Beyond Bitswap: Related Work

Motivation

One of the projects the ResNetLab is currently involved in is in the analysis of potential improvements to file-transfer in IPFS. There are currently two main implementations for the exchange of content in IPFS: Bitswap (block exchange protocol) and Graphsync (graph exchange protocol). The first task we've performed in the scope of the project is a thorough analysis of the current status of file-transfer in IPFS, as well as an extensive revision of the state of the art in the field of file-transfer in P2P networks. In the process, we have not limited ourselves to the study of academic papers, but also to the recollection of ideas and proposals scattered through the IPFS ecosystem (we may have missed some of the proposals and ideas proposed within the extense IPFS community, so do not hesitate to let us know if you are missing something relevant).

In this document we present the results of all our work along with some conclusions, abstractions and a collection of new ideas that can serve as a foundation for the design and implementations of future improvements in IPFS' file-sharing subsystem.

This document extends/complements the work done here: https://github.com/ipfs/notes/blob/master/OPEN_PROBLEMS/ENHANCED_BITSWAP_GRAPHSYNC.md

File-sharing in IPFS

There are currently two main exchange interfaces implemented in IPFS: Bitswap and Graphsync.

Bitswap

Briefly, Bitswap is a message oriented protocol. When a peer wants a specific CID, it sends a WANT-HAVE message with the block CID to all its connected peers. Any node answering with a HAVE message to this request will be added to the same session. Sessions aggregate peers with information about the content being requested. If no connected peers have the block requested, Bitswap fallbacks to the DHT to find peers storing the block.

To actually request the transmission of the block, Bitswap sends a WANT-HAVE message to all peers of the session except for one of them to whom a WANT-BLOCK message is sent to request the actual transmission of the block. If this WANT-BLOCK fails, the peer can try to send it to other peers in the session.

A more in depth description of how Bitswap works can be found here: https://github.com/ipfs/go-bitswap/blob/master/docs/how-bitswap-works.md

Graphsync

Graphsync is a request-response protocol. It relies on the use of IPLD selectors to specify the content that wants to be retrieved by a node. Instead of sending flat WANT messages to its connected peers, in Graphsync an IPLD selector is included in the content request asking for a set of blocks in a IPLD DAG structure.

Implementation details can be found here: https://github.com/ipld/specs/blob/master/block-layer/graphsync/graphsync.md.

Known shortcomings of current implementations

Some of the pressing limitations identified for current implementations of content exchange in IPFS are:

- The same discovery and transmission strategy is followed for any type content exchanged. These protocols only understand blocks and CIDs (being Graphsync a bit smarter in the discovery in this sense). IPFS accommodates a great gamut of use cases, and it may benefit from the implementation of a “use case-aware” (or interchangeable) content exchange interface. Thus, the specific content discovery and transmission algorithms to be used would be steered according to the specific data to be retrieved. The download of large data files may not benefit from the same schemes as the download of small blocks.
- Bitswap follows a blind and optimistic search of content. Peers send a flat Wantlist to its directly connected peers with the hope that at least one of them has the content. If this fails, it has to fallback to the DHT to perform the lookup. Maybe it would be worth evaluating more efficient ways to “direct the search” so that we provide some kind of “a priori” knowledge to the peer for him to be able to orchestrate the content discovery more efficiently. This “a priori” information can come through the periodic exchange of information with its connected peers (similar to how is done in GossipSub, for instance?).
- These protocols are not very bandwidth efficient, as the blind and optimistic search leads to a lot of duplicate messages and blocks being exchanged. Recent improvements over Bitswap were directed on reducing this (thus the use of WANT-HAVEs and sessions) with impressive results.
- Finally, we don't have a reference testbed and a baseline benchmarking of the protocols to help us identify the “expensive parts” of the protocol. Many of the aforementioned problems and assumptions are the result of our observation of the protocol, but we don’t have a complete benchmark to identify what parts of the protocol have a bigger impact in the performance of the protocol, if it is the content discovery, the actual transmission of data, or the scarcity of bandwidth under different scenarios. To fix this, we are planning to build a testbed and a general framework to ease the evaluation of content exchange protocols in order to see if we are able to identify the bottlenecks and overheads of current implementations.

