

École polytechnique de Louvain

An Eclipse attack on content availability in IPFS, a large-scale decentralized storage service

Authors: **François GENON, Sébastien PIERRE**
Supervisors: **Etienne RIVIÈRE, Michał KROL**
Readers: **Ramin SADRE, Matthieu PIGAGLIO**
Academic year 2021–2022
Master [120] in Computer Science

Abstract

The web is an ever-changing ecosystem. New technologies emerge and shake traditional habits. A new phenomenon called *Web3* brings a new wave of decentralized applications to the web. The web is transitioning from an internet dominated by big industries to a more open internet where users gain control of their data. One actor in Web3 enables a decentralized way of hosting multiple types of content. Such a system is absolutely needed for Web3.

The Interplanetary File System (IPFS) is a peer-to-peer distributed data storage service [3]. The content is distributed among users of the network. IPFS is a stepping stone for many projects of Web3 and is considered to be one of the main actors. Since IPFS is fundamental in this new web paradigm, it is important to analyze how the system works and if there are any vulnerabilities that can affect the safety of using this system. Web3 should be an upgrade of Web2.

In this thesis, our first goal is to perform a detailed study of the content resolution in IPFS. We need this content resolution system to find where the data is hosted on the network. Once we know where the content is, we are able to retrieve it. Content resolution in IPFS is constituted of two non-trivial peer-to-peer systems. Both systems have complementary features and rely on each other to correctly serve content from one provider of data to a user. The knowledge we gather when analyzing these two protocols allows us to understand the multiple potential points of failures in IPFS.

As IPFS is a significant actor in Web3, we are interested in improving its reliability and security. The essential component of IPFS is the way it retrieves content. Based on this, we discovered an attack capable of hiding new data from the rest of the network. While not harming other IPFS users, we deploy this attack and measure the consequences it can have for the network.

By deploying this attack on the live IPFS, we highlight vulnerabilities that need to be fixed by the IPFS team. We also introduce some counter-measures in order to help them.

Contents

1	Introduction	4
2	Background	7
2.1	InterPlanetary File System (IPFS)	7
2.1.1	Content resolution system	9
	Content identifier (CID)	9
	Merkle DAGs	10
	UnixFS	11
	Protocol buffers	11
2.1.2	IPFS - Implementation	12
	Golang	13
	Mechanism Details	13
2.2	IPFS Network	14
2.2.1	Bitswap	15
	Bitswap messages	15
	Bitswap Information	19
2.2.2	Distributed Hash Table (DHT)	20
2.3	Network with content resolution	22
2.4	FileCoin	26
2.5	Problem statement	27
3	Network Attacks	29
3.1	State of the art	29
3.1.1	Sybil attack	29
3.1.2	Eclipse attack	30
3.2	Proposition of Attacks	31
3.2.1	Sybils in IPFS	31
3.2.2	Eclipse attack in IPFS	32
3.2.3	DHT attack	32
	Get_Providers	34
	Add_Providers	35

	Challenge	36
	Consequence	36
4	Evaluation	37
4.1	Bitswap - DHT ratio	38
4.1.1	Hypothesis	38
4.1.2	Measurement of the ratio between DHT/Bitswap	38
4.1.3	Result	40
4.2	Analyzing Response Time	41
4.2.1	Hypothesis	41
4.2.2	Measurement of response time	41
4.2.3	Results	41
4.3	Analyzing Bitswap connections	43
4.3.1	Hypothesis	43
4.3.2	Measurement of Bitswap Connection	44
4.3.3	Result	44
4.4	DHT Add contact	46
4.4.1	Hypothesis	46
4.4.2	Measurement of the interaction in DHT	46
4.4.3	Result	46
4.5	DHT Attack Development	47
4.5.1	Tools	47
	Generating closest peers	47
4.5.2	Setting up Sybils nodes	48
	Experimental setup	49
	Virtual machines	50
	Cost	50
4.5.3	Attack on adding the content	51
	Hypothesis	51
	Modified Client	51
	Attack execution	52
	Result	52
4.5.4	Attack on the Get content	53
	Hypothesis	53
	Modified Client	53
	Attack execution	53
	Result	54
4.6	Attack Measurement	55
4.6.1	Hypothesis	55
4.6.2	Impact capture	55
4.6.3	Result	56

5	Counter-measures	57
5.1	Contact More Peers	57
5.2	Avoid getting closest peer of CID	58
5.3	S/Kademlia	58
5.4	Certified nodes	60
5.5	Scoring	61
6	Future Work	62
6.1	Priority want attack	62
6.2	Privacy Attack	63
6.2.1	Analyzing current and future requests	64
6.2.2	Analyzing past requests	64
6.3	Consequences for Filecoin	65
7	Related Work	66
7.1	The Sybil Attack	66
7.2	Low-Resource Eclipse Attacks on Ethereum’s Peer-to-Peer Network	66
7.3	Mapping the Interplanetary Filesystem	67
7.4	Disrupting the InterPlanetary File System	67
8	Conclusion	68

Chapter 1

Introduction

In recent years, a new paradigm for the web has emerged, *Web3*. The goal of this new paradigm is to make the internet more open and free of monopolies like Google, Facebook, and Microsoft. To avoid these monopolies, Web3 promotes the use of decentralized and blockchain-based technologies. A decentralized network gives more power to the users due to the absence of a central authority. To transition from Web2 to Web3, new systems have emerged and replaced centralized applications. For example, with the use of a decentralized identity a user can own their data and make it compatible with websites he visits. This allows the user to control which of its data is shared.

Another application needed by Web3 is a decentralized way to store data on the internet, as well as host websites. This need for decentralized storage increases at a fast pace due to the rise in interest in decentralized applications.

Web2 systems have highlighted the need for performance and scalability in systems, whereas Web3 emphasizes more resilient and censorship-resistant systems. One of these systems is called Interplanetary Filesystem (IPFS)¹. IPFS is already a main actor in the Web3 ecosystem. Multiple blockchain-based applications are built upon IPFS for their data-storage necessities².

IPFS is a distributed storage system, this means that IPFS is not storing data on big data centers like Web2 cloud companies. IPFS stores its data on the machines of its different users. This type of network where a group of computers is linked together in order to share the processing of data, storage, and computational power is called a peer-to-peer network. IPFS provides a way to access the data by storing the data on some of the participants and allows collective file storage. To know where one piece of data is stored among IPFS users, it uses the help of two

¹<https://ipfs.io/>

²<https://awesome.ipfs.io/apps/>

complementary protocols. The fact that data is stored among IPFS users makes it censorship-resistant, since multiple users around the world may host a single piece of content. This property also makes it more robust against failures.

IPFS is built by Protocol Labs, an open-source R&D lab. IPFS is an open-source project so, in addition to Protocol Labs³, anyone interested can help maintain and improve it. IPFS community is constantly growing and as of 2019, there were more than 4000 open-source contributors.

According to Protocol Labs, IPFS has millions of users per week on their HTTP to IPFS gateway⁴.

Cyber Security threats are increasing in the world. As IPFS is public, anyone can join the system. But, no one can kick users from it, it makes it vulnerable to malicious users. So, it is crucial to make the system as robust as possible to make it resistant to malicious behaviors. This is an issue for all systems and also IPFS. The security of IPFS is our main concern in this paper. We would like to give the opportunity to Protocol Labs to increase it. The main vulnerability of IPFS is accessibility to its network, but it is also its strength. For this reason, we will evaluate the implementation of IPFS to distinguish the implication of every technology involved in it. This information is a necessity for the rest of this paper.

IPFS is a recent system and has not much literature available, but it is rapidly growing in recent years. The paper from Prünster et al. [19] demonstrate that Kademlia in IPFS is also not free of attacks. They can hide a user from the IPFS network. In our paper, we demonstrate that it is possible to hide content and make it unattainable to other users by exploiting vulnerabilities in Kademlia. Our kind of attack is a Denial-of-Service (DoS). DoS target the attack that makes unusable or unreachable service. In our case, content is unreachable for honest users. We demonstrate how to execute this attack on the live IPFS network without harming other users. We are closely communicating with Protocol Labs to give details about this attack and help resolve this issue. The IPFS network is censorship-resistant, but our attack puts a dent in this property by blocking access to a given piece of data. On IPFS, content is available through Kademlia, and with one other technology. We are able to block the availability of a given content on Kademlia.

We determine the implication of Kademlia from the other technologies used by IPFS to evaluate the concrete impact of our attack on the content availability.

In our security analysis of IPFS, we highlight vulnerabilities in content availability. As said, our main concern is to protect IPFS from attacks by analyzing their security vulnerabilities through their use of Kademlia. For this reason, we inform Protocol

³<https://protocol.ai/>

⁴<https://blog.ipfs.io/2020-02-10-our-focus-for-2020/>

Labs about the possible resolution of this security breach. We also suggest counter-measures to mitigate our attack.

Chapter 2

Background

Before inspecting the security of IPFS, we make a detailed overview of the different technologies integrated into its system. This overview is based on available documentation and from our code analysis of the implementation of IPFS. Indeed, IPFS is an ever-growing system and often makes new releases. Due to the speed of their development, the documentation is hardly maintained and it is necessary to explore the code base in order to understand the latest mechanisms used by IPFS.

In this chapter, we first discuss the main component of the content resolution system. Then, we primarily focus on the IPFS network to analyze the different interactions of the implementation with Kademlia in order to be able to evaluate the security risk of this integration.

2.1 InterPlanetary File System (IPFS)

IPFS has a *peer-to-peer* (P2P) network architecture. A peer-to-peer network is a network where multiple computers collaborate to share a workload. The computers participating to this network are called *peers*, or *nodes*. Each peer will reserve a portion of their resources, such as disk storage or computational power for other peers to use. Every peer on the network has equal privileges. This property of peer-to-peer networks makes it possible to create a decentralized environment. Since all nodes have equal privileges, there is no need for a central authority.

Peer-to-peer networks have multiple applications. The most popular is for file sharing and it is what IPFS is using this technology for. Each peer in the network will help store and share files with others.

IPFS is a distributed storage system which uses a content identifier called *CID* to identify each content¹. The purpose of IPFS is to retrieve the information given the content, which implies that before being able to provide the content to a user, we need to find its location. Once the location is found, the user can download the content. This system retrieving the information given an identifier for the content is called content addressing. The identifier (CID) is simply a hash of the content. When a user asks for data, he generates a request for the corresponding CID. The IPFS system serves the data to the user thanks to all peers in the network collaborating to find where the data is located. To locate the data among the thousands of peers in the network, IPFS uses two protocols that will collaborate to fulfill the request. These two protocols are called *Bitswap* and *DHT*.

Once the user has received the requested data, it will become a provider for it and help serve requests from other users in the IPFS network. It also means that multiple nodes can provide the same content [1].

Having multiple peers hosting the same content adds redundancy to the network. When taking this into account in addition to the decentralized aspect of IPFS, this network allows to:

- **Support a resilient internet:** There is always a way to find the data, even if a single node crashes. Each content is stored on multiple machines due to the distributed system.
- **Better content delivery:** A Content Delivery Network (CDN) is a network of nodes storing content in places physically close to the users. Having servers geographically closer to consumers assures lower latency, enables load balancing, and allows to adjust content availability with demand. The IPFS network is designed to be a CDN since each node caches what it consumes and serves that material to its neighbors².

Moreover, the benefit of using a decentralized approach is to avoid having a single point of failure. If IPFS used a central peer to manage all requests, we would need to manage closely the system and resources due to the high variability in demand from all nodes in the networks.

Due to the decentralized aspect of IPFS, anyone could join the network, deploy nodes on the IPFS overlay network, and participate in the global storage service. It is a permissionless system. Every node has its own respective **PeerID** which corresponds to the hash of their public key, $H(k_{pub})$ ³. Indeed, any ID is formed by

¹<https://docs.ipfs.io/concepts/what-is-ipfs/>

²<https://docs.ipfs.io/concepts/usage-ideas-examples/>

³<https://docs.ipfs.io/concepts/persistence/#persistence-versus-permanence>

a public and private key. Every public key corresponds to a private key. A node must have two correct keys to join the network.

2.1.1 Content resolution system

In IPFS, the content resolution system allows any user to retrieve the information. In contrast to browsing the Web2/HTTP, IPFS does not search the file given its localization but given its content ⁴. To achieve this task, IPFS uses a Content Identifier (CID), which corresponds to the hash of given content. In this section, we introduce how the data is represented to effectively retrieve it in the peer-to-peer system.

Content identifier (CID)

A **content identifier (CID)**, is a self-describing content-addressed identifier based on a key/value store where the CID would be the key and the value the content. It does not specify where the content is stored, but it creates an address dependent on the content. The length of the CID is not at all correlated to the content's size because the CID is a hash of the content. However, a new CID is created in IPFS when file content is modified and if two nodes introduce the same content in IPFS, they will produce the same CID ⁵.

