Accelerating Content Routing with Bitswap: A multi-path file transfer protocol in IPFS and Filecoin

Alfonso de la Rocha
Protocol Labs
alfonso@protocol.ai

David Dias
Protocol Labs
david@protocol.ai

Yiannis Psaras
Protocol Labs
yiannis@protocol.ai

***Abstract*—Bitswap is a Block Exchange protocol designed for P2P Content Addressable Networks. It leverages merkle-linked graphs in order to parallelize retrieval and verify content integrity. Bitswap is being used in the InterPlanetary File System architecture as the main content exchange protocol, as well as in the Filecoin network, as part of the block synchronisation protocol. In this work, we present Bitswap's baseline design and then apply several new extensions with the goal of improving Bitswap's efficiency, efficacy and minimizing its bandwidth fingerprint. Most importantly, our extensions result in a substantial increase to the protocol's content discovery rate. This is achieved by using the wealth of information that the protocol acquires from the content routing subsystem, to make smarter decisions on where to fetch the content from.**

***Index Terms*—P2P, Permissionless, merkle-link, IPFS, Filecoin, DHT, Kademlia, multi-path, Content Addressing**

I. Introduction

Peer-to-Peer (P2P) overlay networks have been researched extensively as an alternative content distribution model promising more efficient use of resources and faster access to content. P2P technology leverages user resources to offload servers, reducing the need for the content distributor to invest in additional server and network hardware equipment as the demand for content grows. P2P architectures are used today by P2P Content Distribution Networks (CDNs) [1], [2], hybrid P2P-centralised CDNs [3], and rapidly emerging Distributed Ledger Technology (DLT) platforms. In fact, P2P research and development has seen a renaissance as a core tenet of DLTs and blockchains with dozens of projects and initiatives pushing towards a decentralised version of the Internet and its services. There is a significant momentum already being in place by hundreds of high-profile projects with large user bases, such as: Ethereum [4], Filecoin [5], Dfinity [6], Oasis [7], among others.

Content-addressable networks have also received significant attention during the last decade, due to the promising features that they offer [8]–[12]. Location-independent content retrieval and arbitrary in-network caching can increase delivery performance significantly and reduce network resource requirements [13], [14].

What has been mostly missing is protocol designs to optimise content discovery, resolution, and delivery in permissionless networks without any central point of coordination or authority to define their topology. Both the Bittorrent and the IPFS [15] architectures use a distributed hash table (DHT) as the primary content routing mechanism. However, content routing systems often disregard a wealth of information that they acquire through their interactions: a DHT peer A that receives a request for content x from peer B and forwards it further along the DHT ring now knows that peer B caches content x. Subsequent requests received from A for x do not need to "walk" the DHT again – instead, A can redirect the request to node B directly. The utility of this information is not limited to networks using a DHT, but can apply to any content routing system where the content – rather than its original host – is explicitly identified.

In this paper, we introduce several novel extensions to Bitswap, the IPFS block exchange protocol initially introduced in [15], in order to enhance content resolution for content-addressable networks. Baseline Bitswap leverages the node's connection pool and the architecture's content resolution system to gather information and optimise content discovery. It can therefore run ahead of the main content resolution component in an attempt to discover content faster.

Bitswap operates on explicit content naming. It assumes an underlying P2P network[1], where every peer is connected to a number of other peers forming its swarm. The baseline version of the protocol makes use of a Content Identifier (CID) to request content from the peers in its own swarm.

Our novel design extensions add functionality optimising both discovery and delivery performance to the baseline Bitswap specification. These extensions are: (i) the inspection of protocol requests to perform more informed future content discoveries; (ii) the use of a TTL counter in messages to increase the range of discovery; and (iii) the use of protocol-level stream compression to make a more efficient use of bandwidth. Extensions (i) and (ii) can be applied to the baseline Bitswap protocol separately or in combination.

Additionally, we have built and make publicly available **??** our benchmarking and testbed for IPFS and Bitswap. Our testbed is built on Testground and deployed on AWS, a platform for testing and evaluation of P2P networks, which allows for extension and reproducibility.

II. Related Work

Peer-to-peer (P2P) networking has been proposed as an alternative network architecture paradigm and has been in-

[1]Bitswap can work in other environments too, but for the purposes of this study we focus on P2P Networks.

vestigated thoroughly in early 2000s [16]. The promise has always been that by dealing with data transfers in a P2P manner, both central network servers and backhaul links will be offloaded [17]. The architectural paradigm was not only considered promising for general file transfers [18], but also for CDN architectures, [3], [18], VoD platforms [19], [20], as well as real-time VoIP communication. Skype was among the first VoIP applications that built on P2P, while initial versions of Spotify was taking advantage of user-assisted storage [21].

To the best of our knowledge, not many content exchange protocols for P2P networks have been proposed in the literature. The vast majority of P2P file sharing protocols are usually focused on *content discovery*. One of the most widespread solutions for piece exchange is Bittorrent's *"tit-for-tat" algorithm* [22]. Generally speaking, Bittorrent peers leverage one of the available content routing systems in the network (mainly the DHT or a centralised tracker) to find seeders storing the pieces of the content. A peer is allowed to request pieces from as many seeders as it wants. According to "tit-for-tat", the amount of data that a peer can download in parallel is proportional to the amount of upload bandwidth it is contributing to the network. Consequently, the piece selection strategy drastically affects file sharing performance [23], [24]. Proposals to improve file-sharing in P2P networks have mainly focused around the use of network coding [25] and rateless coding [26] to leverage multiple path streams and make all content pieces equally valuable.

Performance optimisation of P2P systems has mainly been attempted through improving the performance of the content routing system itself. Distributed Hash Tables [27], pubsub protocols [28] and tracker operation improvements have been the main focus points for performance optimisation [29]. Alternative network setups, such as collaborative downloads to accelerate downloads [30], or application-oriented content discovery based on user interests and social-networking links have also been investigated.

We argue that content exchange itself *at the protocol layer* has not been looked at to the extent it deserves. We have not seen protocols that leverage the knowledge of content routing systems as well as the network and the peer-connectivity setup to accelerate content discovery and content fetching. Content addressing proves to be an invaluable tool to facilitate this process [8], [11], [31].

Bitswap's design targets explicitly content fetching and discovery optimisation at the protocol layer, taking advantage of the peer's and the network's previous activity, as well as knowledge from the underlying content routing subsystem.

Bitswap is already being used as a content exchange protocol in the InterPlanetary File System (IPFS) [15], being responsible for the efficient and timely transfer of hundreds of GBs of traffic per day.