Security concerns

Bitswap also can be vulnerable to a set of potential attacks:

- **Peer floods a local node with useless blocks:** A peer sends many blocks to the local node that the local node doesn’t want, flooding the local node’s bandwidth. Mitigation: Deprioritize peers that send many unwanted blocks.
- **Peer sends HAVE message when it doesn’t have the block:** The local node sends want-have to a peer; the peer responds with HAVE; the local node sends want-block to

the peer; the peer times out or sends DONT_HAVE. Mitigation: Deprioritize peers that frequently send HAVE but then don't have the block.

- **Eclipse HAVE Attack:** Many sybils connect to the local node: Local node sends *want-have CID;* Each sybil responds with *HAVE CID*; Local node sends *want-block CID to a sybil*; Sybil doesn't respond, so the request times out; Repeat from the want-block stage. Because Bitswap only requests one *want-block* at a time, the sybils can introduce a delay of <*number of want-block requests to sybils*> *x* <*timeout*>*.* Mitigation: Increase parallelism of want-block requests, deprioritize peer that send WANT but don't have blocks. Prioritize peers that have been connected longest.
- **Connect flood:** Crowd honest peers. Mitigation: Favour longer lived, more stable nodes.

File-sharing protocols general framework

To ease the understanding and the analysis of the proposals and improvement ideas collected throughout this work, we abstracted the implementation of file-sharing protocols into different layers. This lays the foundation for the testbed design and frames the ideas over a common context.

Protocol Architecture

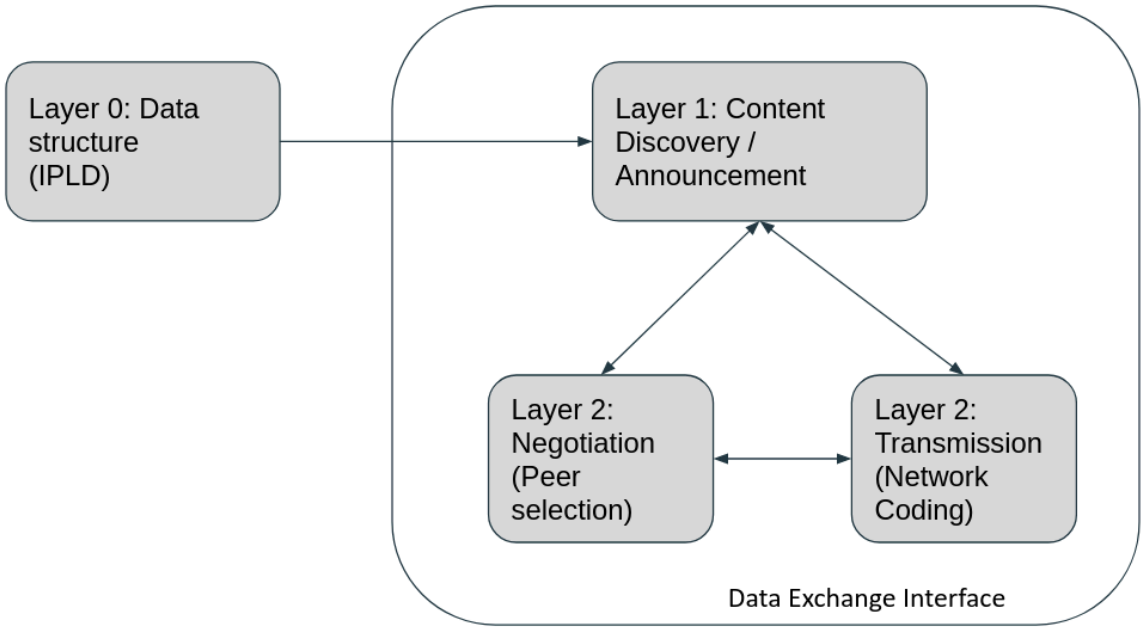
From our study of the current implementations of file-sharing subsystems in IPFS (mainly Bitswap and Graphsync) we have arrived at the following general architecture of file-sharing protocols in P2P networks. The aim of this architecture is to divide file-sharing protocols in different submodules, each with clear responsibilities, so that performance and overheads can be tracked at each of these levels. This framework would allow us to direct our improvements efforts to the less efficient part of the file-sharing protocol (this will make more sense as we progress through the document).