All content in IPFS is hashed using the default hashing algorithm **sha-256** which results in a 256-bit CID. The use of a cryptographic hash allows noticing any changes made by a third party in the content. When someone downloads a file, he can hash it and compare it to the CID to be sure it corresponds. It is important to use a hashing algorithm with a good collision-resistance property to avoid someone recreating the same hash from different content. The fact that IPFS is hashing the content to generate the CID, means that the CID is deterministic. This has multiple advantages such as a way to prove the authenticity of the content quickly.

IPFS is destined to store files of any size. But the Content Identifier, when working with bigger files, can only identify parts of a file. To store and retrieve all parts of a large file, we use a Merkle dag to split and retrieve the data of this kind of file. The default value of the size for parts is 256KB in IPFS ⁶.

⁴<https://proto.school/anatomy-of-a-cid/01/>

⁵<https://docs.ipfs.io/concepts/content-addressing/>

⁶<https://docs.ipfs.io/how-to/work-with-blocks/#what-to-do-with-blocks>

Merkle DAGs

A Merkle Dag allows splitting a large file into small blocks of data where every block points to the next one in order to retrieve the original file. This definition could be basically represented as a chain of nodes where every node corresponds to a block of data and the chain introduces the order of each block in the content.

In fact, Merkle Dag is a directed acyclic graph (DAG) where each node gets an identifier (CID), and this is the result of hashing the node's contents using a cryptographic hash function like **SHA-256**⁷. Indeed, Merkle Dag is a tree that may be unbalanced and where nodes could have multiple parent nodes. A DAG like in figure 2.1, is a directed graph where there is no loop. The IPFS content resolver allows the retrieval of different parts of a file without looping over the same subset of the file. Each node of a DAG is a part of the big file stored in IPFS and the link between the node is embedded into the sources.

Merkle DAG allows multiple advantages like:

- Content Addressing;
- Tamper resistance: The content is verified with the checksum;
- Duplication: On the same node, we will not store twice the same document.

[4]

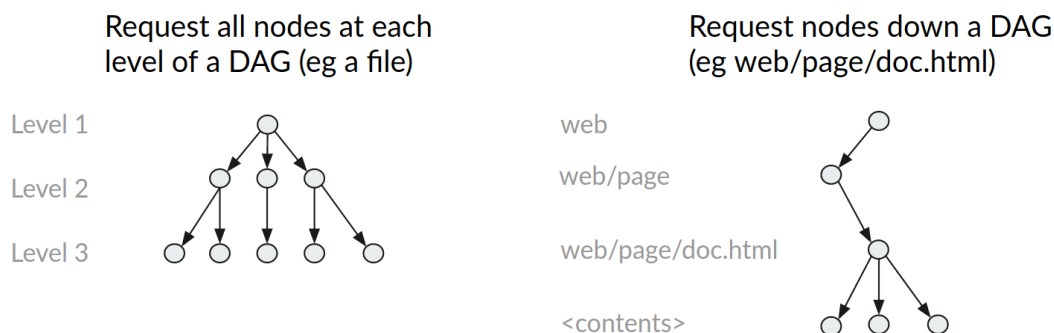


Figure 2.1: DAG Example⁸

Merkle DAG nodes are immutable, any change to a node will affect its identifier as well as all of its ascendants [22].

⁷<https://docs.ipfs.io/concepts/merkle-dag/>

⁸The illustration of the DAG was downloaded from a presentation made by Protocol AI: https://docs.google.com/presentation/d/1mbFFGIKNvboHyLn-k26egOSWkt9nXjlNbxpmCEQfqQ/edit#slide=id.g41cabe941_0_873

UnixFS

To implement the Merkle Dag and link all blocks from the same file, IPFS uses the UnixFS protocol. In IPFS, some files are too big to fit in one block. So, IPFS uses UnixFS as a way to maintain the metadata of the file on all blocks that make up the file in an easy to share data structure.⁹

The UnixFS process has two main purposes:

- Chunking: Chunk up large files into smaller parts. Each chunk is hashed and gets its own CID.
- Layout: Determines the shape of the tree built from the chunk. If the file was small, there will be only one chunk. No intermediate UnixFS node will need to be created.

In order to read the file data, we have to perform an in-order traversal of the tree. An interesting characteristic of UnixFS is that it is a protocol-buffers-based format for describing objects.

Protocol buffers

Protocol buffers, often called *Protobufs*, provide a serialization format for structured data. Data serialization is the process of converting a data structure to a format allowing simpler storage and easier sharing.

Fig. 2.2 describes the process that UnixFS handles. It will take a file or a directory as input and build its DAG, export a representation of the DAG using multiple different objects and attributes a CID to each part of the tree. Finally, the serialization will translate each object to an easy-to-transfer and easy-to-parse representation. The process of parsing the byte representation and retrieving the directory or file structure is called deserialization.

⁹<https://docs.ipfs.io/concepts/file-systems/#unix-file-system-unixfs>

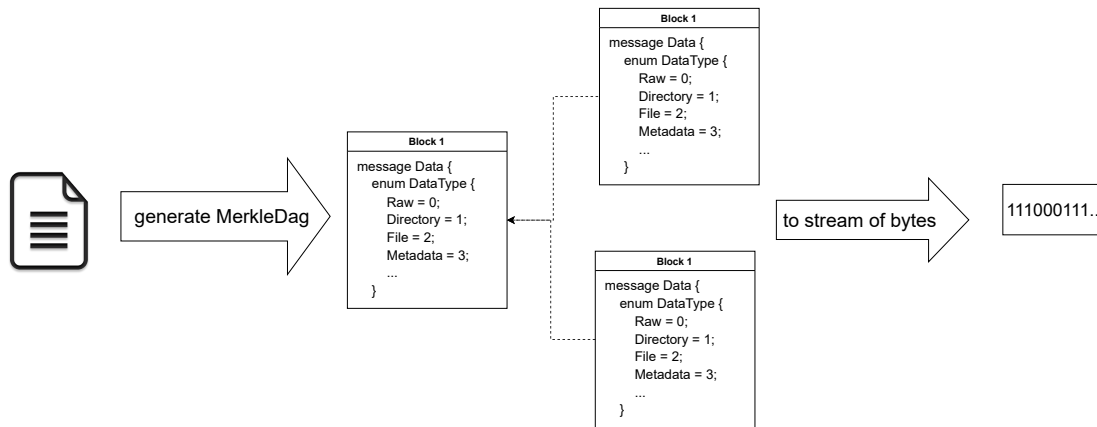


Figure 2.2: Serialization process of file in IPFS using UnixFS (vulgarized)

This has multiple benefits, such as :

- Smaller size: Allowing anybody to join the network and participate, even when their node has a low storage capacity.
- Simpler: faster parsing compared to traditional data structures such as XML, JSON,...
- Faster structure validation: Thanks to the serialization process, messages can be automatically checked for errors by the code responsible to exchange them.¹⁰

2.1.2 IPFS - Implementation

Go-Ipfs is the implementation of IPFS with tools like a command-line interface, also called CLI, to interact with our IPFS node. It also enables a user to change some default parameters of a node, such as its maximum number of connected peers, ports on which the different nodes communicate,...¹¹ The implementation of IPFS is divided into multiple repositories on Github and each of them is focused on one part of the IPFS development, we can find a Git repository for each component, such as CID, DAG, DHT, etc. The main repository, `go-ipfs`, groups all the processes.

Our analysis takes place in all these directories to get precise information about the IPFS processes. During our analysis of this distributed system, we focus on the process of adding/getting a file with a CID using the IPFS CLI.

¹⁰<https://sakshichahal53.medium.com/json-vs-protocol-buffer-simplified-dbd6b69ca528>

¹¹<https://docs.ipfs.io/how-to/configure-node/>

Golang

IPFS is implemented in multiple languages such as Javascript, and Golang. Our analysis is made on the Golang implementation because this language is the most commonly used for distributed systems and this is the reference version of IPFS.

Mechanism Details

In this section, we will discuss the different processes of `go-ipfs`. We will mostly detail the process of getting a file to explain in a generic way how each component inside `go-ipfs` interacts together and achieves the task.

Before processing the request, the node has to be launched in order to communicate with the other nodes of the network through DHT or Bitswap. This action is realised through the command-line interface provided by `go-ipfs` with the command : `./ipfs daemon`.

It should be noted that even without the daemon launched, we can process some actions with the CLI such as getting a CID from our internal storage, adding a file or repository to the system, getting the information of our node, and initializing our node with a new configuration file.

When we execute an action into the CLI, the first process of the node is to translate the command introduced by the user. IPFS has a function to analyze and transfer the request to the correct file. Indeed, every kind of command gets its own file to manage the different behaviors. When we introduce the request to get a file: `./ipfs get <CID>`, the `core\command\get.go` is the file that manages the request "get CID".

Go-IPFS is the only part of the implementation that generates the right to write/read data from the disk. The information about the availability of the information is spread in other sections. But the instance of write/read data is generated through this.

Finally, we realize the main process of getting the file given the CID. To do this, we use an API in order to get the UnixFS object.

Through the UnixFS object, we call the function `Get(context, path)` to get the file. This function will generate a new instance to access the DAG Service in Read-Only. The DAG Service allows obtaining the different blocks of the CID requested by the user. When the dag service is created, we can use it to get the file through the function of the MerkleDag Service. This call transfers the process to another module dedicated to MerkleDag implementation ¹².

¹²<https://github.com/ipfs/go-merkledag>

At this point, the process is given to the Bitswap and DHT parts. This part will be described in the Bitswap section (2.2.1). The end of this process concludes with a verification of the CID by hashing the received file and the file moved to the requested path. Otherwise, the request is not achieved and the command is aborted. Then, IPFS writes the file, folders, etc. in the repository where the action has been launched to conclude the process.

2.2 IPFS Network

In IPFS, the most important part is the communication between nodes to allow the system to work in a decentralized way. The network is based on two different technologies:

- Unstructured P2P: Peer structure does not follow a predefined deterministic scheme. The link between peer is randomly made or follow a probabilistic given the proximity metric in the network. This kind of structure is designed to support rapid information dissemination and content-based resolution like IPFS [25]. Unstructured P2P is rather resilient to nodes coming and leaving the system. However, the only way to find an object is to request all the nodes and there is no guarantee of success. [14]
- Structured P2P: A structured P2P system is a P2P system with a "pseudo"-metric space where each peer will have an identifier in this large range of possible identifiers. The choice of the neighbors of a peer is constrained in terms of the distance function of the space [17]. In addition to the attribution of peer identifiers, pieces of information in the network will be attributed an identifier, e.g. based on a hash, in the same space as the peer identifier [6]. The content identifier will be mapped directly onto nodes in order to reduce search time [21]. In summary, a structured P2P system has 2 main characteristics :
 - A "pseudo"-metric space for identifiers
 - A distance function to measure space between identifiers in order to define which peers will be neighbors, as well as how data will be mapped in the network

The assignment and use of identifiers are essential to the correct operation of the network. It is also its weakness. People may create fake identities to subvert content.

In the following content, we will discuss two networking systems used by IPFS. IPFS uses Bitswap as unstructured P2P and it uses DHT as structured P2P. Both are used in a collaborative way.

2.2.1 Bitswap

The Bitswap protocol from IPFS was inspired by the BitTorrent protocol. BitTorrent is a communication protocol for peer-to-peer file sharing over the internet with a decentralized system[4].

Bitswap is a message-based protocol used at the core of IPFS where its messages contain information about block possession and needs in the network or directly data from a block¹³. A major ability of Bitswap is to traverse a hash-linked graph as Merkle dag and retrieve the different parts of it from different peers. This system is used to request and send blocks from and to other peers in the network. Every node has a set of blocks that it wants to acquire (`want_list`), a set of blocks it can provide (`have_list`), and a set of nodes called neighbors or peers. Every node can get and provide blocks to any node included in its neighbors.

In IPFS, this protocol is the only one used to exchange blocks. The limitation of Bitswap is that we only ask our neighbors and we cannot be sure to find a block in a subset of nodes in the IPFS network [4].

The topology could be summarized as the figure 2.3. Every node has its own set of neighbors with which it can interact to get the file it needs. However, the node does not flood the request through the whole network. The interest of this approach is the speed of the response. The request is directly answered by all the neighbors.

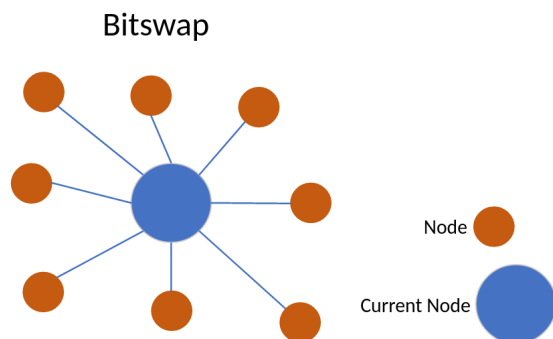


Figure 2.3: Bitswap abstract representation

Bitswap messages

As said previously, the range of the Bitswap protocol is limited to the peers of a node. The first peers connected to one node are called the *bootstrap* peers and are

¹³<http://docs.ipfs.io/concepts/bitswap/>

encoded in the configuration file of a node. The bootstrap peers are given and run by the IPFS team.

