Enhancing IPFS Bitswap

Christos Karapapas*, George Xylomenos*, George C. Polyzos*†
*Mobile Multimedia Laboratory, Athens University of Economics and Business, Greece.
Email: {karapapas,xgeorge,polyzos}@aueb.gr
†School of Data Science, The Chinese University of Hong Kong, Shenzhen (CUHK-Shenzhen), China.

***Abstract*—The InterPlanetary File System (IPFS) holds a pivotal role in the storage layer of the decentralized Web, commonly known as Web3. Its efficient functioning is crucial for a plethora of applications linked to blockchains, NFTs, and similar technologies. A key component, Bitswap, plays a central role in locating and exchanging files within the IPFS network. In this paper, we measure the latency of IPFS in fulfilling download requests and introduce an enhanced version of Bitswap designed to optimize its performance. We take into consideration the popularity of files and perform a series of experiments comparing the baseline version with our proposed method. Our findings indicate that, with the proposed improvements, the latency of download requests is significantly reduced, especially for content with low popularity. We also compare the two versions as network size grows, and demonstrate that our proposed method enhances the scalability of IPFS, as it keeps latency low without requiring node state to grow in proportion to the network size.**

***Index Terms*—IPFS, Bitswap, KAD-DHT**

I. Introduction

The Web3 concept has gained significant attention recently. One of the main attractions of Web3 is an emphasis on returning control of data to their owners, allowing them to determine who can access their data and enabling monetization of the information they generate. Central to achieving these objectives are distributed ledger and blockchain technologies, along with token-based economics. Web3 also envisions decentralized services that cater to the *Internet of Things* (IoT) era, introducing financial elements to the user-Web relationship through cryptocurrencies. Web3 is often conceptualized as comprising different stacks, each consisting of various protocols that collaborate to deliver services to users. These protocols cover areas such as data storage, name resolution, decentralized identities, and, at a higher level, services like social media, gaming, and marketplaces. However, these protocols, along with the bridges connecting them, are still evolving, and their efficiency is not yet as fine-tuned as that of traditional Web protocols. This can lead to poor user experience in various applications, from a game with collectible NFTs to a cross-chain transaction.

The *InterPlanetary File System* (IPFS) is considered to be a cornerstone of the Web3 storage layer, with over 3 million weekly users, 300,000 of which are unique nodes [1], and more than 10,000 sites built on top of it.[1] The fact that companies embracing Web3, such as Pinata and Cloudflare, offer public gateways for users to access IPFS via HTTPs is not a coincidence. IPFS' core component is a Kademlia-based Distributed Hash Table (DHT). Trautwein et al. [1], illustrate that IPFS faces considerable user churn during its operation, negatively impacting the efficacy of its DHT and, especially, its latency.

To limit the DHT's impact on IPFS, a different mechanism called Bitswap is utilized before resorting to the DHT. Bitswap announces to all connected nodes that the initiating node is searching for specific content. If a 1 sec deadline elapses without success, the search request is then redirected to the DHT. In a recent study[2] focusing on popular content, i.e. content that has previously been requested, with more than 800 nodes connected to the requester, Bitswap managed to fulfill 80% of retrieval requests within the specified time frame. Therefore, it is clear that Bitswap has a crucial role in the functioning of IPFS.

Despite its advantages, Bitswap has its issues. For instance, multiple studies [2], [3], [4] have identified shortcomings in Bitswap, especially concerning the leakage of information. Furthermore, there are concerns about Bitswap's operation under suboptimal conditions, such as when a node has a low number of connections or the content being sought is not widely popular. Confais et al. in [5] discovered that 30% of files in IPFS have fewer than 3 replicas, with the average file having 4.6 replicas. Overall, their research revealed that file popularity on IPFS follows a Zipf distribution, where a tiny fraction of files are very popular and have many replicas, contrasted with the vast majority of files that are less popular and have a few replicas. However, their findings indicate that the majority of queries on IPFS are for files that are not popular.

Swift service provision is imperative for IPFS-dependent applications to meet their demands. To enhance IPFS efficiency, in this work, we propose a method to improve the effectiveness of Bitswap, thus reducing dependence on the, slower, DHT. To this end, we initially conducted a sequence of experiments to assess the DHT's response time to a query aiming to locate providers for content that we had previously added to the network, thus making it unique. Our findings from 1800 experiments reveal that the median response time is 8 seconds, which is non-negligible. We then introduced our enhancements to boost the chances that a content search will only use Bitswap. This strategy aims to decrease the total response time to ≤ 1 second. Our findings demonstrate that our method significantly increases the likelihood of finding

[1]https://trends.builtwith.com/framework/IPFS

[2]https://www.youtube.com/watch?v=zppddk2O9UQ

content via Bitswap, especially for content that is not widely popular.

