic and optimistic approaches with the *done* and *return* control flow strategies (see IV-A). The measurements were performed from two servers in central Europe over a duration of two days. The graphs show the number of launched queries for either approach in the lower right corner. To calculate the classic *return* latencies, we record the time until we receive the 15th response from the final record storage remote procedure call (RPC) to provide a fair comparison to the *Optimistic done* strategy.

The results show that the *Optimistic Provide* approach with the early return strategy is the fastest in both networks. Even the *Optimistic Provide done* strategy is always faster than the classic approach with either strategy. Table I shows the latency values for the 50th, 90th, and 95th percentiles and

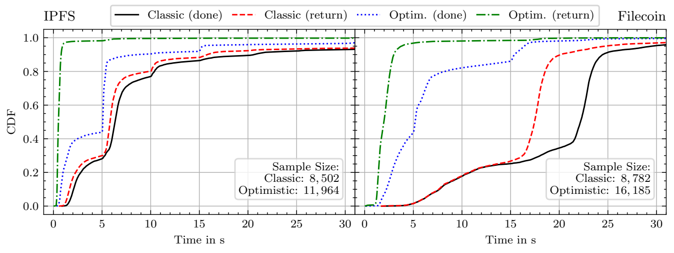

Fig. 2: `PUT` latencies in the IPFS (left) and Filecoin (right) networks using the classic and optimistic approaches with the *done* and *return* strategies (see IV-A)

TABLE I: `PUT` latencies in the IPFS and Filecoin networks for different approach and control flow strategy combinations.

	Return Strategy			*Done* Strategy		
Percentile	50th	90th	95th	50th	90th	95th
IPFS						
classic	5.8 s	15.8 s	43.0 s	6.3 s	20.6 s	49.8 s
optim.	0.51 s	0.80 s	0.92 s	5.1 s	8.7 s	15.4 s
speed-up	11.3 ×	19.7 ×	46.7 ×	1.2 ×	2.3 ×	3.2 ×
Filecoin						
classic	17.5 s	20.1 s	24.8 s	22.2 s	24.5 s	29.5 s
optim.	2.0 s	3.0 s	3.8 s	5.2 s	15.4 s	16.4 s
speed-up	8.7 ×	6.7 ×	6.5 ×	4.2 ×	1.5 ×	1.7 ×

the corresponding speed-ups between both approaches for each strategy. The largest latency difference is between the *Optimistic Provide return* and the Classic *done* combinations, which is also the latency improvement that users will experience if they adopt Optimistic Provide. The speed-ups here are 12.3x, 25.7x, 54.1x in the IPFS network and 11.1x, 8.1x, 7.7x in the Filecoin network for the 50th, 90th, and 95th percentiles, respectively.

Regional PUT Performance. To assess how our approach performs in different regions, we measure its performance across the seven AWS regions. In each region, we deploy a peer that performs `PUT` operations with the optimistic *return* approach and another one with the Classic *done* approach. For each approach, we configured a privileged node that sequentially instructs each peer in rounds to perform a `PUT` operation with a randomly generated record and all remaining peers to retrieve the record (`GET` operation). Doing this allows us also to measure record retrievability rates. Due to resource constraints, we only perform this experiment in the IPFS network (not Filecoin). We perform a total of $10,753$ and $18,756$ probes with the classic and optimistic approach, respectively, across all regions.

Figure 3 shows the 50th, 90th, and 95th `PUT` latency percentiles from each AWS region. The number of probes from each region for each approach is at the top of the graph. We confirm that in every region, our proposed approach outperforms the defined baseline significantly. Further, in Table II, we evaluate the retrievability rates. It shows the total number of `PUT` but also `GET` operations we performed with either approach and the errors we encounter. The table shows that the error rates of our proposed approach are similar to the baseline.

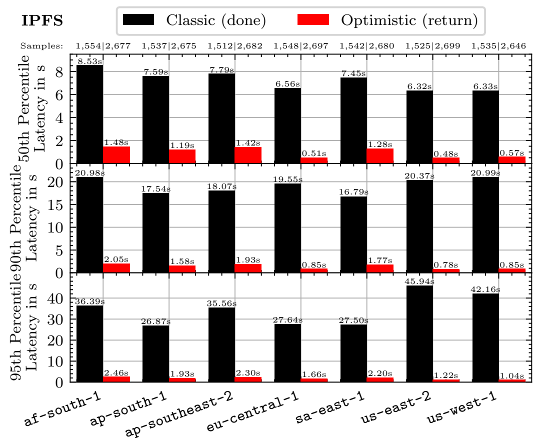

Fig. 3: `PUT` latency percentiles of the IPFS network across AWS regions. Data collected from 2023-06-26 to 2023-06-29.