Bootstrapping is an important security point of failure and it is very important to treat all given nodes with caution. Having malicious bootstrap peers may introduce a node to other malicious peers. Once a node has multiple connections, it may execute the multiple operations allowed by IPFS, such as retrieving a file.

A Bitswap request to query a file may begin in 3 ways :

- by sending a **want-list**
- by sending a **want-have**
- by sending a **want-block**

A *want-list* is a list containing multiple CIDs requested by the node itself and its peers. Whenever our node receives a block, it will check its want-list and check if a peer was looking for this CID beforehand. If it was, the node will send it to the corresponding peer. Because of this system of looking out for Bitswap peers, a node should tell its peers when the CID he queried has been found. This is done by sending a **cancel** message, as illustrated in figure 2.4. If no **cancel** message is sent to all peers after receiving the correct block, other peers will continue sending the block to our node, thinking that we still need it.

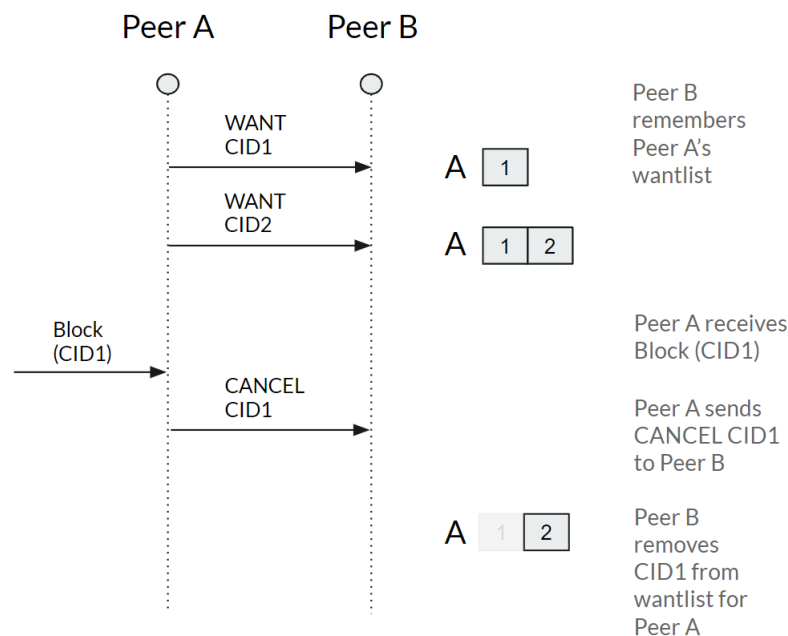


Figure 2.4: Usage of the **cancel** message in the Bitswap protocol¹⁴

Each entry in the want-list has a priority level as well as a **want-type**. The **want-type** could be :

- 0: 'Block' - for directly requesting a block. This heavily speeds up the process of getting a file, especially when it is a small one.
- 1: 'Have' - to ask if a peer has the corresponding block. This is better suited for heavy files. If a node directly asks for all blocks of a heavy file, all peers would send blocks at the same time and use a large amount of bandwidth.

The **priority** is a 32-bits integer and is directly copied when one node appends another want-list. The order of processing each want will be sorted by priority. The types of wants having the bigger priority, meaning that they will be handled as fast as possible, are the :

1. earliest want-haves
2. earliest want-blocks; for a request with the same CID, the want-have has a priority on the block request.

The `want_list` is sent when :

- A new neighbour joins
- After a random time
- After a change in the `want_list`
- After receiving new blocks.

¹⁴The illustration of the cancel message was downloaded from a presentation made by Protocol Labs: https://docs.google.com/presentation/d/1mbFFGIKNvboHyLn-k26egOSWkt9nXjlNbcpmCEQfqQ/edit#slide=id.g41cabeb941_0_873

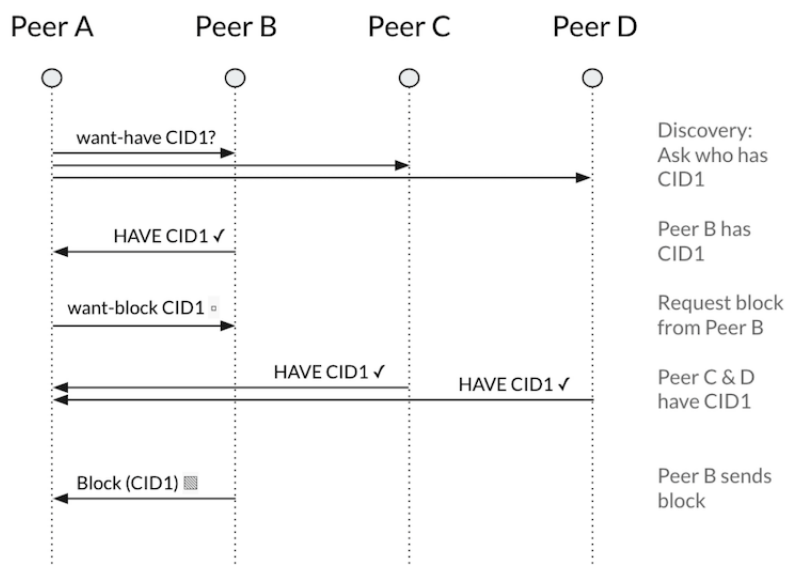


Figure 2.5: Want-have representation from IPFS documentation

As a basic example, we could take the figure 2.5 where we represent the request made for getting a given CID¹⁵. Peer A asks through the want-list if peers B, C, and D have it with the type "Have". The other nodes process the request and peer B provides a positive response. Peer A will send back a request through the want-list with the type "Block" for the CID. In the meantime, the other node responds but peer A waits for the response with the block from peer B.

When a node gets the `want_list`, it stores it locally even if it does not need it directly. Then, it checks if there is any block that it could provide. If it is the case, it provides it. After receiving the block, the node that requests the block becomes a new provider of it.[4]

As we just discussed, wants are communicated using a want-list. Sometimes a list is not necessary and unique messages are more appropriate. To know if a peer has a certain CID, it can send a single `want-have` message. And to directly try to download a block, even if we are not sure that a peer provides it, we can send a `want-block` message.

We have covered how one node transmits its wants but not exactly how one peer answers the request. From a peer point of view, after a want has been received, it can answer the request in multiple ways:

- If the peer is a provider of the CID present in the want request, it responds with a `HAVE` message. The node that queried the CID will then send a

¹⁵<https://docs.ipfs.io/concepts/bitswap/>

want-block message to get the block.

- If the peer is not a provider of the CID present in the want or block request, there are 2 ways to answer:
 - by sending a **DON'T HAVE** message which would explicitly tell the other node that it does not own this specific CID.
 - by not answering the request

It is to be noted that some nodes in the network do not support Have messages and can only directly answer with a block. This is handled beforehand by the message queue process in Bitswap. Before sending blocks or messages about the want list, the Bitswap protocol will gather if the remote node is compatible with Have messages.

Bitswap Information

The process of requesting a block is more complex for two main points:

- each node stores the block provided by their neighbors
- we request the root CID of the DAG

As explained, we store locally a list of CIDs that we can provide, CIDs we need, and gather information provided by our peers. With this information, Bitswap uses a probabilistic algorithm to ask neighbors that have a better chance to retrieve the CID. When searching for a CID that a neighbor is also looking for, we directly ask this neighbor. If they do have the CID, we gain time. Otherwise, we request another neighbor for the CID. It is possible that there are no neighbors that have the CID requested and we have to use another system called DHT to find the peer having the CID.

Furthermore, when a neighbor becomes too useless for a node, this node is removed from its connections. For making this decision, a node gathers multiple statistics about its peers. For example, one node that only gets blocks while never providing can reduce its score.

About the root CID of the DAG, we can only request the root block of a CID for the first time. This means that we will request a series of blocks after the first parts. It makes the get process a little more complex because we cannot directly ask for all the CID information without the root block.

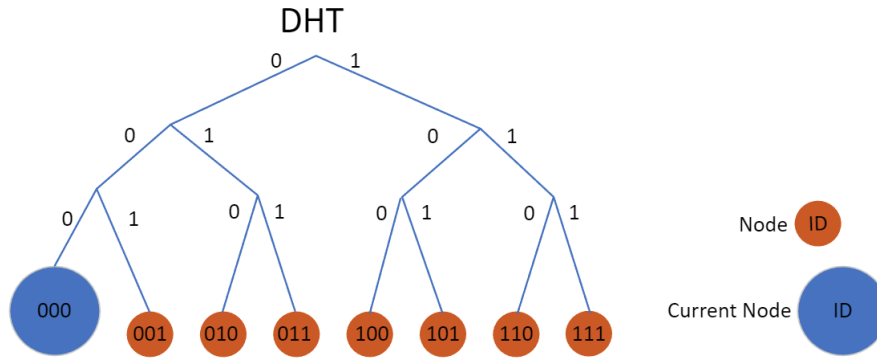


Figure 2.6: Representation of Kademlia

2.2.2 Distributed Hash Table (DHT)

Distributed Hash Table , commonly called DHT, is a peer-to-peer distributed system for mapping keys to values. DHT is a distributed hash table, it means that it is a structured graph where it maps a set of nodes into a large identifier space in a deterministic and distributed way.[20] The IPFS implementation of DHT is inspired by Kademlia [12]. Kademlia minimizes the number of configuration messages nodes must send to learn about each other, it is the advantage of this protocol compared to other approaches[16].

Each value is mapped to a key in DHT. In IPFS, each key in the DHT corresponds to a CID and the value mapped to the key will be a provider of this CID. In IPFS, the CID and the Peer ID are encoded in a 256-bit address space. Kademlia provides a lookup algorithm that locates "closer" nodes to any desired ID[16].

Kademlia is represented in the figure 2.6 where a leaf represents a node. This tree represents the address space of DHT where every node is an identifier, CID, or PeerID. The edge defines a part of this identifier space in this system. The system ensures that every node knows of at least one node in each of its subtrees if there is a node.

The DHT will allow anyone to find who owns what data inside the network. The DHT in IPFS is multipurpose and manages multiple different key-value pairings, 3 to be exact:

- Provider records: It maps a data identifier, such as a CID, to a peer that advertised being a provider to this content
- Peer records: It maps a peerID to a set of multi-addresses (IPv4, IPv6,...) at which the peer may be reached. This is mostly used when we know that some peers have the content we looked for but we do not know its exact address.

- IPNS records: This is the equivalent of DNS on the IPFS network

In this thesis, we will be most interested in two of them: provider records, as illustrated in figure 2.7, and peer records.

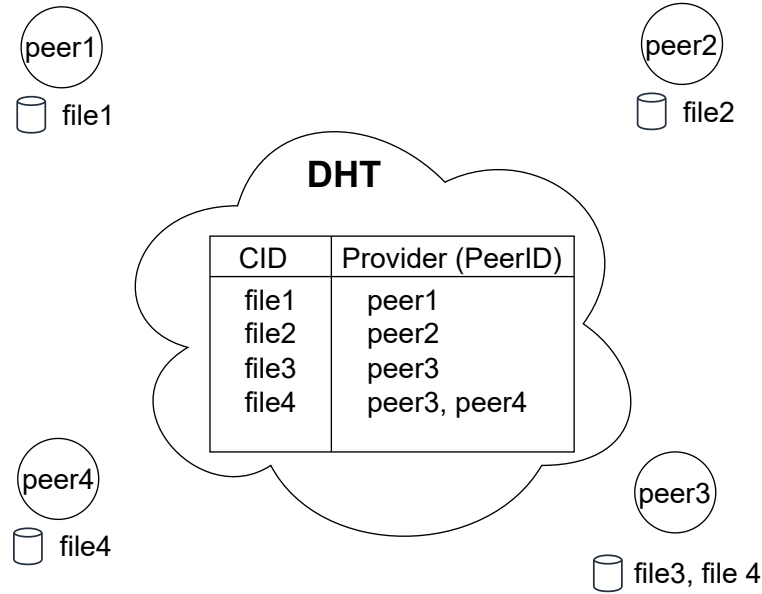


Figure 2.7: Representation of the DHT with provider records¹⁶

The DHT is an integral part of the IPFS content resolution system and has very interesting characteristics. It does not need any central coordination to dispatch information between the nodes, it is fault-tolerant and scales nicely the more nodes there are in the network¹⁷.