The remainder of this paper is structured as follows: In Section II, we present background information and an overview of IPFS technologies. In Section III, we present related research regarding Bitswap and IPFS efficiency. In Section IV we measure response times through the DHT. Then, in Section V, we present the proposed method and in Section VI evaluate it. Finally, in Section VII, we summarize our findings and contributions, and explore future research directions.

II. Background on IPFS

The *InterPlanetary File System* (IPFS) [6] is a decentralized file-sharing system with a focus on distributed data storage and quick file distribution. Unlike traditional file systems, IPFS uniquely identifies files based on their content, assigning each file a distinct *Content Identifier* (CID). A key IPFS component is libp2p, an open-source library of network protocols, which include KAD-DHT, a scalable variant of the Kademlia DHT. KAD-DHT manages mappings such as Provider Records (indicating which peer has a piece of content), Peer Records (identifying the network address of a peer), and IPNS records (mapping static to dynamic data). Bitswap operates as the data-exchange protocol, relying on "want-have" content and "have" content messages to facilitate efficient data exchange.

IPFS employs Merkle DAGs, a combination of Merkle Tree and *Directed Acyclic Graph* (DAG), to certify the uniqueness of exchanged data, ensuring no duplicates are stored. When a user intends to upload a file to IPFS, the process involves breaking the original file into smaller chunks, typically 256 KB each. Each chunk is then assigned a unique CID, these chunks are organized into a Merkle DAG, and a Provider Record is generated, containing the root CID of the file. Subsequently, a query is sent to the DHT. The DHT is utilized to find the $k = 20$ closest peers to the CID in the network. The Provider Record (which peer has the file) is stored in these peers. Independently, 20 more peers are found to store the Peer Record (what is the address of that peer) [1].

In IPFS, each peer manages a network of active connections known as the *swarm*, with a default size ranging from 600, referred to as *low water mark* to 900, referred to as *high water mark*. When a user wishes to retrieve a file from the IPFS network, the Bitswap protocol is activated. It sends a message to the user's swarm peers in the format `want-have <root CID>` [7]. Peers in the swarm individually check if they have the specified CID locally. If a peer possesses the requested content, it responds with a `have` message. Upon receiving a `have` response, a dedicated session is initiated for the specific CID. All peers responding with `have` messages are also included in this session. Subsequent communication within the session only involves these peers.

If no response is received within 1 second, the process is handed over to the DHT, which operates in two stages. Initially, the process searches for the Provider Record which contains the Peer ID storing the content for the requested CID. Subsequently, it looks for the Peer Record which shows how the Peer ID is linked to a network address. Once this process is finalized, Bitswap is reactivated to facilitate the data exchange with the peer hosting the content [1].

III. Related Work

In [7] de la Rocha et al., initially, explain in detail the steps taken by Bitswap from requesting a file until acquiring it. Moreover, they propose a number of modifications to Bitswap, aiming to improve its efficiency and efficacy. Their first proposal makes each node inspect `want-have` messages, and rather than discarding messages for CIDs that it does not have, maintaining a record for the CID, indicating the peers that have requested it. The rationale behind this approach is that if peers have requested a CID previously, they are likely to have it later on. Consequently, when a user seeks a CID, it begins by requesting it from these identified peers. This method aims to reduce the RTT by one cycle and minimize the overall volume of messages exchanged through Bitswap. Their second proposal adds a *Time-to-Live* (TTL) parameter into Bitswap messages. When a node receives a Bitswap message with TTL greater than zero, it functions as a relay, forwarding the message to connected nodes that have not yet received it. The objective is to diminish the likelihood of Bitswap timing out.

The `want-have` inspection technique shares the same objective as our proposed method. The idea is that a node will issue a `want-block` message to nodes it knows have also made a similar request. Ideally, these nodes would respond with the block, bypassing the `want-have` step. However, due to the frequent changes within the IPFS network, there is a likelihood that they might not respond, leading to a fallback on Bitswap's standard operation and incurring an additional RTT. Moreover, their method maintains a local data structure, using computing and storage resources, which escalate as the network expands. Our approach does not need a special data structure; it merely involves a read operation in the node's routing table, which we consider to be of negligible impact.