Bitswap is also used in the Filecoin network [5] as part of the block exchange protocol stack. Filecoin is the first of its kind blockchain-based storage and delivery network with more than 1EiB of storage capacity pledged by 800 storage miners and a total network size of 10,000 nodes. Bitswap is used as part of the chain synchronisation module of the blockchain, that is, it is responsible for finding and fetching blocks between miners that have fell out of sync with the latest blocks produced [32]. Again, Bitswap comes in as an extension to the main message propagation protocol, GossipSub [33], to utilise the information it has gathered and discover content in the vicinity of Filecoin's storage miners.

III. The Bitswap Protocol

Bitswap is a message-oriented exchange protocol used to request and send content blocks between peers in a P2P content-addressable network. Bitswap's key value proposition is the ability to traverse a hash-linked graph that represents a file structure, fetching the multiple parts of the graph from different peers, optimizing for throughput and bandwidth usage. In this section we present a detailed description of its operation and its integral modules.

A. Content Objects in Bitswap

Blocks are the minimum content unit exchanged in the Bitswap protocol. When a peer wants to provide a file to the network, it chunks the file into several blocks of 256KiB (default) and up to 1MiB. Each of these blocks has its own unique identifier, the CID (*Content IDentifier* [34]). The CID of a block is generated from the hash digest of the content included in the block. Any change to the content itself results in a totally different hash – and therefore, a different identifier – making CIDs immutable, permanent, and a very convenient way to uniquely identify blocks of content in the network.

Blocks relate to each other through merkle-links. Blocks belonging to the same file are linked to each other following a Merkle DAG structure. Merkle DAGs are similar to Merkle trees, but there are no balance requirements, and every node can carry a payload (instead of just leaves as in Merkle trees). In DAGs, several branches can re-converge, so nodes of the structure can have several parents. Merkle DAGs are self-verified structures. The CID of a node is univocally linked to the contents of its payload and those of all its descendants. Thus two nodes with the same CID represent exactly the same DAG, and consequently the exact same file. A set blocks structured in a Merkle DAG can represent any kind of content, from a file, to a directory, to a complete filesystem. Content within Bitswap is requested through the root CID of the Merkle DAG representing the content.

B. Protocol Architecture

The Bitswap protocol exposes a simple interface to get content and blocks from the network. Its architecture comprises the following subsystems (Figure 1):

(i) the *connection manager* is responsible for the management of network resources within the protocol. It tracks the peers it is connected to, and is responsible for the exchange of messages with other peers;

(ii) the *session manager* governs the lifecycle of Bitswap sessions. Sessions are independent flows of operation tracking the different requests for content that a peer initiates. Sessions

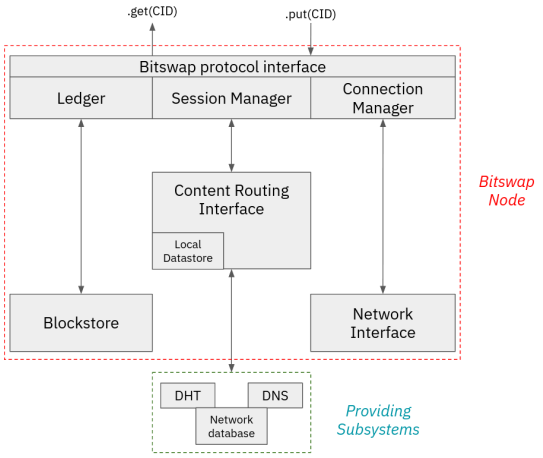

Figure 1. Bitswap architecture

orchestrate the flow of messages with other peers for the discovery and retrieval of content, with each session being dedicated to one file and/or directory. A session can leverage different content routing subsystems to find candidate peers storing the content to whom requests can be sent. Some examples of content routing subsystems that can be leveraged by Bitswap to discover content are a DHT to lookup content providers in a P2P network, DNS to find the location of a resource, databases with information about content (e.g. Trackers), or the node's local datastore;

(iii) finally, *the ledger* tracks the status of all the requests for content that other peers have exchanged. Upon a request from another peer, it checks the local blockstore to determine if it can fulfill the request.

C. Bitswap messages

Bitswap incorporates three types of requests (`WANT-HAVE`, `WANT-BLOCK` and `CANCEL`), plus three types of responses (`HAVE`, `BLOCK`, `IDONT-HAVE`). A list of CIDs with the items being requested by a node is referred to as the *wantlist*.

- Request types:
  - `WANT-HAVE`: Request sent by a peer including a *wantlist* containing the list of items that it wants to retrieve. This request prompts other peers storing any of the blocks for the CIDs in the *wantlist* to reply with a `HAVE` response; those not storing the block reply with an `DONT-HAVE` message.
  - `WANT-BLOCK`: Request used to ask for a block listed in the attached *wantlist*. If the receiving node has the block it transfers it using a `BLOCK` response, instead of a `HAVE` message. Respondents not holding the block answer with an `DONT-HAVE` response. The difference between `WANT-HAVE` and `WANT-BLOCK` is that with the latter, a peer requests transfer of content, saving one RTT compared to the `WANT-HAVE` case. The drawback is if `WANT-BLOCK` is sent to multiple peers, it can result in duplicate blocks being sent back, and therefore higher overhead.
  - `CANCEL`: Request to cancel a previous request.
- Response types:
  - `HAVE`: Response to an `WANT-HAVE` message stating that a set of CIDs included in a *wantlist* are locally stored.
  - `IDONT-HAVE`: Response to an `WANT-HAVE` message sent to notify that the set of CIDs included in a *wantlist* are not stored locally by the peer.
  - `BLOCK`: Response including the requested block (*i.e.,* content) for a CID.

Bitswap peers can include both a `WANT-HAVE` *wantlist* and a `WANT-BLOCK` *wantlist* for different CIDs in the same Bitswap message envelope. This action requests "knowledge of possession" for some CIDs and the transfer of blocks for others. Analogously, a Bitswap message response can include `HAVE`, `DONT-HAVE`, and `BLOCK` responses for different items within a peer's *wantlist* in the same envelope.

D. Detailed operation

1) Initiate the request session: When a user wants to retrieve content from the network using IPFS, it triggers the creation of a *Bitswap session* for the root CID of the content. Sessions are started by the broadcast of a `WANT-HAVE` *wantlist* through the peer's connection manager to all of the node's connected peers. The *wantlist* of this `WANT-HAVE` only includes the root CID of the content being requested. Until the first block is retrieved, Bitswap has no knowledge of the rest of the blocks in the DAG structure of the file.