The errors we encountered for `PUT` operations are exclusively DHT walk timeouts which we configured to be 3 min, and for `GET` operations, the errors are unretrievable records. The `GET` errors correspond to two unresolvable records for the classic and one for the optimistic approach.

B. Performance Exploration

We next focus on the performance of the specific sub-components to understand how they contribute to the above results.

Estimation Accuracy. To assess the network size estimation accuracy, we performed two 1.5 months long-term measurements in the IPFS and Filecoin networks with six peers in each network that continuously participate in either DHT. These peers continually estimate the network size based on data from routing table maintenance tasks. We define full network crawls as the baseline to compare the results against. Therefore, we overlay relevant graphs with the total number of peers our crawls have found in the DHT plus their fraction our crawler was able to dial. We consider network crawls to provide a robust and comprehensive view of all peers participating in a DHT network.

In Figure 4b and 4c, we observe that all peers consistently identify the new network size. The inset graph in Figure 4b shows a period where the response of the network size estimation to a sudden network topology change is observable.

Both graphs confirm the accuracy of the estimations but also show that the peers systematically underestimate the total number of peers in both networks compared to our defined baseline. Individual peers, on average, estimate the network size to be 6.0 % and 13.5 % lower than our defined baseline in the IPFS and Filecoin networks, respectively. This observation can be attributed to the previously mentioned backtracking behavior. In our proposed approach, routing table refreshes continue to use the classic approach to find the 20 closest peers to a target key to find suitable peers for its buckets. Because of the backtracking behavior, the sets of peers that feed into the network size estimation calculation exclude the closest peers

TABLE II: Regional `PUT` performance sample sizes and error counts aggregated across all seven AWS regions.

	Classic `PUT`	Classic `GET`	Optim. `PUT`	Optim. `GET`
Errors	91	12	7	6
Total	10,753	64,515	18,756	112,531

that are unreachable. Consequently, this creates an artificially sparser keyspace, leading to a lower network size estimation. Because the share of undialable peers in the Filecoin network is, on average double the share of the undialable peers in the IPFS network, the relative underestimation is equally higher in the Filecoin. As we have shown above, these underestimates do not have a significant impact on performance, yet if more accurate estimates are required we note that the underestimations exhibit consistency across measurement points, suggesting that a deployment-wide scaling factor could correct for these errors.

Bias Correction Efficacy. The bias correction is intended to overcome the limitations of reusing the routing table maintenance queries. As above, we deployed six peers in both networks and let them perform their ordinary routing table maintenance tasks. Figure 4a presents the accuracy of the bias correction technique proposed in Section IV-B for the IPFS and Filecoin networks. The graphs show the network size estimations overlayed with the baseline results from network crawls over 3.5 d in June 2023. The graphs on the left show network size estimations without any bias correction applied, and the graphs on the right show the same data points but with a weight applied according to Equation (2).

The results confirm that the bias correction achieves a convergence of the network size estimations across all six peers within a network. The standard deviation between the weighted network size estimations among the six peers in each network is less than 5 % of their average size estimate and decreased by 22.6 % in the IPFS network and by 61.0 % in the Filecoin network in the 90th percentile during the measurement period.

These results show that by leveraging the routing table refresh queries as the data source, the introduced bias can be effectively mitigated. Consequently, this enhancement enables the network size estimation algorithm to operate without incurring any additional networking overhead.

Selected Peers to Target Key Proximity. As the network size estimation algorithm allows peers to calculate target key distance thresholds, this subsection's objective is to confirm that the chosen peers responsible for holding the record exhibit comparable target key proximity compared to the baseline. By ensuring this condition, we can be confident that any other peers seeking to retrieve the record will successfully locate it within the network.