In [1] Trautwein et al., provide a comprehensive overview of IPFS's functionality, offering a broad perspective on its mechanics. Moreover, they assess IPFS using data gathered from three distinct sources. The first dataset was obtained by systematically crawling the IPFS network at 30-minute intervals over a span of 10 weeks. The second dataset pertains to the traffic monitored through one of IPFS's public gateways (ipfs.io). Finally, the third dataset is derived from active measurements conducted by the authors, who deployed six virtual machines, each one on a different region on AWS. The study highlights several key findings, including the geographical distribution of nodes, which indicates that IPFS is significantly decentralized. It also examines the network's dynamics, which contributes to high churn rates. In their analysis, the authors found that across the six AWS regions, the delay for publishing 50% of the sample is 33.8 seconds, whereas the delay for retrieving content is 2.9 seconds. Our measurements in Section IV present a sharp contrast, revealing a median time of 8.4 seconds, which significantly deviates from their findings. This divergence likely stems from the different conditions under which the data was requested. The

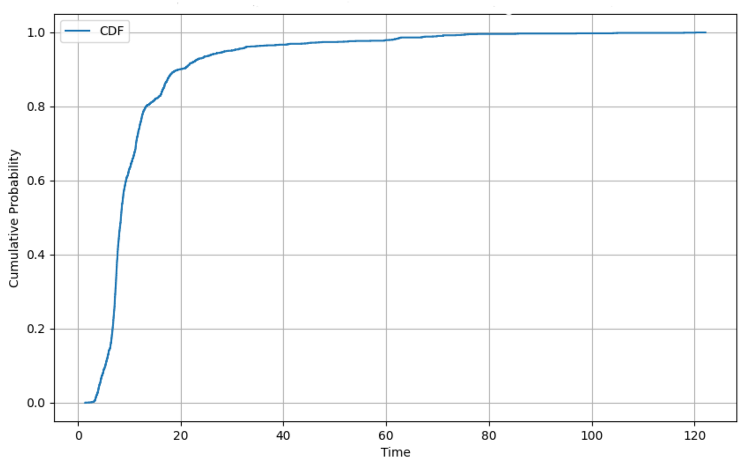

Fig. 1. Cumulative Distribution Function of response times (in seconds).

authors' nodes were acting as DHT server nodes, while our client node was situated behind a NAT and firewall, operating with limited network resources. Despite this, we contend that the conditions of our experiment more accurately mirror the typical user experience on IPFS.

IV. Lookup Latency Measurements

In our experimental setup, we employed two distinct nodes: a client and a server. The server was situated on a workstation with a public IP address and not confined behind a firewall. This server was responsible for generating random files, adding them to the IPFS network, and then adding the respective root CID on a database (to avoid direct communication between the nodes). The client was set up on a computer situated behind a NAT, using the operating system's default firewall. This setup was chosen to more accurately reflect the typical user experience when interacting with IPFS. The client node was utilized to retrieve each CID from the database and subsequently ask the IPFS network for the corresponding content through the DHT.

We measured the time it took for the DHT to respond to each inquiry. Specifically, the client, after the server added the file to the network, executed the command `ipfs dht findprovs <hash>` which returns up to 20 Peer IDs corresponding to providers of the requested file. This experiment was systematically repeated over 24 hours, with a total of approximately 1800 randomly generated files. By executing this series of experiments, we aimed to evaluate the efficiency and responsiveness of the IPFS network in handling requests for randomly generated files. The measured response times from the DHT provide insights into the performance of IPFS in the context of file retrieval for a substantial dataset.

To enhance the objectivity of our results, we excluded 105 experiments out of the initial 1800 that experienced timeouts, i.e., more than 2 minutes. By excluding timed-out experiments, we ensure that the analysis is based on a subset of data where interactions with the IPFS network were successfully completed. Among the experiments that were successfully served, the average response time was 12 seconds, with a median of 8.4 seconds. These values, regardless of the specific metric used, are non-negligible. The average and median response times provide insights into the typical performance observed during the retrieval of files from the IPFS network in our experimental setup. The Cumulative Distribution Function of the response times in this experiment is shown in Figure 1.