Peers receiving this `WANT-HAVE` message add the received *wantlist* to their ledger. The ledger checks if the peer has the requested block stored locally and responds with a `HAVE` message if the block is found. Every node responding with a `HAVE` is added as part of the session on the client side. Subsequent requests are only forwarded to nodes belonging to the session, as they are the most likely to have the rest of the blocks for the requested content. The moment the requesting client receives one of these `HAVE` responses, it responds with an `WANT-BLOCK` request to that peer to retrieve the block for the root CID (Figure 2).

2) Traversing the DAG: The root CID block points to the set of CIDs in the next level of the DAG. With this information, the client's session constructs a new `WANT-HAVE` request including in the *wantlist* these second-level CIDs. This time, to save network resources, the `WANT-HAVE` request is only forwarded to the nodes included in the session. According to the `HAVE` / `DONT-HAVE` message pattern received from the session peers for the CIDs included in the requested *wantlist*, the session tailors the `WANT-BLOCK` requests to trigger the transfer of blocks.

The retrieval of blocks through `WANT-HAVE` requests, and transfer of blocks using `WANT-BLOCK` is repeated iteratively until the full Merkle DAG for the file is traversed and all blocks are retrieved.

We discuss in the next subsection how `WANT-BLOCK` messages are sent to all peers that have responded with

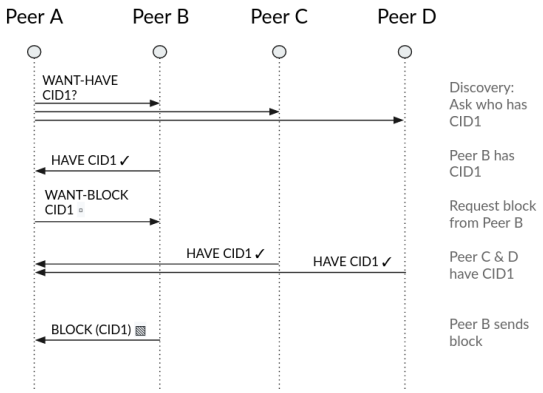

Figure 2. Sessions message flow

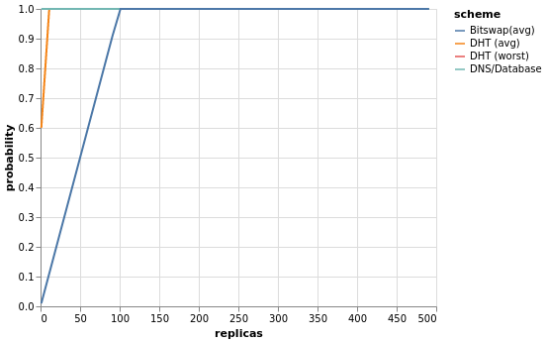

Figure 3. Probability Comparison ($n = 100,000; l = 1000$)

WANT−HAVE messages in order to minimise reception of duplicate blocks. In all cases, once the BLOCK with requested CID is received, a CANCEL message is sent to any peer to whom we previously sent a WANT−HAVE for that CID.

3) Peer selection: Bitswap sessions track the number of BLOCKs a peer has successfully sent throughout the message exchange. This information is used to choose the peer to whom to send the WANT−BLOCK. This selection is performed probabilistically: a peer is chosen stochastically with a probability proportional to the number of blocks successfully sent by the peer in previous exchanges. To illustrate how this works, consider a session with two peers, A and B, that have responded with a HAVE for a requested block. A has successfully sent $2b$ blocks in previous interactions, while B sent b blocks. To select the peer to whom to send the WANT−BLOCK, the session randomly chooses between A and B with probabilities $2/3$ and $1/3$, respectively.

Sessions keep block counters for peers, so if a peer is active in more than one sessions, the probability of it being selected to receive a WANT−BLOCK is different in each session. If the selected peer does not store the block and answers with a DONT−HAVE to the WANT−BLOCK, a new peer from the session is chosen and a new WANT−BLOCK is sent.

Additionally, when a session is already populated with HAVEs, peers may send an optimistic WANT−BLOCK to a random peer in the session to reduce the time required to fetch a block by one Round Trip Time (RTT).

4) Triggering lookup operations in the content routing subsystem: Peers are pruned from sessions when they respond with an DONT−HAVE message to several subsequent requests[2]. This signals that a peer does not have any more blocks from the content object being requested.

Finally, it may be the case that none of the connected peers of a node store the block it is looking for. Bitswap is designed to leverage content routing subsystems for the discovery of peers that have the requested content. When utilising such subsystems, Bitswap populates the corresponding sessions with this information, in order to optimise future requests. These content routing subsystems can be a DHT [35] that returns a set of content providers, a DNS [36] service that gives the session a list of potential servers storing the resource, or networked key-value stores [37] with information about where the content resides in the network.

Additionally, sessions periodically[3] broadcast WANT−HAVEs for a single random block from the *wantlist* using the same operations explained in subsection III-D1 in order to populate the session with new peers. The node may have established new connections, and connected peers may have requested new content, so sessions are periodically repopulated through broadcasts to incorporate this new knowledge.

IV. Content Routing Analysis

In this section we present a comparison between two approaches to content routing using Bitswap: a centralized solution (DNS), and a decentralized solution (Kademlia DHT). The purpose of this analysis is to determine the trade-off between speed to find the first block and retrieval costs and identify scenarios where certain content routing solutions fail.

A. Probability of finding blocks

We consider that the popularity of a block stored in the network is determined by the number of replicas stored in different peers of the network. This number indirectly reflects the number of times the item has been requested (and subsequently stored) in the network. Let's also assume that these replicas are uniformly distributed throughout the network.

Under this scenario, and without any aid from external routing systems, the average probability of Bitswap finding a block in the network depends on the number of the peer's connections (l), the number of peers in the network (n), the overlap in the peers' connections (α) and the popularity of the block (r), according to Eq. 1.

$$rob_{Bitswap} = \frac{r * l * (1 - \alpha)}{n} \quad (1)$$

[2]Set to 16 messages by default. This parameter is configurable to adjust the pruning speed of sessions.

[3]Set to every 60 seconds in the current implementation.