Figure 5 shows CDFs of the selected peer's distances to the target keys. This distribution is susceptible to network size changes, which we verified stayed almost constant during the measurement time. We confirm that the average distance of all selected peers in the optimistic approach differs less than 0.3 %

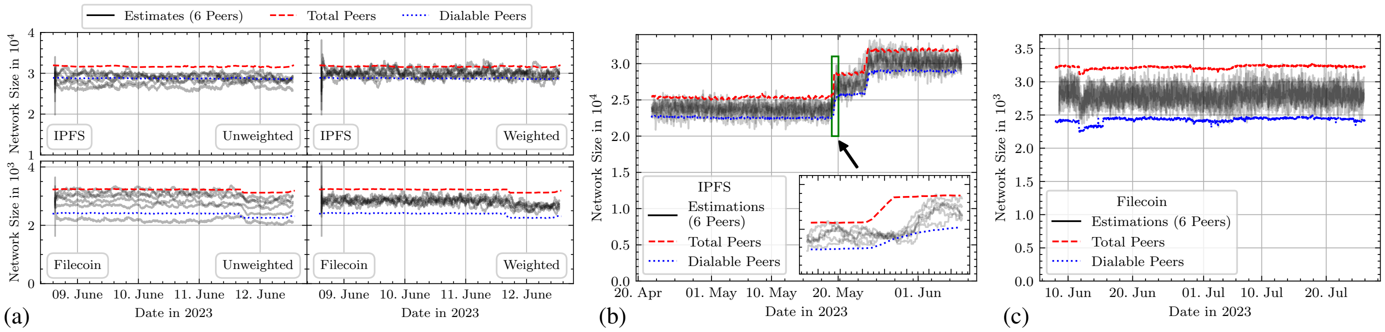

Fig. 4: a) Efficacy of the bias correction technique. Each row shows the same data for the IPFS and Filecoin networks. The columns show unweighted data points (left) and weighted data points (right). b) and c) Long-term measurement of network size estimations from 6 individual peers in the IPFS and Filecoin networks, respectively. All graphs are overlayed with the total number of peers and their dialable fraction from network crawls. The inset graph in b) shows the reaction of the estimation algorithm to a network size increase.

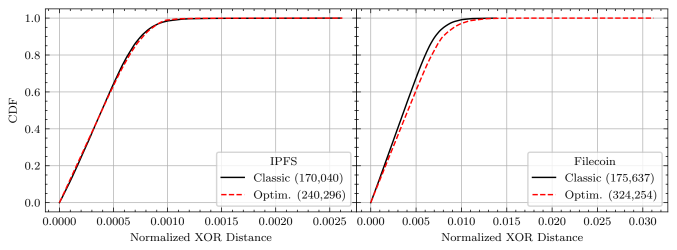

Fig. 5: CDFs of distances of selected peers to hold provider records to the target keys for the IPFS (left) and Filecoin (right) networks.

from the classic approach in the IPFS network and 10.7 % in the Filecoin network. For the IPFS network, we can confidently assert that the retrievability remains uncompromised supporting earlier results in Table II

C. Overhead

We finally measure the overhead of the mechanism. We do so from two angles. Due to the absence of the backtracking mechanism filtering out unreachable peers, the optimistic approach is expected to display a higher overhead in failed RPCs. Yet, we also conjecture that prematurely terminating the DHT walk may reduce traffic. While we acknowledge that the network size estimation algorithm incurs additional computational costs, our experiments did not reveal a significant impact.

ADD_PROVIDER RPC Success Rate We first inspect the `ADD_PROVIDER` RPC success rate, as failed RPC calls are a clear form of wasted overhead. This is the RPC issued to each of the closest peers in the follow-up phase to store the record. A low success rate implies reduced record replication and heightened vulnerability to peer churn.