V. The `know` Message

In Section IV, we observed that DHT response times in IPFS are not negligible and can potentially impact the functionality of an IPFS-based service. To address this issue, we propose an enhancement to Bitswap aimed at reducing the likelihood of a query going unanswered by Bitswap, which would cause the requester to resort to the, much slower, DHT. The proposed enhancement involves introducing a new message type called `know` to the list of Bitswap messages.

The process unfolds as follows: when a client broadcasts a request for a root CID, the nodes within the swarm individually check if they locally store the file. If so, they respond with a `have` message, as usual. In addition, though, they check if they store a Provider Record for this CID, that is, if they know someone else that stores the content. If a Provider Record is found, the node responds with a message of the form `know PeerID`, where PeerID is the identifier of the node that, at some point, advertised that it provides the file. Subsequently, the client incorporates that Peer ID into the ongoing session.

A provider's PeerID is insufficient for establishing contact with it; a node also needs the network address of the peer, that is, its Peer Record. To avoid immediately resorting to a DHT walk to locate it, the client that has the Provider Record, also consults its address book to see if it stores a Peer Record, which, if found, is sent along with the `know message`. Consequently, when the requester receives the `know` message, they will be informed of the Peer ID and, ideally, also obtain the Peer Record, enabling them to link

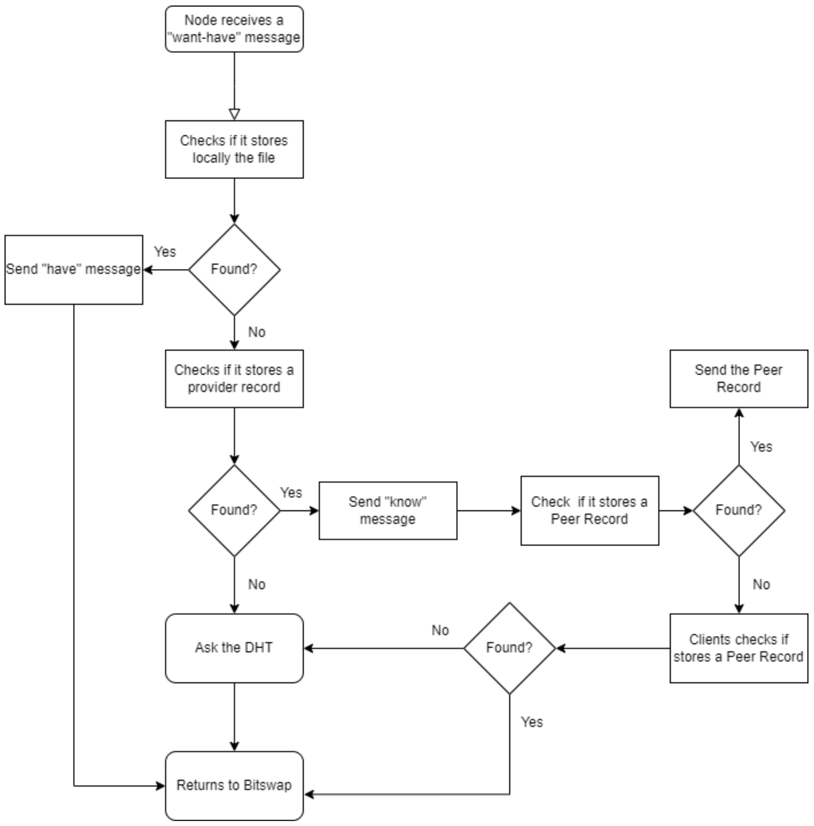
Node receives a "want-have" message
Checks if it stores locally the file
Found?
Yes
Send "have" message
No
Checks if it stores a provider record
Found?
Yes
Send "know" message
Check if it stores a Peer Record
Found?
Yes
Send the Peer Record
No
Clients checks if stores a Peer Record
Found?
No
Yes
No
Ask the DHT
Returns to Bitswap

Fig. 2. The operation flow of the proposed Bitswap enhancement.