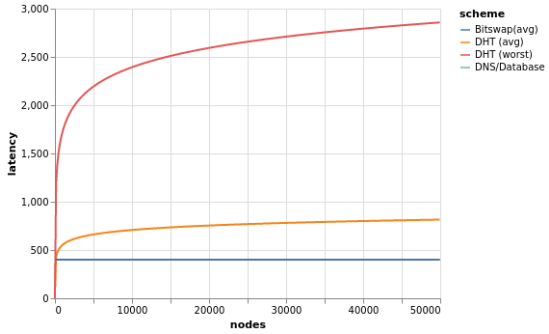

Figure 4. Latency Comparison ($r = 5; l = 1000; k = 20$)

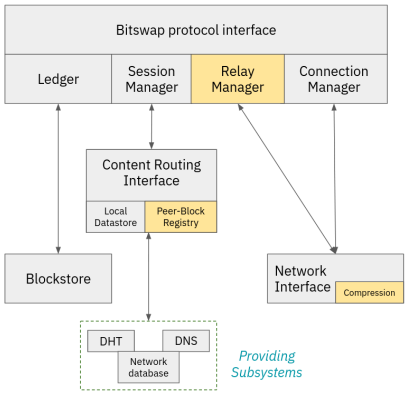

Figure 5. Bitswap architecture with improvements

The probability of finding blocks for the DHT and the DNS can be considered to be 1. Lookup operations in both the DHT and DNS will locate the content, provided that all the nodes in the network are reachable. However, the time to perform the lookup operation is different in each case.

B. Block retrieval

Bitswap is able to pull blocks if directly connected to another peer storing the content. If this is the case, Bitswap is able to discover and retrieve a block in at most 2 $RTTs$ (one RTT for discovery, and another RTT for the content exchange). Note that 2 $RTTs$ is the minimum when using DNS, where a peer does one RTT to find the node that stores the block and an additional RTT to pull the content from the node storing it.

In contrast, the time required to discover a block with the Kademlia DHT depends on (i) the size of the network in terms of number of nodes (N), (ii) the popularity of the content (R) and (iii) the size of the k-buckets in the peers' DHT routing tables (K). A lookup in the network is $O(log\ n)$ in the worst case, and $O(log_k\ n)$ when k-buckets are full (refer to Sec. 3 of [35]). Assuming that all nodes in the network are reachable, the time to fetch content over a DHT is:

$$Time\ fetch_{DHT} = (1 + log_K\ N/R) * RTT \qquad (2)$$

In the worst-case scenario, the above equation becomes:

$$Time\ fetch_{DHT} = (1 + log_2\ N/R) * RTT \qquad (3)$$

Therefore, despite the fact that the probability of finding a block using a DHT system is equal to 1, the time required to discover the content is high even for popular content, and increases as the size of the network grows. More importantly, the time to fetch content using a DHT grows with the number of blocks of the content item requested. Larger items inevitably require more DHT "walks", increasing the total time to fetch. Note that this is independent of whether a single node stores all blocks of an item, as every block is independently addressed.

Centralised approaches like the DNS system in our evaluation above offer both a high probability of content discovery and fast content resolution because the entire lookup depends on contact with a single entity. The decentralized DHT also offers a high probability of discovery, but its lookup and fetch operations are considerably slower. Bitswap is a decentralized enhancement to the underlying content routing system that speeds up content resolution and makes any content routing system comparable to DNS's speed.

V. Improvements to the Bitswap Protocol

In this section, we present our testbed setup and start with the evaluation of the baseline version of Bitswap. We compare the performance of a DHT content routing system with and without Bitswap and demonstrate Bitswap's performance improvements (Section V-B). We then proceed to apply further improvements to the baseline Bitswap protocol (summarised in Fig. 5) evaluate each corresponding performance boost (Sections V-C. V-D, V-E, V-F).

A. Bitswap Benchmarking & Testing Setup

We built a testbed for the Bitswap evaluation using Testground [38], an open-source platform for testing, benchmarking and emulating distributed and peer-to-peer systems at scale. Testground's basic unit of execution, the "test plan", wraps the protocol or subsystem to be tested, and specifies the desired behavior of nodes in an experiment. The test plan allows the creation of tailored test runs by composing scenarios declaratively. A configuration file is used to specify experimental parameters such as the network topology, number of nodes, execution runtime, network traffic shaping, and version of the protocol, as well as any additional configuration parameters required by the test plan. Testground automatically collects real-time metrics specified in the test plan during the experiment's execution.

All the test plans designed for this evaluation were run using Testground's Docker runner inside an AWS *t2.2xlarge* with 8vCPUs and 32GiB of RAM. Testground's Docker runner runs every instance (*i.e.,* node) of the experiment in a different Docker container without any resource limits. Nodes in our experiments are instances of *go-ipfs* [39] running the Go implementation of Bitswap and IPFS's Go implementation of the Kademlia DHT as the content routing subsystems. We have spun up 30 nodes in total for our experiments, unless otherwise stated. Our nodes are connected in different, scenario-specific

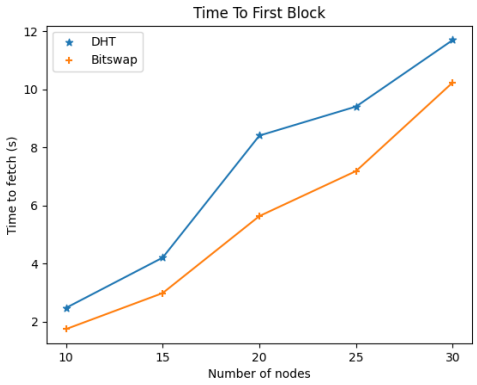

Figure 6. Time to first block of Bitswap compared to Kademlia DHT (BW=100Mbps; latency=100ms)

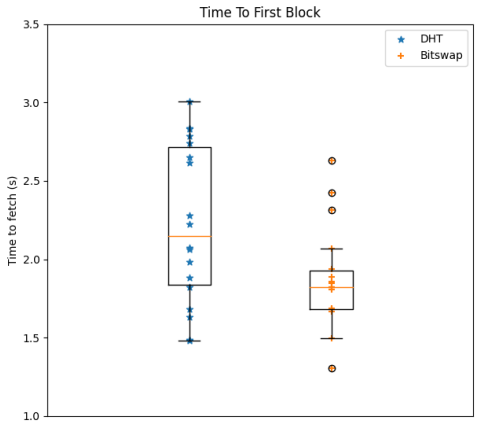

Figure 7. Time to first block of Bitswap compared to Kademlia DHT in waves experiment (BW=100Mbps; latency=100ms)

mesh topologies (discussed in each subsection) with bidirectional links of 100 Mbps and 100 ms delay.