- **Layer 0 - Data Structure**: Layer responsible for determining the structure of the content. It determines how the data is stored, the coding scheme used to represent the data, and the specific architecture followed to store content in the network.
    - Some schemes belonging to this layer: The split of content in blocks (chunker), the use of Merkle DAGs to structure the data and store it, or the use of specific structures to represent the Providers Record to perform lookup in the network (DHT, hierarchical DHT).

- **Layer 1 - Content Discovery and Announcement:** It specifies all the schemes for the discovery of content in the network. It determines how to conform the requests and the messages to be sent to find content in the network, as well as the announcements of seen/stored content to other nodes.
    - Some schemes belonging to this layer: The use of control messages (Wantlists, WANT-HAVE, etc.). The use of sessions to discover nodes with high probability

of having the content, and peer selection algorithms used to select to whom the content will be requested.

- **Layer 2 - Negotiation and Content Transmission:** Along with the discovery of content, there may optionally be a negotiation phase so that once the content has been discovered its transmission is “formally” requested. This layer implements schemes to negotiate the request of content to the most suitable peers available as well as the actual transmission of blocks from the network. It negotiates and opens a transmission channel between the requestor node and the provider of the content. In some implementations this layer may be embedded in layer 1.
    - Some schemes belonging to this layer: Peer selection to ensure the higher throughput, optimal path selection, or the use of network coding to add redundancy in the transmission of blocks to requestor peers (more about network coding below).



Use Cases

In this section we present an overview of general use cases that can potentially be implemented using IPFS. For each use we analyze its specific requirements they may impose in the file-sharing subsystem in order to operate successfully. *There’s a lot of room for discussion in this section (as well as in the overall document), so do not hesitate to refute these claims, complement it with real data, or suggest additional use cases.*

Use Case	DAG Structure	Latency needs	Throughput needs	Sync / Ordered	Churn impact	Parallelizable
Live Stream	Wide tree	High (depends QoS / codec)	High	Yes	Withstands a small number of lost blocks. Replicas may be increased to better availability and throughput.	Depends on DAG structure used
Messaging	Depends implementation	Low	Low	Potentially yes	Low number of replicas. Small files. Events in the network.	No
Large Files	Wide tree	No	Very high	No	Enough replicas to ensure availability	Yes
Databases	Wide and deep tree (Database -> Schema -> Docs).	< 2s	Mid-high (depends on the size of the data)	No	Enough replicas to ensure availability	No for single queries. Yes in batch
Frontend Backend Assets	Wide tree	< 1s	Low	No (except index)	Low # replicas. Accessed regularly	Yes
Blockchain Ledgers	Narrow and deep tree.	Low	Low	Depends	Enough replicas to ensure availability	Yes in-sync, no specific queries

VoD	Wide tree	High (depends QoS / codec)	High	Yes	Withstand a small number of lost blocks. Replicas may be increased to better availability and throughput.	No

Request Patterns Topology

From the use cases above, we may infer a set of common request patterns. This extends the request patterns analysis presented here.

- **Request of specific blocks:** The node is requesting a specific CID from a DAG structure such as: <Root_CID>/link1/link2/<CID>. This pattern requires the discovery of the ROOT_CID, traversing the DAG until we reach the CID, and then the request of the blocks for <CID>. This pattern is widely seen when requesting specific files from a filesystem, blocks of a blockchain ledger, or frontend/backend assets for a web application.
    - Important metrics: This specific pattern would benefit from low latency, fast discovery of the specific CID, data availability and high churn resistance.

- **Request of full DAG structures:** The node is requesting the full DAG structure either from its root CID: <DAG_CID>/*; or from a certain level: <ROOT_CID>/link1/<DAG_CID>/*. This pattern requires the discovery of the DAG_CID and the discovery and transmission of all the blocks belonging to this DAG. This pattern is used for blockchain synchronization and the download of large files.
    - Important metrics: Full discovery of the DAG without requiring full traversal, bandwidth efficiency (there are a lot of blocks to be requested) and high throughput. I also may benefit from “a priori” information of the structure of the tree.