In IPFS, we search for PeerIDs holding a CID because the purpose of DHT is not to provide the content directly. The value retrieved from the DHT is the PeerID. The retrieved PeerID is then contacted via Bitswap to download the CID. This information is provided by the node near the CID. To determine the node near the CID, Kademlia uses the XOR Metric between two identifiers, such as a CID or a PeerID. The distance is defined as the bitwise exclusive or interpreted as an integer. This notion could be summarized as the number of bits that defer between two identifiers. The closeness is determined by the number of bits similar[16]. Recently, IPFS has developed multiple ways to represent CIDs¹⁸ and to make the XOR operation, we need the 2 identifiers that we compare in the same address

¹⁶Illustration inspired by Henningsen et al. [12]

¹⁷<https://docs.ipfs.io/concepts/dht/#kademlia>

¹⁸<https://docs.ipfs.io/concepts/content-addressing/#identifier-formats>

space. To handle this, IPFS hashes both identifiers using SHA-256, making both identifiers in the same address space. For example, when comparing a CID with a PeerID, both are hashed using SHA-256 and then we estimate the distance between a PeerID to the CID using the result from the XOR distance.¹⁹

We said previously that a node knows at least one node in each of its subtrees. It uses the principles of k-buckets to store information about nodes. Each k-buckets is a list where all lists contain a range of nodes between 2^i and 2^{i+1} from itself. Each list contains k elements at most, we can have a bucket without a node. In IPFS, $k = 20$. Each element from the list provides information about the IP address, Port, and ID. Each list is sorted given the most recent node that responds at the tail of the list. If a peer A discovers a node, we map it to the bucket corresponding to its distance from A. However, if the bucket is full, we cannot add it directly. Kademlia pings the last element of the bucket, the last element is the oldest element without response. If A gets a response, it does not add the new node to the bucket. Otherwise, it removes this last unresponsive node and adds the new node[16].

2.3 Network with content resolution

In this section, we describe in detail the different interactions and processes realized to get a file through the Bitswap and DHT systems of IPFS. We also introduce the few steps realized to add a file to the system. It allows having an overview of the whole interaction.

Every node has its own set of Bitswap nodes. If it finds the file through its direct peers, it will provide it. Otherwise, the node passes the request to the DHT system. However, the DHT will never provide the file. It provides the ID of the node in possession of the file to the Bitswap system. The DHT is like an insurance for the system to retrieve any CID due to the limitation of Bitswap. Bitswap repeats the process of querying the node to the provider. Once a file is downloaded from a provider, the node requesting also becomes a provider for the file.

The figure 2.8 is a representation of Bitswap and DHT that allows an understanding of the similarity and differences between the two. As described, Bitswap owns a set of nodes with which it can get information directly. DHT helps to find nodes that provide the requested data when it is not already connected to a provider through Bitswap. This service is constantly updated with new files. It is an essential part of IPFS, it maps each CID to a provider.

When we plan to store a file on IPFS, we have to download the client of IPFS²⁰.

¹⁹<https://docs.ipfs.io/concepts/dht/>

²⁰<https://docs.ipfs.io/install/ipfs-desktop>

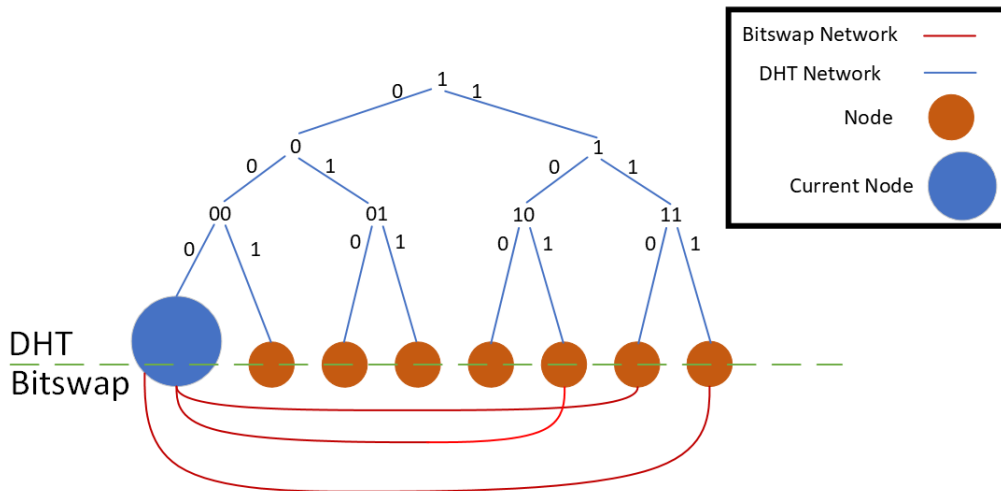


Figure 2.8: Network of IPFS

With it, we can create our own node with the help of the command-line interface (CLI). Any file that we will add to the system will follow different steps:

1. **UnixFS:** Each block of the file gets its own CID and may be linked to the next block;
2. **Broadcast:** The PeerID provides the information to its Bitswap's neighbors about the new CID and the closest node to the CID gets the information about the Peer ID holding the block;
3. **Provide:** The PeerID becomes a provider of the CID for the system. It will answer any incoming requests.

To search for content hosted on IPFS, we get it through the command-line interface. Bitswap and DHT may be solicited by the query. There are major steps followed by IPFS when there is a query to retrieve content. These following steps are represented in figure 2.9:

- Locally
 - **Step 1:** Look if we already have the file;
 - **Step 2:** Get the list of items to query by UnixFS;
 - **Step 3:** If required, group the different blocks of the file together to piece back up the original file;
- Bitswap protocol is then used if the file was not already present in the local storage.

- DHT, or Distributed hash table as it is called, will receive a CID as a key and return the corresponding PeerID of one or more providers.
 - **Step 7:** Get the closest peers to the CID in its bucket ;
 - **Step 8:** Ask the closest peers provided;
 - **Step 9:** Find peers holding the data in the network through the request;
 - **Step 10:** Transfer the Peer ID of the provider found to the Bitswap service. Bitswap will handle the connection to the peer and retrieve the file.

The lookup is stopped only if it finds the provider. Otherwise, it will recursively ask for closest peers to the CID.

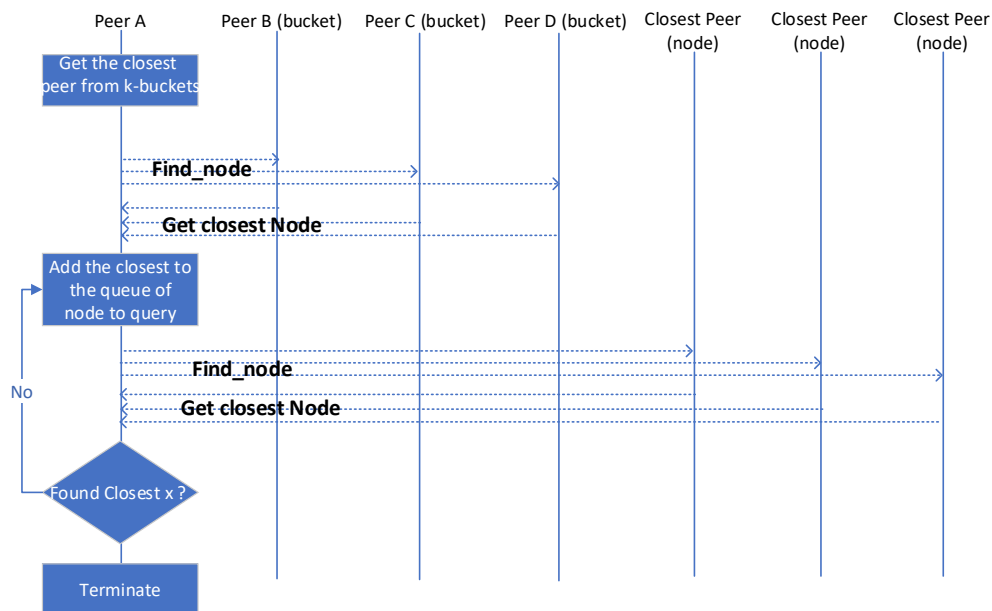


Figure 2.10: Lookup algorithm of DHT

To illustrate the lookup algorithm, we make the diagram 2.10. The DHT search for the k closest peers to the CID x . The process is done in 6 phases:

1. Load the k closest peers to x from the k -buckets (routing table) of our node into a query-queue.
2. We ask to 3 of the k closest peers if they know a provider of x

- If they know a provider, we return the PeerID of the provider.
 - If they do not know a provider, return new k closest nodes and add them to the query-queue
3. Ask the next 3 peers in the query-queue.
 4. Terminate when the 3 closest peers to x have successfully responded without error or timeouts.
 5. Finally, return the k closest peers that do not fail ²¹.

Since DHT cannot download the file, we retrieve the information from the k closest peers about providers of the given CID x . Finally we connect to the provider(s) using Bitswap and verify if they really host the file using a **HAVE** message. If the response is positive, we download the blocks. Otherwise, we continue searching using the DHT.

2.4 FileCoin

Multiple projects are using IPFS technology, a major one is FileCoin. FileCoin gives an incentive for users to store third-party content on their nodes. FileCoin is the new project of Juan Benet, the man behind the initial implementation of IPFS. Filecoin has multiple objectives that enhance the utilization of IPFS. In our analysis of IPFS, we inform about the new technology brought by Filecoin to ensure that there is no fix provided to our domain of analysis by Filecoin as well as complete our knowledge about the ecosystem built around IPFS.

In IPFS, the data in the network exists as long as a user stores it. It implies that there are no guarantees about the life span of a CID. With the use of monetization for data-storing, FileCoin allows for a longer-term persistence to safely store a huge amount of data, it also allows for quicker retrieval of data since data will be stored on more nodes. Filecoin enables a way of earning money in exchange to store a file for others. For this behavior, Filecoin must provide other features than the ones IPFS already does. One of those additional features is Proof of Work.

The proof of work used by Filecoin is called Proof of SpaceTime (PoSt). PoSt is an algorithm that is running in a loop to prove the storage of the same file for an amount of time. To summarize the idea behind this algorithm, it randomly chooses blocks of a root CID through Merkle Dag and verifies that the provider has specific

²¹<https://docs.ipfs.io/concepts/dht/#routing-tables>

bytes in its storage. If the byte is there, the proof is done. Otherwise, it fails and the miner is penalized. In addition to that, it uses the Proof of Replication to prove the storage of a unique copy of data²².

The client uses PoSt to believe in a provider. To ensure the correctness and transparency of the system, it works like a blockchain where everyone can see the interaction of every provider in the network. In this way, any observer of the system can see the availability or the behavior of the host. Then, any client of a given host can inspect the work done by a provider²³.

2.5 Problem statement

To conclude this chapter, we resume the most relevant knowledge for our security analysis. IPFS is content-centric and a node must first discover which nodes provide the content that it is looking for, before downloading the data.

The process of finding the provider is realized either through DHT or Bitswap. In our research, we focus on DHT because when a file cannot be found via Bitswap, it uses the DHT. The range of lookup for Bitswap is limited to the direct peers of a node, while the DHT has no limit. It allows finding a CID anywhere in the network, making it an essential component of IPFS.

Since the content can be provided by multiple locations inside the network, we will not block a specific location but content. This attack is a Denial of Service because we block the access when someone tries to retrieve the content. We can make this attack by disrupting the resolution or the download. We decide to inspect the resolution with the use of Kademlia. By inspecting the process of DHT and its scientific documentation, we discover vulnerabilities in its lookup algorithm. We make an attack on this part by hiding content for the rest of the network.

If this attack is not fixed, one attacker can reproduce it for malicious purposes, such as political usage where one entity blocks access to the website of an opponent. IPFS is rapidly growing and is a main actor of Web3. To stay a pertinent actor in the Web3 space, IPFS needs to provide robust security for all content storing and guarantee the availability of content to all users of its network.

Since IPFS works with two complementary protocols to retrieve the content. We must consider the different consequences of the interaction between Bitswap

²²<https://proto.school/verifying-storage-on-filecoin/04>

²³<https://docs.filecoin.io/about-filecoin/why-filecoin/>

and DHT in our attack. Therefore, we make a deep analysis of the resolution by extracting information through experimentation. This lead-us to experiment with our attack on the IPFS network. This attack seems pretty effective in hiding content.

As we want to improve the security of IPFS, we document all stages we do to realize this attack, as well as the consequences of each step. In addition, we prepare a series of counter-measure to help Protocol Labs. This counter-measure is dedicated to reducing the feasibility of the attack realized in this document. All results presented in this these are shared with Protocol Labs to help further enhance the security of IPFS and its users.

Chapter 3

Network Attacks

Previous chapters have introduced how the main components for content resolution are working. From this gathered knowledge, we identify the critical components of the content resolution process and how each of them can be exploited for executing a Denial of Service attack. Furthermore, we introduce an attack on the content resolution system. This attack can hide the content once added to the IPFS network.

We examine the different attack possibilities in order to improve the security of IPFS. The most important component of IPFS is its content resolution system and it is important to make it as robust as possible to various kinds of attacks.