B. Bitswap baseline performance

To evaluate the performance improvement that Bitswap brings as a protocol extension to content routing subsystems, we ran an experiment where $N-1$ leecher nodes try to fetch a block from a single seeder storing that block in a network of N nodes in total. Nodes are arranged in a fully-connected mesh topology. We ran an experiment comparing content retrieval using the Kademlia DHT [35] as a content routing system vs content retrieval using Bitswap to search for content opportunistically. The results in Figure 6 show that when leechers use Bitswap, they are able to retrieve a single block 30% faster on average than leechers using the DHT, independently of the number of nodes in the network. The fact that we are using a mesh topology for the experiment means that leechers are directly connected to the seeder storing the content. While Bitswap leechers leverage the fact that they are already connected to the seeder to find the content and retrieve it, leechers using the DHT have to perform a full lookup to discover providers.

Furthermore, Bitswap's time-to-first-byte (TTFB) is not affected by the number of nodes storing the block in the network. As long as the peer storing the block is within Bitswap's reach, the time to retrieve it is deterministic and determined by the network's RTT and network conditions. For the DHT, on the other hand, the lookup operation time decreases with the number of nodes storing the block, confirming the assumptions presented in IV. We repeated the above block retrieval experiment in a network of $N = 20$ nodes, but now instead of leechers requesting the block all at the same time, they request it one node at a time. The results in Fig. 7 show how the dispersion of the TTFB is higher for the DHT compared to Bitswap, with the difference reaching up to more than 30% in most cases. Nodes in the last waves using the DHT are able to retrieve the content with a comparable performance to Bitswap because the DHT lookup runs into a node storing the content early in the process. Even in this case, however, Bitswap retrieves the block faster than baseline DHT (*i.e.,* without Bitswap's support). While the DHT needs to contact a node to perform the lookup (even if it finds the seeder in the first step) before requesting the retrieval of the block, Bitswap is able to poll its connected peers and immediately request transmission of the block.

C. `WANT` message inspection

Bitswap's baseline design does not use information about the operation of the protocol, or previous events in the network, to direct subsequent lookups. An example of useful information being discarded by Bitswap nodes in its vanilla implementation is the knowledge about what blocks are being requested by its directly connected peers. This information can be extremely useful for later requests.

In this first improvement of the protocol, we add the ability for the connection manager to inspect `WANT` messages being received from other peers. All this information is stored in a local data structure exposed through the Content Routing Subsystem interface called the *"peer-block registry"*.

The peer-block registry maps each CID seen by a node to the peers that have recently requested this particular CID.

This information is then used by Bitswap sessions to direct their search for content. Whenever a peer wants a CID, it checks the peer-block registry for that CID to see if it is populated with peers that already requested that content recently. The hypothesis being that if someone requested it, it will likely have found it and retrieved it meanwhile.

This is in contrast to the protocol's baseline operation of broadcasting `WANT-HAVE` messages to all its connected peers. Instead, it only sends an `WANT-BLOCK` to the n_{pb} peers in the peer-block registry that have seen that CID most recently[4]. If the contacted peers have the content, they immediately respond

[4] $n_{pb} = 3$ in the default implementation, but this number can be configured to set the protocol's aggressiveness. For instance, to minimise the number of duplicate blocks in the network we can set $n_{pb} = 1$

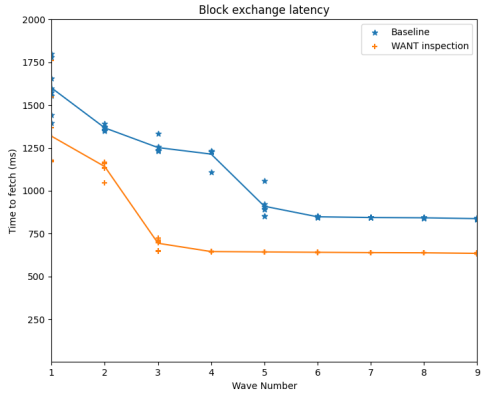

Figure 8. Time to fetch block in waves experiment of Bitswap with and without WANT inspection (BW=100Mbps; latency=100ms)

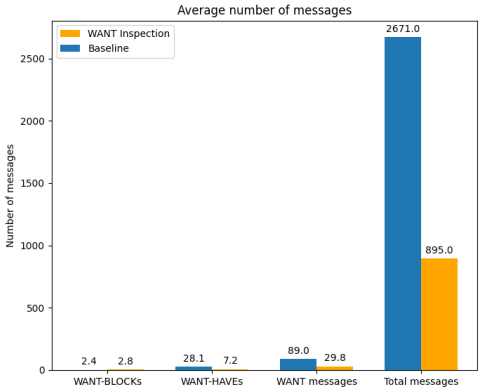

Figure 9. Number of messages in waves experiment of Bitswap with and without WANT inspection (BW=100Mbps; latency=100ms)

to the WANT−BLOCK with the corresponding block, while if this is not the case, they answer with an IDONT−HAVE message. In the latter case, the peer performs broadcasting of WANT−HAVEs to all its connected peers.

Adding WANT inspection and the peer-block registry to Bitswap nodes (Sec. V-C) reduces the time required to discover and transfer popular content by one RTT. To validate this result, we run an experiment where 30 different leechers try to retrieve a block from a single seeder in the network. We compare baseline Bitswap with Bitswap with "WANT inspection" enabled. Leechers request the content in waves to emulate the retrieval of increasingly popular content.

As shown in Fig. 8, for the baseline implementation of Bitswap, the first wave is the slowest because only the seeder has the content. For subsequent waves, multiple nodes in addition to the original seeder already have the content, so when leechers broadcast their WANT−HAVEs they have a higher probability of hitting a node with the requested content. Despite hitting a node with the content, the minimum number of RTTs required by the vanilla implementation of Bitswap to get the content is two: one for the WANT−HAVE broadcast, and another one to explicitly request the content with an WANT−BLOCK.

If peers are lucky, they will hit the content in a single WANT−BLOCK and receive the block in that same interaction, reducing the time to fetch the content to a single RTT. Fig. 8 shows the result of the experiment using *100ms* latency links between nodes. The improvement brings a reduction of at least *200ms*, *i.e.,* 1RTT, in the time to fetch blocks, which translates to a 30% improvement from the baseline implementation.

WANT message inspection brings one more performance improvement for popular content: the number of control messages exchanged by Bitswap nodes is significantly reduced. Keeping a list of peers that have recently requested a specific CID in the peer-block registry allows for transmission of optimistic WANT−BLOCK messages (*i.e.,* transmission to targeted peers only), eliminating the need to broadcast WANT−HAVEs to all our connected peers. The average number of WANT messages exchanged is thus reduced by 33%, while the number of WANT−HAVEs is reduced by 75% as shown in Fig. 9.