- **Request of a custom subgraph in a DAG structure:** When a specific subset of the DAG wants to be retrieved. In this case we are not requesting the full DAG but a subgraph (all the leaves, a branch of the DAG, the right side of a symmetric graph, etc.). The only efficient way of achieving these complex requests is by using IPLD selectors. This pattern requires the parsing of the IPLD selector, and the retrieval of all the CIDs

fulfilling the request. This pattern may be useful for databases, large datasets or messaging apps.
    - Important metrics: Smart discovery of blocks and bandwidth efficiency. Churn resistance.

- **Ordered transmission:** The node requests an ordered transmission of the blocks conforming a DAG. In a request such as: <ROOT_CID>/link1/<DAG_CID>, the upper levels and left side of DAG_CID should be discovered first and transmitted before the leaves which needs to be transmitted in an orderly manner. Improved IPLD selectors may be needed to implement this pattern efficiently.
    - Important metrics: Low latency. Ordered delivery. Caching, buffering schemes?

- **Regularly accessed content:** The node requests content regularly accessed in the network. This may be useful for web caching, filesystems, registries, name systems, etc. If the regularly accessed content has the following CID and belongs to the following DAG: <Root_CID>/link1/link2/<CID>, both requests should lead to the same number of requests and control messages in the file-sharing protocol. Requesting through the full path shouldn't lead to additional messages. Some kind of pre-defined WANT messages, hashed wantlists, or caches may be required for this specific content.
    - Important metrics: Fast discovery. Caching schemes.

More complex request patterns may be devised, but in our opinion they will end up being a combination of the ones mentioned above. Thus, in order to request the content to load a large video and be able to playback and seek, the resulting content requesting will be a combination of three of the above patterns: (i) request a subgraph with the first blocks of the video and then (ii) all of the tree structure except the leaves; finally (iii) request the full DAG (all the blocks of the video) in an orderly manner.

Other examples of use cases and request patterns can be found here: https://github.com/ipld/specs/blob/master/block-layer/graphsync/graphsync.md#example-use-cases

State of the art

In this section we will present some of the main improvement ideas proposed by the community (in Github Issues or public forums) and academia that could serve as an inspiration for potential improvements to file-sharing in IPFS. Every proposal will be mapped to the specific layer in the protocol we think may benefit from the idea. *Our aim is for this list to be "alive" so every new paper or community proposal around file-sharing in P2P networks can be suggested and appended here if considered relevant enough. So feel free to suggest additions to these lists.*

Community Proposals

Many of the candidate open problems and proposals collected throughout the work have already been considered or implemented in the latest implementation of Bitswap. Thus, we only add the ones not yet added in the protocol. The baseline implementation of Bitswap considered in the analysis are the ones discussed here: https://github.com/ipfs/go-bitswap/issues/186 and merged here: https://github.com/ipfs/go-bitswap/pull/189.

Proposal	**How it works**	**Use Case**	**Potential impact**	**Protocol Layer**
Paralellize DAG walks	We have a priori information of the structure of the DAG. We traverse the structure in parallel.	Useful for symmetric DAG structures and when the full structure wants to be traversed.	Same number of RTTs, less latency. Probably required more bandwidth.	Layer 1 - Content Discovery and Announcement
Bitswaxx (@alanshaw) - 2 hop Bitswap	Your wantlist becomes your peers wantlist. Increases scope of the search.	Useful for the discovery of rare files far from the source.	Increased bandwidth and storage requirement in peers for others wantlists. Faster discovery	Layer 1 - Content Discovery and Announcement
Bare hash wantlist	We request a wantlist hash instead of a full wantlist.	Useful for content in the network requested regularly.	Fast discovery. Less overhead and bandwidth. Requires storage of common wantlists.	Layer 0 - Data structure
Reed Solomon	Add redundancy when storing blocks of content.	Useful for large files	Increases the number of blocks but eases the discovery and	Layer 0 - Data Structure

			transmission.	
Implementation of Reed Solomon over IPFS	Example of the implementation of Reed Solomon over IPFS	Resilient storage in IPFS	Churn resistance	Layer 0 - Data Structure.