Figure 6a shows the number of successful `ADD_PROVIDER` RPCs per `PUT` operation in the IPFS and Filecoin networks. The measurement methodology and dataset are identical to the one in Section V-A-*PUT Performance*. Because the record replication factor in both networks is set to 20, the optimal distribution would be a single bar at 20 successful RPCs per

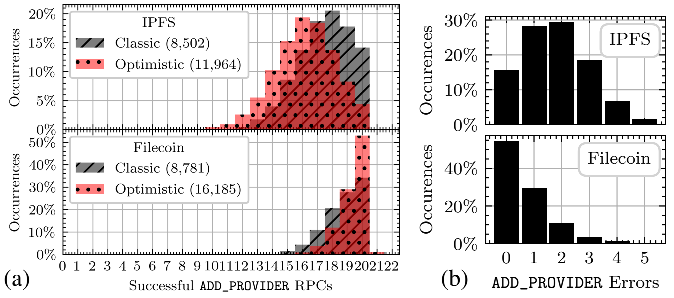

Fig. 6: a) Distributions of the number of successful `ADD_PROVIDER` RPCs per provide operation. The numbers in parenthesis show the number of `PUT` operations. b) `ADD_PROVIDER` RPC errors in the asynchronous phase of the Optimistic Provide approach in the IPFS and Filecoin networks. The sample size is $11,964$ and $16,185$, respectively.

`PUT` operation. However, in a permissionless network where peers can join and leave the swarm at any time, this is rare.

The data shows that the centers of masses are 16.1, 17.5 in the IPFS, and 19.3, 18.7 in the Filecoin network for the optimistic and classic approaches, respectively. This means our proposed solution performs worse in IPFS but better in the Filecoin network. This is in line with the result in Section V-B, where we found that in the Filecoin network, our approach selected peers that are slightly further away from the target than the baseline. These peers are more likely to be reachable.

Figure 6b shows the number of `ADD_PROVIDER` RPC errors per `PUT` operation of the asynchronous phase after control has been handed back to the user for the optimistic approach for the IPFS and Filecoin networks. This graph shows that in almost 85 % and almost 50 % of the `PUT` operations, at least one of the five RPCs fails and would delay the whole operation. This shows that the adaptation of the *return* control flow strategy has a significant benefit. We justify the asynchronous phase, wherein additional network requests are in-flight but remain invisible to the user, with the notably reduced networking overhead achieved through our optimistic approach, as demonstrated in the following section.

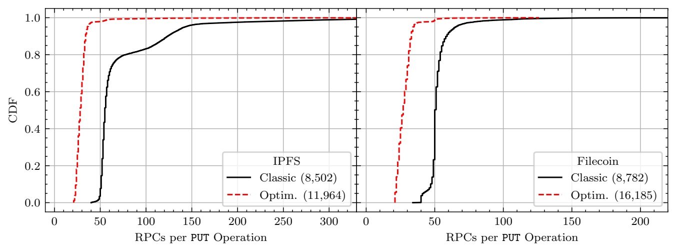

Fig. 7: CDFs of issued RPCs per PUT operation during DHT walk and follow-up phases for the IPFS (left) and Filecoin (right) networks.

Networking Overhead. Last, we evaluate the networking overhead using the number of issued RPCs per PUT operation by reusing the dataset from Section V-A-*PUT Performance*. Figure 7 shows CDFs of the number of issued RPCs per PUT operation. This includes all RPCs in both the DHT walk and follow-up phases, which means the theoretical minimum is 20 RPCs (because the follow-up phase always consists of at least that many). In both networks, the optimistic provide approach utilizes fewer networking resources. It issues 28, 34, and 36 RPCs in the IPFS indexing DHT, and 28, 34, and 36 RPCs in the Filecoin DHT for the 50th, 90th, and 95th percentile, respectively. The classic approach issues 55, 124, 143 RPCs in the IPFS and 50, 59, 69 RPCs in the Filecoin network for the same percentiles. Our approach uses at most 56 % the networking resources than the baseline in both networks across all considered percentiles. This confirms that, overall, the Optimistic Provide requires lower network traffic.

VI. Related Work

Decentralized Web. The IPFS network has expanded in parallel with other Decentralized Web technologies, specifically the “fediverse” - a collection of server-based federated services like Mastodon [20], Pleroma [21], and Diaspora [22]. Nonetheless, it’s important to note that these fediverse applications depend entirely on the uptime of federated servers [20], while IPFS operates without such servers.