D. TTL field in Bitswap messages

When Bitswap's attempt to find content quickly from its directly connected peers fails, it resorts to the underlying content routing subsystem to find a provider for the content. By adding a *Time to Live* (TTL) to Bitswap messages, nodes are able to forward their requests for blocks to nodes TTL+1 hops away, extending their range of discovery and minimizing their dependence on the content routing system lookup operations.

With this improvement we add a new module in the Bitswap protocol interface, the *relay manager*. All the logic behind the request and discovery of content on behalf of other peers is managed through the *relay manager*. When a node receives a WANT message with a TTL greater than zero, it reduces the TTL of the request by one and forwards the request to d (degree) of its connected nodes that have not yet received this request. The *relay manager* tracks all the WANT messages and CIDs being requested on behalf of other nodes. When a node receives a block from a request belonging to some other peer, the *relay manager* forwards the block to the source, following the same path as the original request (symmetric routing).

The degree d of the *relay manager* is used to limit the outreach of requests and avoid flooding the network. Thus, the performance of the protocol can be conveniently adapted through the configuration of the TTL and the degree of the *relay manager*. Larger TTLs lead to more extended ranges for content discovery at the expense of a higher messaging overhead. The amount of overhead can be controlled through the degree d, which can vary depending on the depth of the TTL-formed tree, *e.g.,* layer 1 peers can set $d = 100\%$ of their connection pool, layer 2 peers can set it to $d = 50\%$ and so on. This assumes a universal setting for d, which is normal in protocol design.

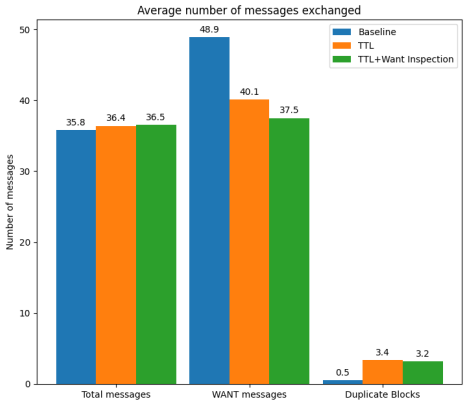

Figure 10. Number of messages exchanged latency of Bitswap + DHT compared to Bitswap with TTL and Bitswap with TTL and `WANT` Inspection (BW=100Mbps; latency=100ms)

This improvement gives Bitswap additional discovery capabilities which, in turn, minimises dependence on the underlying, typically slower, content routing system.

We set up an experiment where 15 different leecher nodes request blocks from five different seeder nodes in the network. Leecher and seeder nodes are not allowed to be connected directly, and they can only communicate through five passive nodes that neither provide nor request content from the network but run the Bitswap protocol normally. For the experiments we used $TTL = 1$, and degree $d = 10$.

As a baseline, we use vanilla Bitswap with a Kademlia DHT as the complementary content routing subsystem. We compare its performance against standalone Bitswap nodes (*i.e.,* not connected to any content routing subsystem) using TTL in messages and with the *relay manager* enabled.

Our experiments show that this improvement enables Bitswap nodes to reduce the time to fetch content not stored by directly connected peers by 33% compared to vanilla Bitswap.

By enabling the relay manager and adding the TTL field to Bitswap messages, nodes are allowed to broadcast `WANT` messages to nodes TTL+1 hops away, discovering seeders storing the block faster than the iterative request-response process of the DHT.

The use of TTL in Bitswap messages results in additional overhead in terms of the average number of messages exchanged by peers compared to Bitswap's vanilla implementation of only 1.6% when TTL=1 and $d = 10$ at the cost of a 5-fold increase in the number of duplicate blocks exchanged in the network. The range of Bitswap requests is amplified by delegating the discovery of content to other nodes, but the requester does not have a direct channel to notify the peers seeking blocks on its behalf that it has received the requested blocks. Peers that are not able to see the requester's `CANCEL` message continue their search, forwarding additional duplicate blocks and increasing the network's bandwidth requirements and the peers' load.

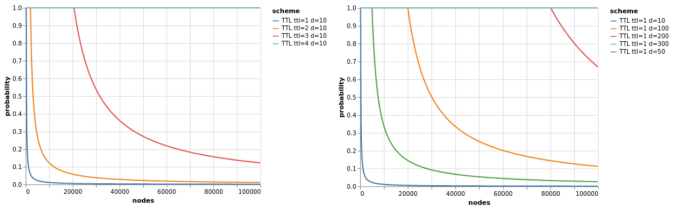

Figure 11. Probability of finding one replica of content in a network for different values of TTL (left) and d (right)

In order to evaluate the probability of content discovery using Bitswap with TTL≥1 in large networks, we simulated networks of up to 100,000 nodes for different configurations of d and TTL. The results are presented in Fig. 11. As expected, the higher the degree and the TTL, the higher the probability of discovery. Hitting the right balance between TTL, d, and network overhead depends on the network size and the connectivity of nodes. Ultimately, this is an application-level decision and can vary depending on the application's requirements.

E. `WANT` inspection and TTL

Combining the previous two improvements results in significant performance gains. The fact that nodes can relay `WANT` messages from other peers increases the number of requests exchanged by a node. On the other hand, by inspecting `WANT` messages Bitswap nodes can populate their peer-block registry (see Section V-C) with valuable information.

In addition to direct sessions that can take advantage of this extra information, as discussed earlier, this information can also be utilised by the *relay manager* to target peers with a higher probability of storing the requested content, instead of selecting peers randomly. Thus, the protocol intelligently selects to whom the TTL'ed `WANT` requests are forwarded. When a node receives a `WANT` message with a TTL larger than zero, instead of forwarding the `WANT` message with TTL-1 to a random subset of d of its connected peers, it first chooses n_{pb} candidates from the peer-block registry that have recently seen the CID (if any) and selects the remaining $d - n_{pb}$ peers for the forwarding session randomly.

This increases the probability of intermediate nodes finding the content, enabling faster discovery of blocks. Additionally, the fact that nodes may be looking for blocks on behalf of other peers makes the use of the information in the peer-block registry more powerful than in baseline Bitswap.