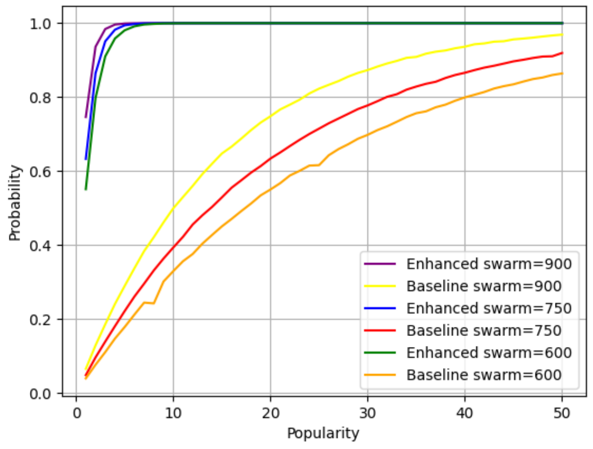

Fig. 3. The probability of finding content via Bitswap, against file popularity.

the Peer ID to a physical address. If this information is not available, the process transitions to the peer discovery stage, where the requester first checks its address book for a stored Peer Record for the specific Peer ID, before turning to the DHT for more information. The flow of the aforementioned procedure is depicted in Figure 2.

VI. Evaluation

To estimate the impact of our proposal scheme, we first need an estimate of the size of the IPFS network. For this reason we used the Nebula DHT Crawler [8]. These measurements show that during the measurement period the network size was about $\nu = 15,300$ nodes. Let p be the probability that a node storing the Provider Record of the original uploader, is also part of the client's swarm. We approach the problem as follows. We consider each of the broadcasted messages as an attempt to select at least one of the 20 peers that have the Provider Record. The probability of finding no one on the first try is $\frac{15280}{15300}$, the second is $\frac{15279}{15299}$ etc. So in general the probability of not selecting even one is $q = \prod_{i=1}^{\nu_s} \frac{15280-i}{15300-i}$, where ν_s is the swarm size. The probability we are looking for is $p = 1 - q$, which is the probability to find at least one.

We conducted a series of Monte Carlo experiments to compare our improved Bitswap version against plain Bitswap. The primary objective was to ascertain the probability of a query receiving a response from Bitswap in each version. We employed Python's `random` module, particularly the `random.sample()`[3] function, to generate random sets of nodes, thereby simulating the sets of swarm nodes and the set of 20 nodes that store the uploader's Provider Record. Our goal was to determine if there were common elements within these sets, indicating that a swarm node could respond with a

[3]https://docs.python.org/3/library/random.html

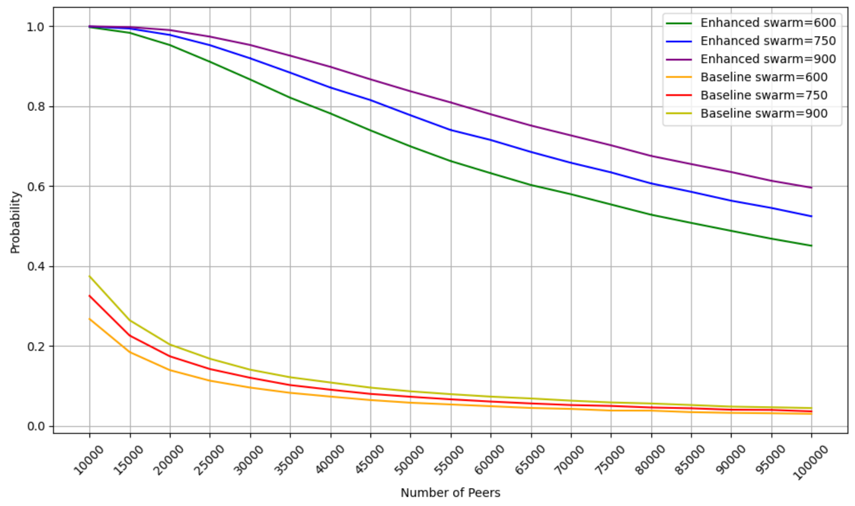

Fig. 4. The probability of finding content via Bitswap, against network size.

`know` message. To ensure statistical robustness, we repeated this process 100,000 times for each distinct scenario, varying network size, swarm size, and file popularity.

Recognizing the multifaceted nature of network dynamics, particularly in relation to file availability and peer participation, we meticulously designed our experiments to incorporate diverse levels of file popularity. Additionally, we explored the impact of swarm size, which as we mentioned before, ranges from 600 to 900 participants. We categorized swarm sizes into three distinct scenarios: a conservative scenario with $\nu_s = 600$ peers, an intermediate scenario with $\nu_s = 750$ peers, and an optimistic scenario with $\nu_s = 900$ peers. Each scenario, as mentioned before, underwent rigorous evaluation through 100,000 iterations. The results are illustrated in Figure 3 for files of different popularities.