Papers, Patents and Publications

Paper	**Protocol Layer**	**Main idea**
Rate sensitive packet transfer mechanism over a peer-to-peer network	Layer 2 - Content Transmission	A pool of nodes collaborate to send the packets to the node increasing the overall bandwidth of the transmission.
2Fast: Collaborative downloads in P2P networks	Layer 2 - Content Transmission	Delegate the download of files to a group of nodes. A node shares bandwidth for a promise for future bandwidth from that node.
Performance Analysis of Peer-to-Peer Networks for File Distribution	Layer 0 - Data Structure	Analyses the use of tree, linear and forest architectures for chunk distribution. May be worth reading it for ideas on how to chunk and manage blocks in IPFS.
QoS Prediction for Neighbor Selection via Deep TransferCollaborative Filtering in Video Streaming P2P Networks	Layer 2 - Content Discovery and Announcement	Use deep learning to generate a model that enables nodes to evaluate QoS of other nodes, and select them accordingly for content transmission. Potential alternative to Bitswap's current peer selection scheme.
Improving Media Services on P2P Networks	Layer 2 - Content Discovery and Announcement	Conform a weighted graph to choose the "best path" for the exchange of content according to the advertised cost. This relates with the idea presented above of having "a priori" information for the discovery and efficient transmission of content.

Data Location Management Protocol for ObjectStores in a Fog Computing Infrastructure	Layer 0 - Data structure	DNS-like tree structure to locate content in the network. Applies a Dijkstra-like algorithm to identify the “best path”. Includes space locality information for the discovery in the structure of stored blocks.
Binary search routing equivalent (BSRE): A circular design for structured P2P networks	Layer 1 - Content Discovery and Announcement	Binary search routing equivalent, a circular design for structured P2P network. Alternative to DHT-based search.
Survey of Network Coding Based P2P File Sharing in Large Scale Networks	Layer 2 - Content Transmission	Survey of different network coding techniques (full-coding, sparse coding, generation-based, etc.). Interesting field to explore for the transmission of blocks.
Understanding and improving piece-related algorithms in Bittorent Protocol	Layer 1 - Content Discovery and Announcement	Analyzes different piece selection algorithms. Proposes the dynamic management of unfulfilled requests. The ideas presented here may give inspiration on smarter ways of requesting blocks to the network. It also proposes the use of signals between connected nodes to get knowledge about what is happening with content in the network.
Cooperative Caching for Efficient Data Search in MobileP2P Networks	Layer 0 & 1	Conform cache cluster between different connected nodes. Search is performed using metadata received from neighboring peers. Nodes collaboratively build a cache cluster storing content from the network. To discover who is storing the data there is an exchange of metadata between them (relate to GossipSub?).
MDHT: A Hierarchical Name Resolution Service forInformation-centric Networks	Layer 0 - Data Structure	Name-based anycast routing using a hierarchical DHT. Location-dependant overlay architecture. Maps the architecture of the Internet.
Really Truly Trackerless BitTorrent	Layer 1 - Content Discovery and Announcement	Proposes the use of graphs for piece discovery. It removes the Bittorent tracker completely. It is really focused on node discovery but the ideas may be extrapolated to content discovery. Would it make sense to have random walker

		supernodes (behaving as decentralized trackers) that share some information about how content is distributed in the network?
Sloppy hashing and self-organizing clusters	Layer 0 - Data Structure	Proposes a new way of indexing (and storing) data in the network. It uses an alternative DHT structure where the same key may store several keys depending on the node storing the data. Requests for the same key to different nodes may lead to the reception of different content.
Peer-to-Peer resource discovery in Grids: Models and systems	Layer 1 - Content Discovery and Announcement	Construction of flat P2P overlay networks. Use of super-peers with metadata about the rest of peers (similar to trackers). One DHT per attribute.
Broccoli: Syncing Faster by Syncing Less	Layer 2 - Content Transmission and Negotiation	A blog post describing new improvements in file syncing from the engineers at Dropbox. They propose the use of compression in the transmission of blocks. It is preceded by a negotiation phase between client and server. Not specific for P2P networks but may be useful as inspiration.
Rateless Codes and Big Downloads	Layer 0 - Data Structure	It uses rateless erasure codes to enhance the transfer of data between peers. It performs linear combinations between blocks. Good explanation also here
On-Demand Routing for Scalable Name-Based Forwarding	Layer 1 - Content Discovery and Announcement	On-Demand Routing(ODR) computation for content name prefixes as interests arrive.ODR makes use of domain-level, per-prefix routing instructions usable by all the forwarders in a domain, namedRouting InformationObjects (RIO). This could be useful to build “content routing tables”.
A Native Content Discovery Mechanism for theInformation-Centric Networks	Layer 1 - Content Discovery and Announcement	Use opportunistic content discovery in the path.