For the evaluation of this improvement we repeat the TTL improvement experiment in Section V-D, but in this case enabling the `WANT` inspection and TTL improvements in Bitswap nodes. As shown in Fig. 12, combining both improvements decreases the time-to-fetch by 12%, compared to plain TTL improvement and approximately 70% compared to baseline. Nodes are now able to inspect forwarded `WANT` messages from nodes they are not directly connected to, expanding the range of recently requested content that they can view and as a result,

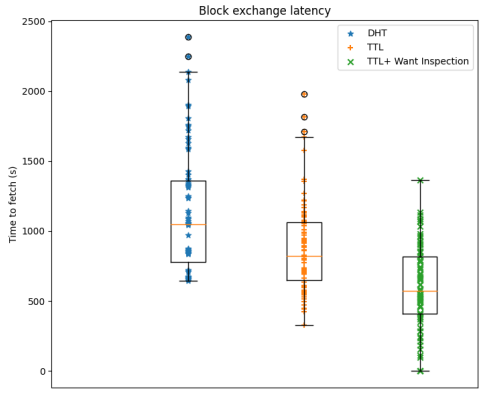

Figure 12. Block exchange latency of Bitswap + DHT compared to Bitswap with TTL and Bitswap with TTL *and* `WANT` Inspection (BW=100Mbps; latency=100ms)

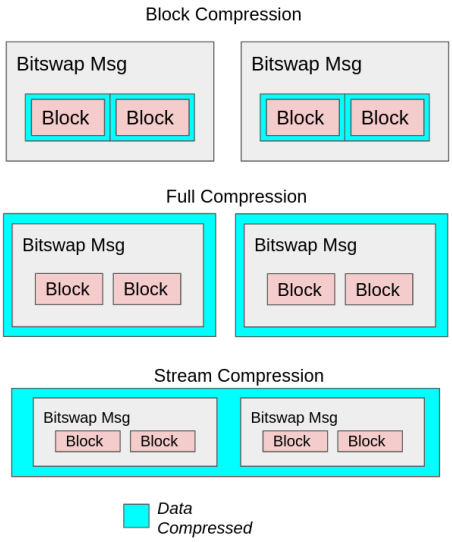

Figure 13. Bitswap compression strategies

the probability of fetching content without having to fall back to the content routing system.

Furthermore, the inclusion of the peer-block `WANT-BLOCK` round in the sessions' discovery phase before broadcasting to every connected peer, and the use of the peer-block registry in the relay session reduce the number of previously seen duplicate blocks. The fact that nodes have a way to request the content from peers with a larger probability of storing it reduces the number of messages required in both direct sessions and relay sessions as shown in figure 10.

F. Stream Compression

Bitswap's fast block retrieval capabilities position it as a highly useful block exchange companion and extension to existing content routing systems. In order for Bitswap to make a more efficient use of bandwidth, and speed-up the transmission of blocks, we introduced a compression layer in Bitswap's network interface.

We introduced three different compression strategies in Bitswap:

- *Block compression*: In this compression approach, we compressed blocks before including them in a message and transmitting them through the link.
- *Full message compression*: Instead of only compressing blocks, we compressed the entire message envelope before sending it.
- *Stream compression*: This method uses compression at a stream level, using a stream wrapper to compress every byte that enters the stream writer of a node at the transport level.

Our experiments with these strategies demonstrated that the rate of compression achieved using the block compression strategy is significantly lower than the rate achieved using the full message strategy or the stream compression. This is particularly the case if we exchange random files, i.e. those with little redundancy. These observations aligned with our expectations: compression results were largely determined by the number of redundant elements that could be exploited by the compressor.

The computational overhead and the file-sharing latency using block and full message compression is significantly higher than when using stream compression. This was another expected outcome: in stream compression, the node can directly output the data to the transport layer through the network interface; in the block and full message compression strategies, the node has to perform the appropriate compression before it can send the first byte to the transport layer.

Applying compression at the stream level can lead to significant performance improvements and a more efficient use of bandwidth compared to other compression strategies. In some cases – and depending on factors such as the compression algorithm and the file format – the bandwidth savings can reach up to 50%.

To evaluate the improvement of adding a compression layer to Bitswap, we designed an experiment where two connected peers, one client and one provider, exchanged datasets of sizes up to 30GB from the "awesome IPFS" dataset collection [40].

We compared the bandwidth use in the exchange of these datasets with and without compression. The use of the protocol-level compression strategy achieved up to a 75% decrease in bandwidth for large datasets, showing bandwidth savings of at least 12% for datasets of every size.

VI. Future Work

We designed and implemented a series of Bitswap improvements following a modular architecture to encourage future experimentation and extensions. In this section, we propose further enhancements.

1) Support for complex queries through wantlist manifests: Transforming `WANT` requests into manifests. Session manifests can include structured information about content query

selectors, or additional protocol configurations to fine-tune the retrieval of blocks according to the application's needs. Thus, instead of sending plain `WANT` *wantlists*, sessions can perform more complex requests asking for a subset of blocks from specific content, alternative coding schemes in blocks, multiple non-overlapping streams of transmission, request quality of service requirements, or add any other custom protocol extension.

2) Adapt block size to data access pattern: By default, blocks have the same size. By adjusting the size of the blocks according to the content structure and size, we optimize discovery and transfer for different access patterns (e.g. streaming video vs. fetching a webpage).

3) Connection Manager: Adding intelligence to the Connection Manager module so that instead of randomly connecting to other peers, or establishing a connection as a result of a previous interaction with a peer, the Connection Manager intelligently chooses the best connections to make. In this way, it can intelligently connect to peers according to a score, which is determined depending on the peers latency, its observed available bandwidth, or any other heuristic that could make it a good candidate to speed-up content retrieval.

4) Introduce novel network coding techniques: By applying network coding and generating new blocks that are linear combinations of the original blocks for a specific piece of content, we make all of the blocks of an object equally valuable. This reduces the number of duplicate blocks, minimizes the impact of "rare blocks", and enhances the use of multiple streams of transmission for the same content. This can potentially reduce the bandwidth requirements of the protocol.

5) Network Interface: Using schemes to make more efficient use of bandwidth, such as the use of different compression strategies and algorithms beyond those we have experimented with already.

6) Peer-Block Registry: Using the characteristic time of the cache for nodes storing some content, as well as other heuristics to predict when a block may not be available in a peer's cache, we can optimise further the selection of peers to send `WANT-BLOCK` messages to. Furthermore, dynamic configuration of n_{pb}, according to a number of parameters, such as the state of the peer-block registry and the number of active connections, enables fine-tuning of the number of `WANT-BLOCK`s sent in the discovery phase and more efficient use of bandwidth, as a result.