P2P Networks. Numerous P2P overlay architectures have been developed, and several DHT structures have been proposed, including Chord [23], Tapestry [24], Koorde [25], Pastry [26], and others [27]. These architectures have served as a foundation for building different applications, such as large-scale content delivery platforms [28] and decentralized social networks [29], among other use cases. Instead of developing an entirely new system, IPFS has embraced the Kademlia DHT for content indexing [12]. By leveraging and expanding on these existing technologies, IPFS has become one of the most extensive implementations of the ”Decentralized Web” on a global scale. Another prominent example of utilizing Kademlia for large-scale deployment is BitTorrent [28]. We note that the Optimistic Provide technique is equally useful for these deployments. In its pursuit of decentralization, IPFS aims to resist censorship, taking inspiration from platforms like Freenet [30] and Wuala [31]. To achieve censorship resistance, these platforms store encrypted content across a random subset of peers. On the other hand, IPFS adopts a BitTorrent-like approach, where nodes only retain the content they have a specific interest in.

DHT Evaluation & Optimization. The study that most closely resembles our work measures the operational performance of IPFS [5], and we draw insights from this paper to inform our own methodologies. Numerous evaluations have been conducted to assess the performance of other P2P systems too. For instance, [32] and [33] assess the implementation of Kademlia in BitTorrent and discover a notable number of failed nodes, leading to adverse effects on lookup times. Additionally, Stutzbach and Rejaie [34] create a model for evaluating Kademlia’s performance and propose various enhancements. In an effort to enhance the performance and usability of DHTs, several approaches have been explored, including caching [35], network-aware peer selection [36], and parallelizing lookups [34].

Network Size Estimation. Our approach relies on accurate DHT network size estimation. Various approaches have been proposed previously. Some of these methods necessitate a custom protocol [37], which in turn requires peers to upgrade their installation. Others involve explicit sampling of a key space area, leading to increased networking overhead [38]. A comparative study [39] evaluates three generic counting algorithms: Sample & Collide [40], HopsSampling [41], and Gossip-based Aggregation [42]. The approach proposed in this paper builds atop [17]. We distinguish ourselves from previous methods by offering a lightweight mechanism that is feasible for individual peers to compute without requiring peers to collaborate.

VII. Conclusion

This paper has presented the design, implementation, and evaluation of an accelerated DHT PUT operation for IPFS. We show that the PUT operation speed-up surpasses one order of magnitude in various percentile and region combinations, while maintaining record availability and reducing network overhead by over 40 %. Notably, our approach complements the already satisfactory GET performance in IPFS [5] by enabling sub-second PUT latencies in over 90 % of operations from North America and central Europe [9]. We have deployed our solution in the public IPFS codebase, and our work now enables a wider spectrum of delay-intolerant applications within the IPFS ecosystem. We further emphasize that our technique can be applied to any Kademlia DHT implementation, extending its impact beyond the IPFS networks.

There are a number of avenues for future work. We plan to investigate the impact of severe network conditions such as network partitioning, high churn, or Sybil attacks on our approach, and the components it comprises. Moreover, developing a more robust understanding of the key factors influencing the variations in RPC success rates between the IPFS and Filecoin network warrants deeper investigation.

References

[1] T. V. Doan, R. van Rijswijk-Deij, O. Hohlfeld, and V. Bajpai, "An empirical view on consolidation of the web," *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 3, pp. 1–30, 2022.

[2] C. Bommelaer de Leusse and C. Gahnberg, "The global internet report: Consolidation in the internet economy," *Internet Society*, 2019.

[3] "Ethereum Name System," 2023. [Online]. Available: https://ens.domains/

[4] "Unstoppable Domains," 2023. [Online]. Available: https://unstoppabledomains.com/

[5] 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," ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 739–752. [Online]. Available: https://doi.org/10.1145/3544216.3544232

[6] J. Benet, "IPFS-content addressed, versioned, P2P file system," *arXiv:1407.3561*, 2014.

[7] "Mastodon," 2023. [Online]. Available: https://joinmastodon.org/

[8] "Bluesky," 2023. [Online]. Available: https://blueskyweb.xyz/

[9] "IPFS KPIs," 2023. [Online]. Available: https://probelab.io/

[10] "Ipfs ecosystem directory," https://ecosystem.ipfs.io/, 2022.

[11] "How to host dynamic content on ipfs," https://blog.ipfs.tech/2023-how-to-host-dynamic-content-on-ipfs/, 2023, accessed: 2023-06-21.