Moreover, we discuss the different hypotheses in order to deploy the attack and evaluate the severity of this attack. The results of the different attacks are explained in the next chapter.

3.1 State of the art

In this section, we discuss the scientific documentation about the kind of attacks that we realize. We explain the interest in such information.

3.1.1 Sybil attack

Large-scale peer-to-peer environments, such as IPFS, consists of a large network of *nodes* or *peers*. These nodes share resources, such as disk storage and CPU power. Each peer in this network will help to serve requests from other peers and other peers will help it. A popular application for peer-to-peer systems is data storage, this type of system adds redundancy to data. Different nodes can host the same data and thus avoid centralization of information and single points of failure. In peer-to-peer systems, we differentiate network participants (entities)

from network nodes (identities). We make this distinction on the assumption that an entity may possess multiple identities [9].

Originally, we allow one user to host multiple nodes in a distributed network, in order to add redundancy. This property can be used maliciously if it is possible for one participant to control a large number of nodes. It makes it possible for one entity to control a substantial fraction of the network and thus undermine the redundancy property.

We will call the forged identities *sybils*.

A Sybil attack is the term given to the action of forging multiple identities corresponding in reality to a single entity.

3.1.2 Eclipse attack

Distributed systems have good properties when their algorithms are executed correctly [24], but sometimes not every scenario is taken into account.

When someone controls multiple nodes inside a network, they may use the algorithms to their advantage.

An *Eclipse attack* [23] consists of filling the routing table of an honest node with a coalition of malicious nodes. When successful, it allows the attacker to manage the incoming and outgoing communication of the honest node. The node is thus *eclipsed*, a visualisation of this concept is shown in figure 3.1.

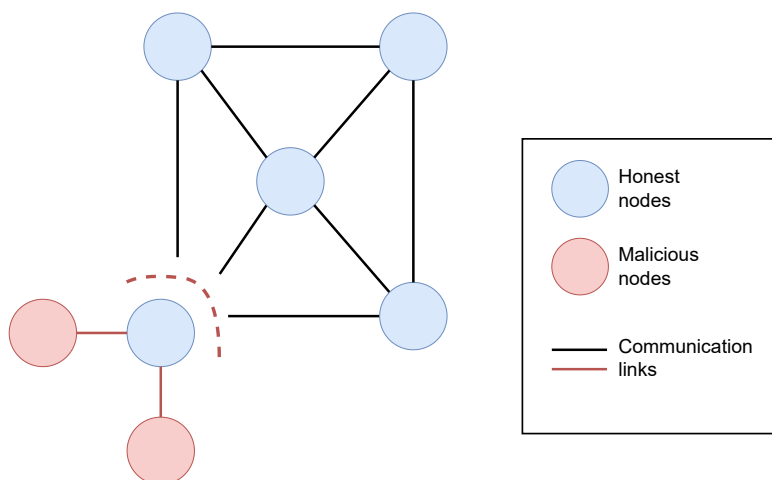


Figure 3.1: Eclipse attack

Once an attacker controls incoming and outgoing messages, it could manipulate its behavior by feeding the node with false information.

The Eclipse attack is closely related to the Sybil attack but even when a system has a defense mechanism against the latter, e.g. by exclusively allowing certified nodes on the network, Eclipse attacks are possible.

Kademlia has a mechanism by which the nodes periodically look for new neighbors to update its k-bucket every time it communicates with another node [16]. Malicious nodes can exploit this by advertising only other malicious peers. This is why we insist in subsection 2.2.1 that one user should be careful when manipulating the Bootstrap peers on IPFS and that they are a central component when a node is launched.

Ways to prevent eclipse attacks are to make it harder to create Sybil identities, as well as to increase the number of connections by node.

3.2 Proposition of Attacks

As our objective is to find a security failure in the content resolution of IPFS, we focus our security analysis on how the different protocols work. Our security analysis focuses on finding a failure in the DHT protocol. We choose DHT over Bitswap because for every CID not found using Bitswap, the content resolution system asks to the DHT which node is a provider. DHT is thus an essential part of locating content and if a vulnerability is found, a majority of requests will be affected.

We must keep in mind that we are not attacking the whole IPFS system due to the two complementary networking systems used.

3.2.1 Sybils in IPFS

Making a Sybil attack on IPFS would be equivalent to creating and controlling multiple nodes on the network, even though all these nodes are originating from single or multiple computers and someone, usually malicious, may control them in order to make other types of attacks inside the network.

The risk of Sybil attacks is inherent to completely decentralized systems, due to the fact that no authority can control the number of nodes an attacker may generate. Douceur [9] proves that this attack can only be hampered. One way to impede this attack is proposed by Douceur [9] as well as Benet [3] in the original IPFS draft. In a centralized system, one solution is to make someone pay when creating a new identity. To adapt this concept to a decentralized network, we can make someone pay using its system resources. The identity of a node must be validated using a resource-demanding challenge, then entities on the network can pool the identities that all have a separate validation [9][3].

In IPFS, no previously discussed solutions have been implemented in order to impede the creation of Sybil identities. In other decentralized systems, a solution to impede the effect of Sybil attacks is to make the Sybil less important in the network. To create an identity on IPFS, it just requires initializing a node using the IPFS CLI. Another way is to directly insert a PeerID and a private key into the configuration file of the node. The latter allows any attacker to define a specific PeerID.

One of many dangerous operations possible, when an entity has possession of *sybils*, is an eclipse attack.

3.2.2 Eclipse attack in IPFS

One possible approach to eclipse a node is to fill all its connections with other nodes. In this way, the attacker completely manages the incoming and outgoing messages viewed by the target. This approach implies knowing exactly how Bitswap and DHT manage their routing table.

Bitswap and the DHT routing table are different. The DHT will use the bucket system from Kademlia for routing purposes and Bitswap will use a more classical routing table consisting of a single list of peers to directly communicate. The classical routing table does not take into account the distance concept of Kademlia. Of course, some peers will be present in both routing tables. This kind of eclipse attack on IPFS has been covered by Prünster et al. [19].

However, nodes are not the only thing we can eclipse in IPFS. In DHT, the content is placed in the same address space as the PeerID. This property makes it possible to eclipse the content from the other peers in the network if we control the access to the CID. The access control is realized by the closest nodes of the CID. It means that if an attacker is able to have its nodes close to a given CID, he can control all accesses to this CID. For the attacker, it has the benefit of avoiding filling the routing table of the node because a CID has no routing table. Our attack is based on this Denial-of-Service in the process of the content resolution by eclipsing the access to the content in DHT.

3.2.3 DHT attack

DHT is the second network component used in IPFS, but it has the specificity to always find the content if it is in the system. The advantage of attacking the DHT is to cancel the backup system of IPFS in case Bitswap fails in its content finding process.

The objective of DHT is to find the PeerID (provider) possessing the CID. To do so, we have to ask the closest peers to the CID and not from the provider. As

explained in section 2.2.2, the content and the nodes are mapped together in the same address space in the DHT. As it is in the same address space, we can compare the distance between CID and PeerID. It means that the provider of the CID may not be close to the location of the CID in the DHT and the closest peer from the CID, may also not be close to the provider. The lookup algorithm is in charge to find the link between the provider and CID.

As an example, it means that a node A searching for the CID C will ask the closest peer from C in the DHT to deliver the PeerID of the provider of the content. The closest nodes will provide the information to A . A will be able to connect to the provider in order to download the content with Bitswap.

One weakness of the DHT is the lookup algorithm which consists to search the k closest neighbors of a CID in our case. In fact, the node making the request has to trust every node to be sure to obtain the PeerID of the node having the CID. To be fault-tolerant, DHT asks multiple nodes to get the closest node from the CID in order to get multiple time the information about the PeerID. But the algorithm does not consider that multiple nodes can give the same faulty result to disturb the system. This fact is the basis of our attacks.

In order to hide the CID from the network, an attacker can exploit messages used by the DHT. The types of messages essential to retrieve the location of a CID are:

- the GET_PROVIDER message.
- the ADD_PROVIDER message.

Get_Providers

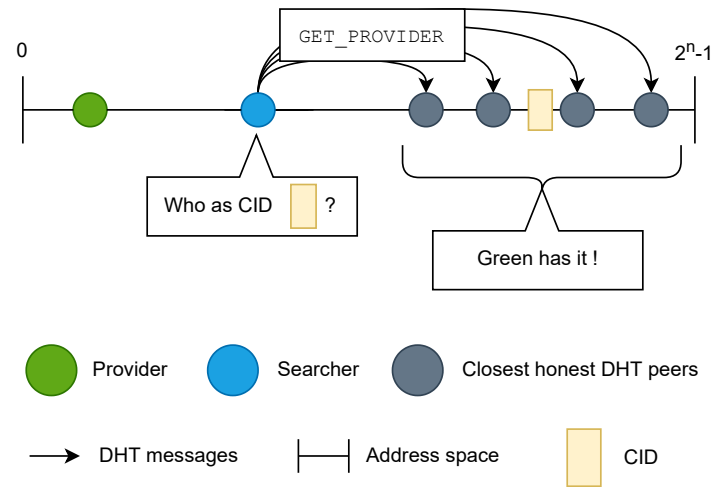


Figure 3.2: Process of looking for the provider of a CID on the DHT

The figure 3.2 represents the concepts behind the lookup for getting the PeerID having a given CID. We call the node making the request, a *searcher*. Once the process is initiated for retrieving a file in the network, the searcher emits a `GET_PROVIDER` message. Then, this message is sent to the closest nodes from the CID present in its bucket list. In IPFS this list will contain twenty nodes. Once a node receives a `GET_PROVIDER` message it checks in its memory if it knows any providers for the given CID:

1. If it does know a provider, it answers to the searcher with the corresponding PeerID.
2. If it does not know the provider, it answers to the searcher with nodes from its bucket list that will be closer to the CID.

This process operates recursively until a provider is found, as described in 2.2.2.

On figure 3.3, we illustrate how to disturb this process. The attacker must place 20 sybils around the CID. The 20 sybils is the number of closest nodes considered by IPFS and if there are no closer nodes, the process will not be able to recursively continue. Then, the 20 sybils must be implemented in order to block all accesses to the CID. It must not inform about the PeerID having the CID when they get the query.

In this way, the CID will never be provided to nodes making the query.

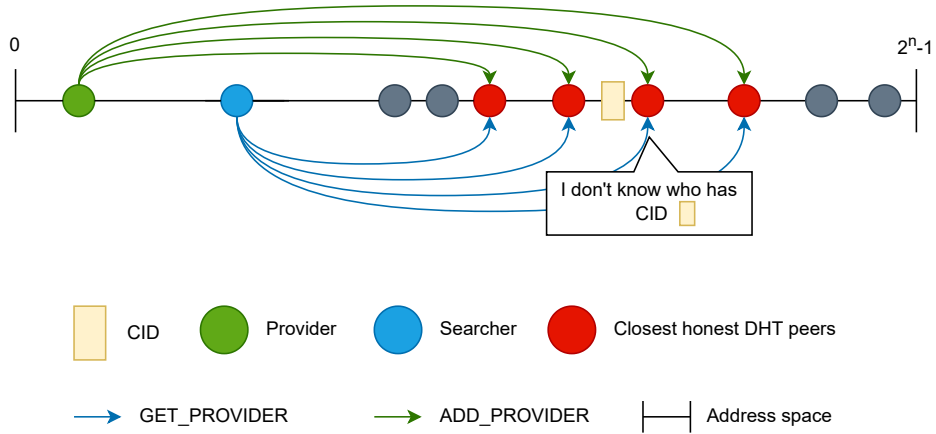


Figure 3.3: Attacking the DHT to eclipse a given CID

Add_Providers

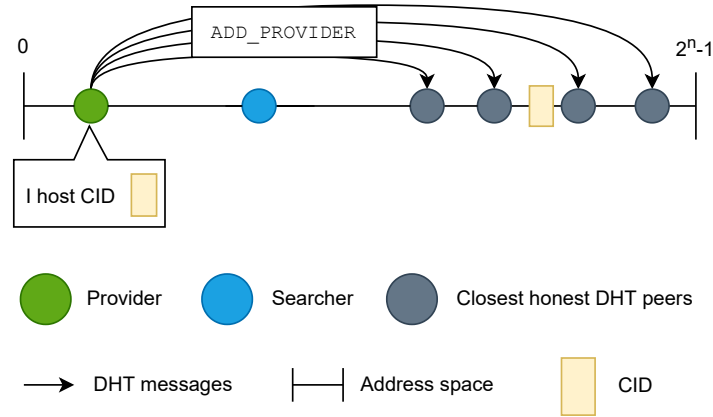


Figure 3.4: Broadcasting a CID on the DHT

In order to know which nodes are providers for a given CID, they need to advertise it to the network. This is done using the `ADD_PROVIDER` message.