Our analysis reveals a marked improvement in the likelihood of query responses with the introduction of the proposed solution, particularly evident in scenarios featuring files with low popularity levels. To illustrate, when examining files with a popularity rating of 1, i.e., one replica of the file in the IPFS network, and a swarm size of 600, the probability of receiving a response under the current system stands at a mere 4%. However, with the implementation of the proposed solution, this probability surges to a notable 55%. Similarly, for files boasting a popularity rating of 10 under the same swarm size conditions, the probability escalates from 32% with the existing system to an impressive 99.2% with the proposed upgrade.

It is worth noting that as file popularity increases, the gap in probability between the current and proposed solutions gradually narrows. Nevertheless, the substantial improvement afforded by the proposed version remains evident across the entire spectrum of file demand, indicating significant gains for IPFS, especially in scenarios characterized by low file traffic.

Another significant issue with plain Bitswap is scalability. In order to maintain its present level of effectiveness, the swarm's size must be proportional to the network's size. For instance, if we imagine a scenario where the network comprises 15,000 nodes and each node's swarm has 600 active connections, then if the network expands to 60,000 nodes, each node's active connections should increase to 2400 in order to sustain the same success rate.

We conducted a series of experiments to assess whether the `know` message can maintain Bitswap's efficiency, without expanding swarm size. In our experiments, we kept file popularity at an average level [5], with 5 replicas, and examined how response probabilities changed with larger network sizes. These tests were carried out for three different swarm sizes, comparing plain Bitswap against our solution, each over $100,000$ iterations. The experiments assumed that the replicas are uniformly distributed across the network. As shown in Figure 4, even when the network size reaches 65,000 nodes—roughly four times larger than our initial measurements—plain Bitswap with $\nu_s = 600$ only achieves a success rate below 10%, whereas our solution achieves a 60% success probability. In the extreme scenario where the network expands to 100,000 nodes, our approach maintains a 45% success rate, significantly outperforming the baseline's modest 3% probability, making Bitswap futureproof. The findings of these experiments are presented in Figure 4.

VII. Conclusions and Future Work

Given the importance of IPFS in the Web3 ecosystem, we contend that operational efficiency is paramount. Despite ongoing upgrades, there is room for improvement, especially in retrieval latency. In this paper, we introduced a method for enhancing the performance of Bitswap. Our goal is to ensure that queries are resolved by Bitswap itself, reducing the need to rely on the DHT. Our experiments demonstrated that our solution can significantly improve Bitswap's success rates, particularly for files with low popularity, a scenario that existing research confirms is common. We also investigated how our method would perform as the IPFS network expands. The results reveal that our solution maintains a high success rate without inflating the number of active connections, which would exhaust the node's network resources. Therefore, our solution could be beneficial in enhancing the scalability of IPFS. In the near future, we plain to conduct measurements using Testground[4], a simulation tool tailored for the IPFS network. Through this initiative, we aim to validate the efficacy of the proposed method and assess the benefits that stem from its use.

References

[1] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of IPFS: a storage layer for the decentralized web," in *ACM SIGCOMM Conference*, 2022, pp. 739–752.

[2] L. Balduf, S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Monitoring data requests in decentralized data storage systems: A case study of IPFS," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 658–668.

[3] E. Daniel and F. Tschorsch, "Privacy-enhanced content discovery for bitswap," in *2023 IFIP Networking Conference (IFIP Networking)*, 2023, pp. 1–9.

[4] C. Karapapas, G. C. Polyzos, and C. Patsakis, "What's inside a node? Malicious IPFS nodes under the magnifying glass," arXiv preprint arXiv:2306.05541, 2023.

[5] B. Confais, B. Parrein, J. Lacan, and F. Marques, "Characterization of the IPFS Public Network from DHT Requests," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems*, ser. Lecture Notes in Computer Science. Springer, 2023, vol. 14280, pp. 87–108.

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

[7] A. De la Rocha, D. Dias, and Y. Psaras, "Accelerating content routing with bitswap: A multi-path file transfer protocol in IPFS and filecoin," ProbeLab, Tech. Rep., 2021.

[8] D. Trautwein, "Nebula – A crawler for networks based on the libp2p DHT implementation," 2021. [Online]. Available: https://github.com/dennis-tra/nebula-crawler

[4]https://docs.testground.ai/master/