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

[13] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding I/O performance of IPFS storage," in *Proceedings of the International Symposium on Quality of Service*. ACM, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3326285.3329052

[14] "Filecoin: A Decentralized Storage Network," 2017. [Online]. Available: https://filecoin.io/filecoin.pdf

[15] "Measure performance with the RAIL model," 2023. [Online]. Available: https://web.dev/rail/

[16] M. Cortes-Goicoechea and L. Bautista-Gomez, "Rfm 17 — provider record liveness," https://github.com/protocol/network-measurements/blob/master/results/rfm17-provider-record-liveness.md, 2022.

[17] "A new method for estimating p2p network size," https://eli.sohl.com/2020/06/05/dht-size-estimation.html, 2020, accessed: 2023-06-02.

[18] J. E. Gentle, *Computational Statistics*. Springer New York, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-98144-4

[19] "*NIST Digital Library of Mathematical Functions*," https://dlmf.nist.gov/, Release 1.1.10 of 2023-06-15, f. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds. [Online]. Available: https://dlmf.nist.gov/

[20] A. Raman, S. Joglekar, E. D. Cristofaro, N. Sastry, and G. Tyson, "Challenges in the decentralised web: The mastodon case," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 217–229.

[21] A. I. Hassan, A. Raman, I. Castro, H. B. Zia, E. De Cristofaro, N. Sastry, and G. Tyson, "Exploring content moderation in the decentralised web: The pleroma case," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, 2021, pp. 328–335.

[22] B. Guidi, M. Conti, A. Passarella, and L. Ricci, "Managing social contents in Decentralized Online Social Networks: A survey," *Online Social Networks and Media*, vol. 7, 2018.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[24] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, 2004.

[25] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *International Workshop on Peer-to-Peer Systems*. Springer, 2003, pp. 98–107.

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001, pp. 329–350.

[27] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Privacy-preserving p2p data sharing with oneswarm," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 111–122. [Online]. Available: https://doi.org/10.1145/1851182.1851198

[28] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6. Berkeley, CA, USA, 2003, pp. 68–72.

[29] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "LifeSocial. KOM: A secure and P2P-based solution for online social networks," in *CCNC*, 2011.

[30] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing privacy enhancing technologies*. Springer, 2001.

[31] T. Mager, E. Biersack, and P. Michiardi, "A measurement study of the wuala on-line storage service," in *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2012.

[32] S. A. Crosby and D. S. Wallach, "An analysis of bittorrent's two kademlia-based dhts," Rice Technical Report, Tech. Rep., 2007.

[33] S. Wolchok and J. A. Halderman, "Crawling BitTorrent DHTs for fun and profit," in *4th USENIX Workshop on Offensive Technologies (WOOT 10)*. Washington, DC: USENIX Association, Aug. 2010. [Online]. Available: https://www.usenix.org/conference/woot10/crawling-bittorrent-dhts-fun-and-profit

[34] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, 2006, pp. 1–12.

[35] O. Saleh and M. Hefeeda, "Modeling and caching of peer-to-peer traffic," in *Proceedings of the 2006 IEEE International Conference on Network Protocols*. IEEE, Nov. 2006. [Online]. Available: https://doi.org/10.1109/icnp.2006.320218

[36] S. Kaune, K. Pussep, C. Leng, A. Kovacevic, G. Tyson, and R. Steinmetz, "Modelling the internet delay space based on geographical locations," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2009, pp. 301–310.

[37] N. Evans, B. Polot, and C. Grothoff, "Efficient and secure decentralized network size estimation," vol. 7289, 05 2012, pp. 304–317.

[38] S. Mane, S. Mopuru, K. Mehra, and J. Srivastava, "Network size estimation in a peer-to-peer network," 2005.

[39] E. Le Merrer, A.-M. Kermarrec, and L. Massoulie, "Peer to peer size estimation in large and dynamic networks: A comparative study," in *2006 15th IEEE International Conference on High Performance Distributed Computing*, 2006, pp. 7–17.

[40] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: Random walk methods," ser. PODC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 123–132. [Online]. Available: https://doi.org/10.1145/1146381.1146402

[41] D. Psaltoulis, D. Kostoulas, I. Gupta, K. Birman, and A. Demers, "Practical algorithms for size estimation in large and dynamic groups," 01 2004.

[42] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, 2004, pp. 102–109.