The advertisement process is illustrated on figure 3.4. This process is initialized by the provider of a CID whenever a user decides to add a file or download a file. This `ADD_PROVIDER` message is advertised to the k closest peers to the CID, in the current implementation of IPFS it sends the message to the 20 closest peers. Once a node receives the `ADD_PROVIDER` message, it will map the given CID to its provider. This will allow the node to respond to every `GET_PROVIDER`.

The `ADD_PROVIDER` message introduces an alternative way of hiding a CID, that is easier to implement.

On figure 3.3, we illustrate how to disturb this process. We first place 20 sybils close to the CID so that all `ADD_PROVIDER` requests go through them. Then, in the opposite of responding to the request from a searcher, the sybils drop the message `ADD_PROVIDER` advertising the CID.

When a future node requests the CID, there will be no answer because the Sybils have not integrated the information about which node possess the CID.

This attack is easier for the attacker because the propagation of the information is more effectively controlled and it has to change a small piece of the implementation.

Challenge

In both attacks, the challenge for the attacker is to place with certitude its sybils at the right place to eclipse effectively the CID.

Finally, we only require the CID to place our sybils and to make a command `ipfs DHT query [CID]` to obtain the current node closest to the PeerID. With the help of our different tools, we use the XOR distance between the current closest node and the CID to obtain their distance, as well as the distance between the generated PeerID and the CID. Then, we have to compare the two distances to see if it is effectively closer than the current closest node. If it is the case, we have correctly generated 20 PeerIDs and private keys that can instantly be used to run our sybils.

Consequence

The consequence of this attack could be relatively important. Until now, we consider only the possibility to hide a single CID. But, we could ignore all the CID closest to us and it makes all a small region of the DHT blind from the other part of the network given a DHT point of view. Indeed, the Bitswap network could always get the CID from the neighbor but only 0.45% of the network can access the CID if there is only one provider.

Moreover, our sybils in this implementation look like a regular node for the rest of the system because it acts like a regular one when it responds to other queries. With the basic idea that this is difficult to detect the misbehavior of a node from a crash in a distributed system, if our node appears like a regular node most of the time, the attack becomes difficult to detect and be canceled in DHT.

Chapter 4

Evaluation

In the previous chapters, we introduce the principal technologies used in IPFS in order to be able to do a security analysis. We focus on the content resolution since this system is essential for IPFS. IPFS is a data-centric system where we have to first locate where content is stored before being able to download it. A compromise in this process would affect every user on the network. After we identify vulnerabilities, we propose an attack and analyze its feasibility in the live IPFS network as well as the impact such an attack can have.

The goal of this security analysis is to identify potential vulnerabilities in the content resolution system and to notify Protocol Labs.

In this chapter, we are describing our experimentation in order to realize our attack. The attack exploits vulnerabilities in the DHT content resolution process and is capable of hiding a given CID from the rest of the network.

In the first part, we will introduce the ratio between the number of requests answered by DHT or Bitswap in IPFS. It should give more information about the potential severity of our attack.

Then, we estimate the response time of a query handled by both systems. We want to get this information to be aware of the time taken to get a response from IPFS. In the same idea, we make an experiment to have a better understanding of the evolution of the number of peers connected through Bitswap to a node. Both information will be taken into account to evaluate an optimistic scenario for our attack.

Afterward, we introduce additional experiments that allow us to set up our attack. These experiments allow us to verify our knowledge about which nodes are contacted when we add a CID to IPFS and where the messages between nodes are managed in the implementation of DHT.

Finally we introduce the implementation of our Denial-of-Service (DoS) attack

that takes place on the DHT. To achieve this attack hiding a CID from the rest of the network, we discuss how we are able to generate 20 Sybils. And more specifically to our DoS attack, how to place these 20 Sybils close to a CID. This placement is essential to our attack and allows the interception of `GET_PROVIDER` and `ADD_PROVIDER` messages. We describe the tools and how the experimentation has been executed. Furthermore, we realize the last experiment to establish the severity of the attack by making a lot of requests from different nodes in order to retrieve the CID eclipsed. We execute this attack, without disturbing other users of the IPFS network.

4.1 Bitswap - DHT ratio

4.1.1 Hypothesis

In this experiment, we want to have a better understanding of the implication of Bitswap and DHT in the content resolution system. We presume that there is one used more than the other. Calculating this ratio will give us more information about the probable severity of an attack in each protocol. Otherwise, the responsibility for security is split between both.

We have 2 hypotheses given the information gathered from IPFS:

- More CIDs will be found using the DHT because IPFS is a huge network and peers are only connected to relatively few peers. Finding new peers via the DHT is inevitable and will account for a large part of query answers.
- The IPFS team made an announcement in the IPFS documentation explaining the implementation of IPFS. It states that the DHT should account for 50% of the answered queried and Bitswap the other 50%. Of course, this is just an average¹.

Based on the IPFS statement² we believe the estimation of IPFS to be wishful thinking. We do not have confidence in the information provided since they do not provide the details in their DHT-Bitswap ratio estimations. As the two hypotheses are different, we will try to confront them in our results.

4.1.2 Measurement of the ratio between DHT/Bitswap

The principle is to define correctly when a CID is found through Bitswap or DHT. Unfortunately, this information is not directly available in the IPFS client

¹<https://docs.ipfs.io/concepts/bitswap/>

²<https://docs.ipfs.io/concepts/bitswap/>

and we must determine it for ourselves. In order to achieve this task, we have to understand exactly the cooperation behind both. As explained, it is the Bitswap component that will decide when the DHT will be needed to retrieve a CID. It means that we always use Bitswap when we retrieve content.

Our objective is to determine when the use of DHT is required. Therefore, we analyze the source code of Bitswap to find the different aspects of the content resolution until we find the part where it communicates with the DHT. It brings us to a function call that delivers us with the PeerID providing the CID we are looking for.

Thanks to this discovery, we are able to determine where DHT is called in the Bitswap process. This execution implies that no neighbor has returned the CID. In order to make the experimentation, we realize a small script to collect the information with two boolean variables in order to determine if it has been found via DHT or if it has been directly found via Bitswap, meaning that a provider of the CID was already present in its direct peers.

A problem we encountered when creating our script is about the DHT lookup. It never stops until it finds an answer. We change the implementation to stop it after a given number of iterations. Indeed, if a CID is no longer available in IPFS, the system will not be noticed and it will continue to search even if there is no more PeerID providing it. This scenario could happen quite often. To manage this scenario, we introduce the variable `MAX_DHT_RETRY` in our implementation of IPFS to control the number of iterations done by the lookup algorithm.

In our script, the process of getting a CID is summarized in two variables initialized to false by default. In our modified client, we introduce two lines of code in the implementation, one in the code concerning Bitswap requests and the other present in the DHT part. We keep our changes local. When we get a file through IPFS, the process of finding the provider will begin and whenever we go through Bitswap or DHT we put their respective boolean variables to true.

This technique allows us to determine the different steps made to retrieve a given CID. Moreover, it provides us with information about failed queries. We represent in table 4.1 to explain the different outputs handled by our script and to which scenario it corresponds.

DHT	Bitswap	Meaning
False	False	Query is not completed
False	True	Query is completed using Bitswap (via direct peer)
True	True	Query is completed using the DHT (via a peer on the DHT)

Table 4.1: Truth table we use in order to apprehend the background processes

To realize multiple requests, we use a list of 40 000 CIDs that were used in another IPFS paper studying the structure of the overlay network in IPFS [12].

4.1.3 Result

Found via a direct peer	Found via DHT	Never Found
511	331	5569

Table 4.2: Results from our python script after 6411 CID **GET** queries -
MAX_DHT_RETRY = 1

The first part of the experimentation has been realized with the number of iterations of the lookup set to 1. The time taken to make a request with the iteration set to 1 is around one minute. This experiment has been realized for 4 days and 12 hours. We introduce our result of this experimentation in table 4.2. The result is interesting because we observe a large number of CIDs "Never Found" in the system. Even though this CID list has been used for the study from Henningsen et al. [11]. This number of CID "Never Found" could be explained by our number of iterations.

Otherwise, we get interesting results from our two systems. We can see that there are 511 CID found through Bitswap and 331 through DHT. If we do not consider the CID not found, we have found 822 CID in the system. The ratio of CID found by DHT is 37,83% and the ratio of CID found by Bitswap is 62,16%.

Found via a direct peer	Found via DHT	Never Found
1050	104	5076

Table 4.3: Results from our python script after 6229 CID **GET** queries -
MAX_DHT_RETRY = 4

With the second part of the experimentation, we set the maximum number of DHT iteration to 4. We expect to retrieve more results with DHT than in the first part.

In the table 4.3, we can see that the result is pretty much the same as the first experiment. We have even more results through Bitswap because our node making the request has discovered more interesting neighbors compared to the first part. Given our hypothesis, we discover that there is not a ratio of 1/2 as claimed by IPFS and there is no prevalence of DHT. Both hypotheses were wrong. It means that there is more chance to find a node through Bitswap than DHT even if we increase the number of iterations in DHT.

4.2 Analyzing Response Time

4.2.1 Hypothesis

This experiment must bring concrete information about the mean response time of a get request in IPFS. We would like to know this information because searching for a provider using the DHT can take a long time to process. This is due to the large number of iterations allowing a searcher to come closer to the given CID.

With this information, we would like to determine a limit of time where we assume that it becomes impossible to retrieve the content through DHT.

Moreover, we are also looking at the difference in response time for a request between DHT and Bitswap.

Our first hypothesis is that the process takes less than 2 minutes in general to retrieve content in both systems. Our second hypothesis is that Bitswap takes significantly less time to retrieve the content than DHT.

4.2.2 Measurement of response time

To realize this experimentation, we extend the script from the previous experimentation 4.1. We are only collecting the time when we execute the request to retrieve content and which system gives the response.

We are taking the advantage of the previous architecture of our script to make the difference between the request realized with DHT and Bitswap. Due to the architecture of IPFS, we are not able to distinguish clearly the time taken by DHT without considering Bitswap. For this reason, the measure of time does not take the time separately between DHT and Bitswap but the global time to resolve the request.

4.2.3 Results

In the table 4.4, we represent the results retrieved on 306 `GET` requests. The first hypothesis is about the General Mean Time and we can see that the result is

clearly below one minute. Moreover, the time taken by Bitswap is clearly inferior compared to the result regarding DHT. It validates our second hypothesis about the different mean times between both systems.

Mean Time via Bitswap	Mean Time via DHT	General Mean Time
0,4s	28,71s	2,81s

Table 4.4: Results from our python script from 306 CID **GET** queries

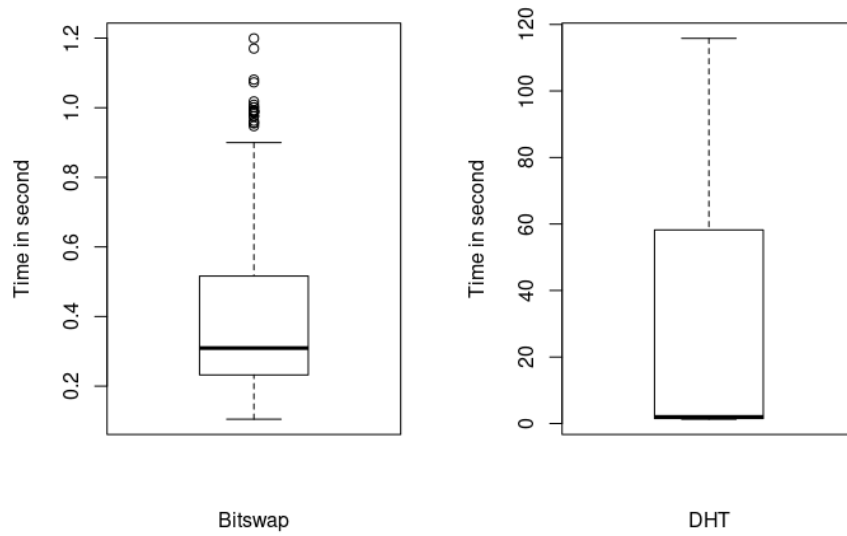


Figure 4.1: Boxplot of Bitswap and DHT time

We create two boxplots to visualize the difference in terms of time between both systems. We can see that the maximum number of times taken in DHT is 115s. However, the third quartile is lower than 60 seconds, which means that there is exactly 75% of the time data taken below 54.2s. The median is equal to 1.997. These statistics help us determine that if we do not receive a response to a **GET** request in under 115s, there is a high probability that this request has failed. Bitswap responds most of the time below one second. The maximum value from our sample is 1.1994s. Between the two systems, there is a huge difference in time.

4.3 Analyzing Bitswap connections

4.3.1 Hypothesis

In IPFS, a request for getting a file can be answered in 2 ways. One way is by querying all peers from the Bitswap routing table of a node. If one of the Bitswap peers is a provider for the given CID, it answers positively to the query and sends the corresponding content. The success of finding a CID through Bitswap is thus dependent on its number of connected nodes. The more one node is connected to others, the bigger the probability to find a CID within the Bitswap protocol and not rely on the DHT.