7) Relay Manager: In the asymmetric routing approach, WANT messages would include a "source" field along with the TTL field. Peers would include their ID as source in the WANT messages so that when the requested block for a relay session is found, instead of being forwarded following the same path followed by the WANT message, it can be directly sent to the requester without traversing intermediate nodes. This extension to the protocol would reduce the bandwidth use compared to the current implementation, at the cost of requiring the establishment of a connection between the peer storing the block and the requester.

VII. Conclusions and Final remarks

In this work we presented the operation of Bitswap, the content exchange protocol being used in the InterPlanetary File System (IPFS) and as part of the block exchange protocol stack of the Filecoin blockchain. We show how an exchange protocol such as Bitswap can provide significant speed-ups and benefits compared to traditional content exchange protocols in P2P networks. The modular design of Bitswap makes it convenient to introduce performance-enhancing improvements leveraging the information gathered through the regular execution of the protocol and its interaction with other external subsystems.

We evaluated extensively each contribution in an AWS-based testbed environment and make our test environment available for others to repeat the research and build upon it [41]. All the work performed so far is now being incorporated into the IPFS protocol by the IPFS team.

We invite the research community to consider additional ways of exploiting Bitswap's rich information store, and to reuse or extend the existing testing harness.

References

[1] peer5: Reliable, scalable ecdn based on webrtc, https://www.peer5.com.
[2] strivecast p2p cdn, https://strivecast.com.
[3] Mingchen Zhao and et al. Peer-assisted content distribution in akamai netsession. In *ACM IMC'13*, pages 31–42, 2013.
[4] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.
[5] J Benet and N Greco. Filecoin: A decentralized storage network. *Protoc. Labs*, pages 1–36, 2018.
[6] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.
[7] Oasis Protocol Project. The oasis blockchain platform. 2020.
[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *ACM SIGCOMM'01*, 2001.
[9] Amos Fiat and Jared Saia. Censorship resistant peer-to-peer content addressable networks. In *SODA'02*, page 94–103, USA, 2002. Society for Industrial and Applied Mathematics.
[10] D.R. Cheriton and M. Gritter. Triad: A new next-generation internet architecture, 2000.
[11] Christian Dannewitz and et al. Network of information (netinf): An information-centric networking architecture. *Computer Communications*, 36(7):721 – 735, 2013.
[12] Van Jacobson and et al. Networking named content. In *ACM CoNEXT '09*, page 1–12, New York, NY, USA, 2009.
[13] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, and M. Varvello. From content delivery today to information centric networking. *Computer Networks*, 57(16):3116 – 3127, 2013.
[14] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. ICDN: An NDN-Based CDN. In *ACM ICN'20*, page 99–105, 2020.
[15] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
[16] Andrea Passarella. A survey on content-centric technologies for the current internet: Cdn and p2p solutions. *Computer Communications*, 35(1):1 – 32, 2012.
[17] Amit Mondal, Ionut Trestian, Zhen Qin, and Aleksandar Kuzmanovic. P2p as a cdn: A new service model for file sharing. *Computer Networks*, 56(14):3233 – 3246, 2012.
[18] Yaoqi Jia, Tarik Moataz, Shruti Tople, and Prateek Saxena. Oblivp2p: An oblivious peer-to-peer content sharing system. In *USENIX Security 16*, pages 945–962, Austin, TX, August 2016. USENIX Association.
[19] Yan Huang and et al. Challenges, Design and Analysis of a Large-Scale P2p-Vod System. In *ACM SIGCOMM '08*, page 375–388, 2008.
[20] Bin Fan, David G. Andersen, Michael Kaminsky, and Konstantina Papagiannaki. Balancing throughput, robustness, and in-order delivery in p2p vod. In *ACM Co-NEXT '10*, 2010.

[21] G. Kreitz and F. Niemela. Spotify – large scale, low latency, p2p music-on-demand streaming. In *IEEE P2P 2010*, pages 1–10, 2010.

[22] Bram Cohen. The bittorrent protocol specification, 2008.

[23] Jiaqing Luo, Bin Xiao, Kai Bu, and Shijie Zhou. Understanding and improving piece-related algorithms in the bittorrent protocol. *IEEE TPDS*, 24(12):2526–2537, 2013.

[24] Ernst W Biersack, Pablo Rodriguez, and Pascal Felber. Performance analysis of peer-to-peer networks for file distribution. In *Quality of Service in the Emerging Networking Panorama*. Springer, 2004.

[25] Anas A AbuDaqa, Ashraf Mahmoud, Marwan Abu-Amara, and Tarek Sheltami. Survey of network coding based p2p file sharing in large scale networks. *Applied Sciences*, 10(7):2206, 2020.

[26] Petar Maymounkov and David Mazieres. Rateless codes and big downloads. In *International workshop on peer-to-peer systems*, pages 247–255. Springer, 2003.

[27] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of dht security techniques. *ACM Comput. Surv.*, 43(2), February 2011.

[28] Tobias R. Mayer, Lionel Brunie, David Coquil, and Harald Kosch. On reliability in publish/subscribe systems: a survey. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):369–386, 2012.

[29] R. L. Xia and J. K. Muppala. A survey of bittorrent performance. *IEEE Communications Surveys Tutorials*, 12(2):140–158, 2010.

[30] Pawel Garbacki, Alexandru Iosup, Dick Epema, and Maarten Van Steen. 2fast: Collaborative downloads in p2p networks. In *IEEE P2P'06*.

[31] Onur Ascigil, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. A native content discovery mechanism for the information-centric networks. In *ACM ICN'17*, pages 145–155. ACM, 2017.

[32] Filecoin. Chainsync: Filecoin chain synchronisation. https://spec.filecoin.io/#section-systems.filecoin_blockchain.chainsync, 2020.

[33] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks. *arXiv preprint arXiv:2007.02754*, 2020.

[34] Cid spec. https://github.com/ipld/specs/blob/master/block-layer/CID.md.

[35] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[36] Paul V Mockapetris. Rfc1035: Domain names-implementation and specification, 1987.

[37] Giuseppe DeCandia and et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6), 2007.

[38] Testground testing platform. https://github.com/testground/testground.

[39] Go ipfs implementation. https://github.com/ipfs/go-ipfs/.

[40] Ipfs dataset collection. https://awesome.ipfs.io/datasets/.

[41] Bitswap rfc improvement proposals. https://github.com/protocol/beyond-bitswap#enhancement-rfcs.