The ideas collected from the analysis of the state of the art may be summarized per layer as follows:

Protocol Layer	**Ideas**
Layer 0	Chunking schemes Structure and storage of data in the network (hierarchical architecture, alternatives to DHT).
Layer 1	Build graphs or overlay networks to ease the discovery of content. Use of supernodes to track content in the network. Node clustering for efficient content discovery. Exchange of metadata to share local views of the network.
Layer 3	Use of network coding and compression in the transmission of content. Delegate the download of data to a group of nodes (and maybe content discovery). Finding the “best path” for the transmission of data.

Baseline Test Framework

In this section, we present the different metrics and test case scenarios we are currently considering for the building of our testbed and the evaluation of the protocols. *This is another important part where we really appreciate as much feedback as possible.*

Proposed metrics

Metric	**How to measure it**
Time to fetch / Latency	Time elapsed since the request of the content to its full transmission. It will be the sum of the time to full discovery and the transmission time.
Time for full discovery	Time required to discover all the blocks comprising the requested content. This time shouldn’t account the time required to send the data through the wire.
Transmission time	Time from the transmission of the first bit to the full transmission of the content requested. To effectively measure

	this for content comprising several blocks we need to remove the time required to discover subsequent blocks.
Overall throughput	Amount of data per second sent by all nodes serving the actual content. To accurately measure this we shouldn't account for duplicate content. Ideally we should only focus on the actual content requested.
Control Message Counters	Counters to identify the number of control messages exchanged by the protocol (WANT, HAVE, WANT-HAVE, content requests, etc.). Is a good way of understanding the behavior of the protocol under different scenarios.
Bandwidth efficiency / Protocol Overhead	Rate of useful data exchanged through the wire. A rate would mean that through the wire only data related to the content is exchanged (bandwidth efficiency = 1). This metric measures the protocol overhead. It would be interesting to compare this metric with existing file-exchange protocols such as TCP exchange, FTP or BitTorrent.
Rate of data loss	Number of requests for content unsuccessful or lost (requiring retransmission or additional requests).
Computational footprint	Computational resources used by peers (CPU, RAM, etc.)
Rate of data duplication	Number of duplicate messages received. This metric could give good indication about if the protocol would benefit from caching schemes.
Number of times we resort to the DHT	Number of times the file-sharing protocol is not able to discover blocks and has to resort to the DHT to find providers.
Latency overhead	Time required to download content compared to TCP / FTP / Bittorent

Some of these metrics may require the addition of loggers or trackers on protocol implementations to effectively measure them (these probes shouldn't impact the operation of the protocol in any way).

Test case scenarios

To evaluate the overall performance of the protocol, the following test case scenarios will be considered.

Test Case	**Test Flow**	**Behavior to test**

Different number of connections between peers	We modify the maximum number of connections for peers. Less connection will increase the number of hops required to discover content.	We want to test the performance of the protocol for different network topologies.
Different number of seeders and leechers	We modify the seeders/leechers ratio. Lowering this ratio may lead to a congestion of seeders.	We want to test how the number of content providers affect the performance of the protocol.
High churn	We rotate nodes during the test to see how churn affects the performance of the protocol.	We want to test the churn resistance of the protocol.
Different file sizes	Change the size of the files used for the test.	We want to test the performance for different file sizes.
Different DAG structures and request patterns	Test different request patterns.	We want to evaluate the performance of the protocol for the different request patterns.
Different types of links between nodes	Have nodes with different bandwidth, latency and jitter.	Understand the performance of the protocol in heterogeneous networks.
Protocol coexistence (hot experiment)	Evaluate the coexistence of different content exchange protocols in the network. The best way to achieve this is to force the communication of the nodes of our testbed through the real IPFS network.	We want to evaluate the behavior of the protocol when coexisting with legacy nodes in a real deployment.

The configuration of all these test cases should be designed so that they test some of the use cases and request patterns presented above, that way we have a way of evaluating and fine-tuning the protocols against their desired performance.