Due to memory and processing power limitations, one node must limit its number of Bitswap connections. This limit is defined by two parameters in the IPFS configuration file, **LowWater** and **HighWater**[19]. The number of Bitswap active connections will range between these two values. Once the **LowWater** threshold is attained, the node increases its number of connections until the **HighWater** threshold is reached. Once reached, the node defines which of its peers are the least useful and prunes them. The default values for **LowWater** and **HighWater** are 600 and 900 respectively³.

This experimentation has to determine the amount of time required for Bitswap to reach the **LowWater** threshold. This information is required for our scenario attack explained later in this chapter. In our attack, we create a victim node whose role is to get a given CID. We need to create our own victim node so that our attack does not target an honest peer. This victim node must be representative of a common user of IPFS. Indeed, we would like to create a scenario where the Bitswap routing table of our victim node has the time to connect to a number of neighbors representative of a genuine node. Having a number of peers between the 600 and 900 makes our victim node similar to other nodes in the network using the default threshold values for **LowWater** and **HighWater** and thus allowing us to analyze the severity of the attack on an honest node. In addition, we would like to discover more about the evolution of the peers and especially the difference between a passive node and an active node. Here, the active node is defined as a node making requests to many different CIDs. The passive node will not make any user requests, it only processes messages it will receive from other nodes in the network.

We suppose that the process of reaching 600 connections proceeds in less than 10 minutes.

³<https://github.com/ipfs/go-ipfs/blob/master/docs/config.md#basic-connection-manager>

4.3.2 Measurement of Bitswap Connection

To realize the measurement, we use the information received from the CLI of IPFS. The `ipfs stats bitswap` command gives us the information about the number of peers present in Bitswap.

To analyze the general behavior of this system, we run this measurements on 10 different machines with Ubuntu OS and IPFS installed. We use our own script to generate the information sampling in a regular one-minute interval. The script runs the command discussed and collects the information about the number of peers. We save the collected information in a CSV file with the date and hours of the collection. The 10 devices are run at the same time when we make the collection of information.

The first collection is on the passive node. The passive node is a node running but doing no active tasks for the user, it is simply relaying messages from its peers, caching files, and being a provider for others. On the opposite, the active node is run with the script of this experiment and the previous script explained in 4.1. As the previous script makes a lot of requests, it allows us to emulate the behavior of an active node.

The collection of measures is made over a period of an hour. We realize a mean value on the number of Bitswap connections taken every minute on the 10 Virtual Machines. It should give a correct representation of the mean number of neighbors through time.

4.3.3 Result

In Figure 4.2, we are providing the behavior of one passive node and one active node. The active node is making periodic get requests and the passive node makes no requests. The connections from the passive node represent well how **HighWater** and **LowWater**, defined in sec. 4.3, work in a real context. The Bitswap routing table of a node will not always hit the default **HighWater** threshold before dropping the least useful peers.

With the active node, we can see that it is more often under the **LowWater** threshold than the passive node despite the fact that it contacts more nodes.

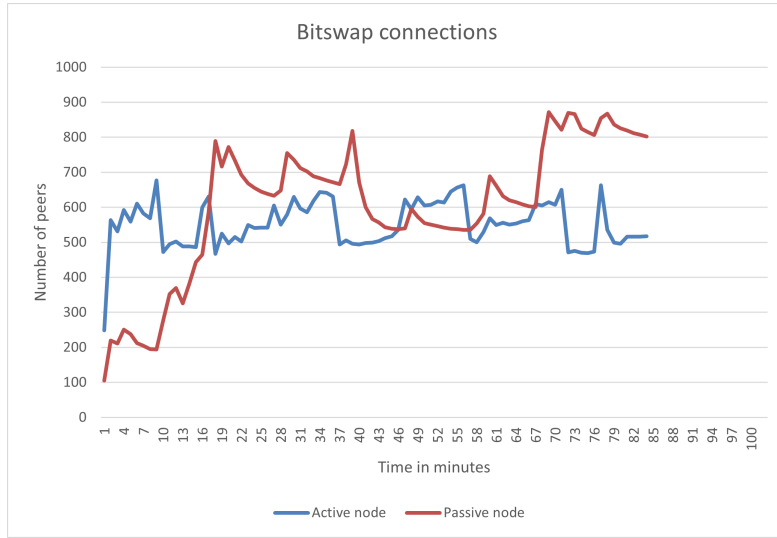


Figure 4.2: Bitswap connections through time - 84 minutes
(Default HighWater and LowWater values)

To have a more global view of the behaviors, we realize the same experience with 10 devices working in parallel. We calculate the mean of peers for each minute and we introduce the information in the Figure 4.3.

In this Figure 4.3, we clearly see that the passive node has more nodes in average than the active node. However, the active node is faster to fill its list of neighbours. It takes only three minutes to be in its range between **LowWater** and **HighWater**. The passive node takes eight minutes to be in the range.

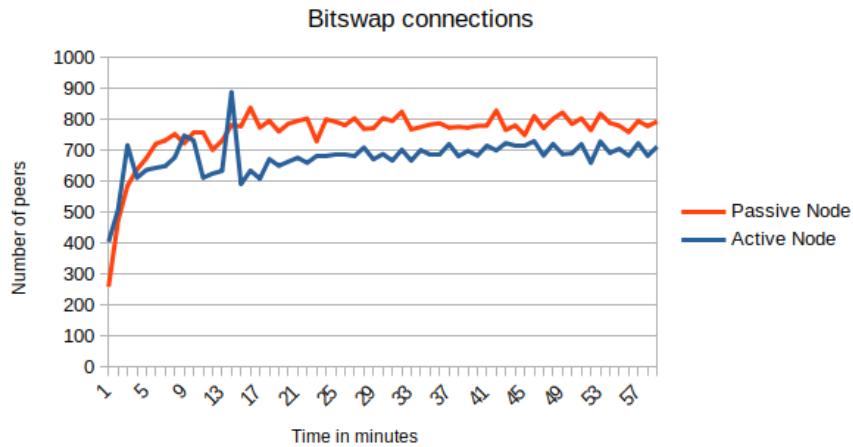


Figure 4.3: Bitswap connections through time - 60 minutes with mean value

With both information, we are able to confirm our hypothesis that a node reaches 600 neighbors in less than ten minutes whether it is an active or passive node.

4.4 DHT Add contact

4.4.1 Hypothesis

When we add content to IPFS, we suppose that we are only informing the closest nodes of the CID and no other one. Our hypothesis is that the process contacts the same node from the command `ipfs dht query <CID>`.

This command is originally used to query the closest node from a PeerID. In our experimentation, we want to be sure that this command works for a CID and that we are effectively contacting the same nodes as this command. It also means that we are contacting only 20 nodes when adding a file to the network since the `ipfs dht query <CID>` command provides only 20 nodes.

This information is primordial to realize our attack in order to know the number of Sybil nodes required to eclipse the CID and know which honest node is the current closest from the CID. We need this information to create Sybils that are closer to the CID than any honest node.

4.4.2 Measurement of the interaction in DHT

The principal challenge of this experiment is to retrieve the list of nodes contacted when we realize the `ipfs add <file>` command. To validate our hypothesis, we only have to compare the list of nodes reached and the PeerIDs present in the `ipfs dht query <CID>` command.

To identify the list of nodes contacted by our command `ipfs add <file>`, we suppose that the request is managed at the same location in the source code as the request sent by Bitswap when asking the DHT for a CID, in the file `routing.go`.

Thanks to a comment in the source code from the IPFS team, we are able to identify the function `Provide`. This function is used to announce that the node is providing a value (PeerID) for a given key (CID).

Now that the function is identified, we print the list of peers to which we broadcast the CID.

4.4.3 Result

Our hypothesis succeeds because we find the same PeerIDs in the list of peers reached by the `ipfs add <file>` and the command to retrieve the closest node

`ipfs dht query`. We do not only find some common peers, we find that both lists are identical and have a length of 20 PeerIDs.

Consequently, it means that we are able to find easily the closest nodes for a specific CID and that we contact the same peers when the node announces the CID. Moreover, we know that it contacts exactly 20 nodes to add content.

4.5 DHT Attack Development

This section is dedicated to our attack. We explain the different tools we use to make it possible and we explain in detail our Denial-of-Service attack. This Denial-of-Service attack hides a given CID from the rest of the network and makes it inaccessible. We introduce our results at the end of this section.

4.5.1 Tools

Generating closest peers

In section 3.2.3, we talk about the scenario where one attacker can intercept all DHT messages concerning one CID, not reply to these requests, and therefore making a CID non-accessible by other peers in the network. The attacker must have 20 Sybils located close to the CID to accomplish this. This will allow the Sybils to always be the last peers relaying the messages about the availability of the given CID in the DHT.

To reproduce this attack we will need to replace the 20 current closest peers with Sybil nodes. Retrieving the 20 closest peers to a CID can easily be done using the IPFS CLI command `./ipfs dht query <CID>`.

Once we are in possession of the 20 closest peers, we need to proceed in two steps to create our Sybils nodes:

1. Find which peer among the 20 is the closest.
2. Generate 20 new PeerIDs and their corresponding private key that are even nearer.

To estimate distances, we represent each key found using `./ipfs dht query <CID>` by their own XOR keyspace⁴. This will normalize all identifiers, CID or PeerID, and then we XOR the keys together in order to calculate distances.

We are using the XOR metric because that is how distances are calculated in Kademlia, as explained in sec. 2.2.2.

⁴<https://github.com/libp2p/go-libp2p-kbucket/>

Once we know which key is the closest from the CID, we need to generate 20 Sybils key-pairs that will have a PeerID closer than the current closest honest peer.

A peerID is generated from the public key of a node. We cannot generate first a peerID that will be closest and then derive the key-pair, due to cryptographic limitations. What we do is:

1. Generate a random key-pair⁵
2. Derive the PeerID from the public key of the key-pair
3. Compare the distance between this new key and the current closest honest node
 - If the generated Sybil PeerID is closer, then we keep the public and private keys in memory.
 - If the generated Sybil PeerID is not closer, we reiterate the previous steps. We generate a new random key-pair, ...

We extend a script written by Kroł and Keizer, this script is called `generateClosest2.go`.

In this script, we repeat these steps in order to create at least 20 Sybils key pairs. The amount of time taken to generate 20 keys for the deployment of our sybils is relative to the distance between the original 20 honest closest peers and the CID. While deploying our attack on a variety of CIDs, we notice that this process can take 2 minutes, with around a thousand retries per key, or hours, with around millions of retries per key. In the majority of cases, the process takes a short amount of time given that we use our personal computers. An attacker with a large number of resources could use more powerful computers and calculate the 20 keys in parallel.

4.5.2 Setting up Sybils nodes

Once we are in possession of 20 different Sybils key pairs, it is time to set them up and launch them to the IPFS network.

To instantiate a node, we need to replace the existing PeerID and private key in the `config` file of the IPFS node. Once this operation is done, the node can be run using `./ipfs daemon`. The Sybil node will directly be located close to the CID, thanks to the fact that its PeerID and the CID are very close in the XOR Keyspace.

⁵<https://github.com/libp2p/go-libp2p-core/>

The difficult part is hosting multiple nodes on the same machine. When naively launching multiple nodes on the same machine, some requirements for IPFS nodes limited us to one node:

- Each node must have a distinct port for each IPFS service: 4001 (SWARM), 5001 (API), and 8080 (GATEWAY).
- Each node must have a distinct IP.

To avoid these caveats, we have 2 solutions:

1. Hosting each node on a **Virtual Machine** (VM).
 - A VM is an abstraction of physical hardware, turning one server into multiple ones. Each VM will have its own copy of an Operating System as well as all libraries necessary for running applications.
 - Disadvantages of a VM is that it can take up multiple GB and is slow to boot. 20 VM's on a personal computer is a bit overwhelming. Fortunately, it is possible to pay for a VM hosted in the cloud⁶.
2. Hosting each node on a **container**.
 - A container⁷ is a package of the software code we need to run, in addition to operating systems libraries and dependencies required. With a container, bundling an entire operating system is no longer required.
 - The benefits of using a container are multiple. The ones we are more interested in are the fact that they are more lightweight and faster to boot. Running 20 containers on a single machine is totally feasible.

The properties of containers are very interesting and are a huge time saver when deploying 20 nodes. Nevertheless, we use the two proposed solutions when executing our attack. Even though Virtual Machines take up more disk space and are slower to boot, they have beneficial properties.

In the next section, we explain how we deploy each solution, as well as the difficulties we encounter.

Experimental setup

There are two mandatory approaches to make the experimentation with each their benefits and their drawback.

⁶<https://aws.amazon.com/>

⁷<https://www.ibm.com/cloud/learn/containerization>

Docker is our first approach because it has a lot of advantages. Indeed, we can easily duplicate our setup by creating multiple containers with the image of our malicious client. We have to be careful when deploying our container with the port because IPFS uses the same port to communicate between nodes. It means that we have to set different ports for each container in the machine hosting them.

The disadvantages are mainly the CPU cost. When using docker we can set up many nodes on the same machine, but due to the heavy usage of the CPU with IPFS we recognize often less than 20 Sybils in the `ipfs dht query <PeerID>`. Sybils appear and disappear from it after minutes. We try to share the load of the 20 containers on 4 machines but we find the same behavior.

We can conclude that containers are easily deployable, they are great for a rapid attack but since Sybils are sometimes disconnecting and reconnecting, the probability that the attack goes well is lowered.

Virtual machines

Docker containers are not perfect. The alternative is Virtual Machines, they are more costly, slower, and it is harder to automate actions such as replacing the current key pair in the IPFS `config` file. Nevertheless, using Virtual Machines alleviates the constraint about the use of custom ports for IPFS for each Sybil. Furthermore, each Virtual Machine has its own public address which makes it more genuine.

Fortunately, all this effort is worth it. After verifying the placement of our Sybils using `ipfs dht query <CID>`, they are constantly there. This was not the case when using Docker on a single machine or even by sharing the load of the 20 containers on 4 different machines. All Sybils become the closest peers and are stable. The Sybils stay there for more than 8 hours.

Using Virtual Machines is a slower process but allows for a more efficient attack thanks to the stability of the position of Sybil nodes in the DHT.

Cost

We have 2 methods to place 20 Sybils on the network. It is essential to have 20 Sybils at all time close to the CID for our attack to work. The method that guarantees the correct placement of Sybils while keeping them for a long period of time is by using 20 Virtual Machines. The cost of placing Sybils and thus executing an attack is relative to the use of one VM per Sybil. If we take as reference the price of a VM in Google Cloud with the 2 virtual processors, 4 Go of RAM, and 10 Go of memory. The price per hour averages 0,04\$ for one VM. This price may vary depending on the location.

Thus placing 20 Sybils in the IPFS network costs at least 0,80\$ per hour. In the worst case, it will cost 576,00\$ per month to run the attack using the Google Virtual Platform.

As a cheaper but less trustworthy alternative, one could use docker containers to host Sybils. A better way is to share the CPU load on 5 different VMs, which equals to a cost of 0.20\$ per hour.

4.5.3 Attack on adding the content

Hypothesis

As described in the chapter Network Security, we will make an attack on the process of adding a file into the DHT. Thanks to our different experiments, we are able to have a clear understanding of the process and the different implications.

Our hypothesis is that we are able to eclipse a given CID with 20 Sybils placed around the target CID in DHT. The requirement is to have the Sybils run when the PeerID makes the request to add the content to IPFS. We do not target an existing CID, we create a custom CID only used by us. The peer making the add request is also controlled by us, so we do not disturb other users in the network.

Modified Client

We discover in our analysis of the IPFS source code that `handler.go` manages the different DHT messages, such as `ADD_PROVIDER` which notifies one node that a peer is hosting a specific CID.

When we analyze the code of this function, we can see that there are multiple operations made. A big part checks the source of the request and if the CID is in the correct format. Once these steps are done, we add the content resolution through a function call, `dht.providerStore.AddProvider(ctx, key, peer.AddrInfoID: p)`. This function maps the CID received to the new provider in the cache.

Once a provider and its corresponding CID are stored in the cache, it will allow this node to give information to anybody about the provider of this CID. Nodes asking for this information will send a `GET_PROVIDER`.

Once the information is added to the cache, the node stops communicating through the DHT. It returns a nil value that informs that there is no message to give to the source of the `ADD_PROVIDER` message. The source node providing for a CID trusts the 20 closest nodes to relay information. We can abuse this trust by never storing the information that this node is providing for a certain CID. This is done by removing the call to the `AddProvider` function. If the information is never stored, the node will not be able to correctly answer to the `GET_PROVIDER` request.

We change the implementation locally on the IPFS client and use it to connect to the existing IPFS network accordingly to realize our attack.

Attack execution

To make the attack, we use the twenty sybils explained before, we use a file that we can use as a victim CID, we deploy a victim node that will provide the given CID and an honest node that will allow us to verify if the attack worked.

The twenty sybils will be placed as close as possible to the CID thanks to the tools described in the section 4.5.1.

Once the Sybils are located around the CID, the victim node adds the CID using `ipfs add <file>`. Our 20 Sybils will be contacted and asked to store the information that the victim node provides the given CID. Since we modified the behavior of the sybils, they will ignore the request. As seen in sec. 4.4, the only nodes that the victim will contact are our 20 Sybils.

Because our Sybils are well located, our malicious node can intercept all the `ADD_PROVIDER` requests from the victim node. Since the 20 Sybils use our modified client, we do not process the `ADD_PROVIDER` message and thus information about the victim providing a given CID is never stored.

To ensure the result of our experiment, we run the attack using different types of content created by us, such as pictures or text files. It ensures that we are not perturbing the content resolution of IPFS for other honest users. It also the uniqueness of our content to verify the success of our attack.

Based on the experiment 4.2, we consider that if a regular node gets the response in less than two minutes, the attack fails. This two minutes time frame is based on the fact that we are waiting for two DHT iterations, each of 1 minute. We will deep-dive into details on the impact of the attack in section 4.6.

On the opposite, the attack succeeds because the lookup operates indefinitely until it gets a valid answer. It means that we are not able to get the information.

Result

After executing our attack, we discover an interesting behavior. We realize for the first time the attack with twenty Virtual Machines around a CID of a picture. This attack works as expected by our hypothesis. The regular node does not get the file after 2 minutes because our malicious nodes have intercepted the `ADD_PROVIDER` messages and are not processing them.

When having 20 Sybils in Docker container shared on 4 Virtual machines, we have a higher probability of nailing the attack. Unfortunately, it still happens that an honest node processes the rendering of our attack obsolete.

When executing our attack, it is better to stick with 20 Virtual Machines, each hosting one Sybil node.

4.5.4 Attack on the Get content

Hypothesis

As discussed in the section about Networking Attacks, there are two ways to execute our attack. The first way is by abusing the `ADD_PROVIDER` message where peers store information about a provider, and the second way is to abuse the process of getting information about a provider.

When a node wants to retrieve a CID, it will ask multiple nodes for the PeerID of a provider to this CID. During this process, the node will stumble across the 20 closest nodes, which are the ones that know who the provider is. The honest node sends a `GET_PROVIDER` message to retrieve the PeerID corresponding to the CID he is looking for.

We will ignore the `GET_PROVIDER` message so no node can know who is the provider of a given CID and thus nodes in the IPFS network will never be able to retrieve the content.

Our hypothesis is that it is possible to block access to a CID even after it has been added to IPFS.

Modified Client

Just as in the previous attack on the `ADD_PROVIDER` message, we have to modify the code of the DHT in the `handler.go` file. We notice the process of handling the `GET_PROVIDER` DHT message in this file.

In the opposite to the previous change, we have to comment and change more lines. In this implementation, we have to comment the function `dht.providerStore.GetProviders(ctx, key)` that retrieves from the cache the known provider of the CID. It allows to never give the information about which peer is hosting the CID.

To avoid errors, we assign a `nil` value when the code is supposed to return the value of the commented function.

Attack execution

In the opposite to the attack with the `ADD_PROVIDER`, we do not have requirements about the execution. This attack can take place after the victim has added the CID to IPFS with the query `ADD_PROVIDER`. Indeed, We can execute this attack at

any time. For this reason, we execute our attack after our victim announces the CID.

Similar to the attack on adding content, we place 20 Sybils close to the CID. There is another node that plays the role of the victim, hosting a CID we created, and another node that plays the role of the regular node making GET requests.

Once all actors of the attack are created, we launch a GET request from the regular node. It will request the CID hosted on the victim node. Finally, we check if the GET request is fulfilled and the CID downloaded.

Result

The regular node is able to download the CID even with twenty Sybils around the CID. Our hypothesis about the fact that we can eclipse a CID after it has been added with the process of GET_PROVIDER was wrong.

The failure of this hypothesis is due to the previous closest nodes that have previously received the information about the CID provider. These peers have the information and provide it when they get the request to retrieve the content in DHT. This is possible even when these nodes are no longer the closest ones because the DHT works by iteration. A node getting a CID will send a GET_PROVIDER message to nodes that are close to the CID, at the first DHT iteration it will not always be the 20 closest one.

Thus if any node that knows the provider of a CID, in our case the previous 20 closest ones, they will correctly answer the GET_PROVIDER message. If the nodes do not know who is the provider, they will provide 20 closer nodes. Our Sybils will not be able to affect the availability of information if the information is held by other honest nodes.

On the basis of this observation, we recreate this attack with the same timing as the attack on ADD_PROVIDER. On the attack targeting the ADD_PROVIDER message, we place our 20 Sybil nodes in the address space of the CID before the CID is broadcasted by its provider. We realize the same scenario with the modification on functions concerning GET_PROVIDER explained in this section. This time we place our 20 sybils before the CID is added to the network. Thanks to this timing only our nodes will possess the information about the provider of the targeted CID, except that we will never relay this information when we receive a GET_PROVIDER message. This experiment worked as expected.

This last experiment demonstrates that we are not able to block access to a CID after that a provider of this CID is announced to honest nodes.

4.6 Attack Measurement

4.6.1 Hypothesis

This last experiment has to measure the severity of our attack. Indeed, we are not in a network using only a DHT. We also have the Bitswap protocol that can manage the retrieval of content when directly connected to providers of CIDs.

We hypothesize that Bitswap will play a role in some requests realized during the experimentation of our attack.

4.6.2 Impact capture

To capture the impact of our attack, we need to have the same environment as our previous experiments. We place 20 Sybils close to a target CID we created, we deploy a regular IPFS node that will try to get the CID and a victim node that will host the target CID.

We are doing the same routine from the last experiment 4.5 for measurements, except that we will perform several `GET` requests over time.

Before doing any requests, the regular node will wait for 8 minutes to reach the default `LowWater` threshold in Bitswap, equivalent to being directly connected to approximately 600 peers. In the experiment 4.3, we find that eight minutes is the average amount of time to reach 600 peers.

We will make the attack on the `GET_PROVIDER` and `ADD_PROVIDER` messages. These will take place in the same context since an attack at any moment on the `GET_PROVIDER` message is not working. For both attack configurations, we will set up our Sybils before making the command to add the content to IPFS hosted on our victim node.

We have realized a script that executes a `GET` request of the targeted CID and after a wait of two minutes, it checks if the CID has been downloaded. We see in the section 4.2 that the Mean General Response Time was 2,81s and the worst case with DHT takes 28,71s. With two minutes, we are largely above these two means. In order to see if the placement of an honest node in the network has an impact, we instantiate the node with a new random PeerID before every request. It gives more chances to retrieve the CID through Bitswap because the new PeerID could be closer to our victim node. In this way, the new identity could become a direct peer of our victim and get it through Bitswap.

To instantiate a new node, we remove the `config` file that owns the information about our PeerID in IPFS. We use the script `clean_and_get.sh` to remove all the

files stored in the cache and on the local drive. If we do not clear the cache and the downloaded CIDs, our node will not make any requests to the network since it already possesses them.

4.6.3 Result

After 7 hours of process and 42 requests realized on our laptop, all the requests fail in getting the CID with the attack on the `ADD_PROVIDER`. In other words, our attack eclipses the node for 7 hours straight.

We did the same experiment with the client modified to ignore `GET_PROVIDER` and we get the same result. Our regular node has not succeeded to get the CID. In both cases, the hypothesis was wrong. In 7 hours of process, Bitswap did not help our honest node in retrieving our CID. However, this does not mean that there are no possibilities to get the information in the system. If an honest node is directly connected to a provider, the CID will be advertised through Bitswap and the honest node will be able to download the CID.

Chapter 5

Counter-measures

Our main concern is to avoid IPFS being a victim of similar attacks. After evaluating the impact of our attack on IPFS, we discuss the possible counter-measures. The presented counter-measures would make the attack less feasible for the attacker. In this chapter, we also discuss the advantages and disadvantages of each counter-measure.

5.1 Contact More Peers

When we make an `ADD_PROVIDER` request, we are contacting twenty nodes to announce the content added in the DHT. To make it even more difficult to eclipse a CID, we can increase the number of nodes required to announce a CID. In this way, there is a high probability of a peer having the information about the providers. The lookup algorithm has more probabilities of getting the answer it is looking for with the closest node. This idea requires more machines to eclipse the CID.

Contacting more peers implies that the attacker creates more Virtual Machines or Containers. It requires more resources to develop all these Sybils. As we see in the section 4.5.2, it will cost even more money if we increase the number of nodes required and it will block some attackers from making these attacks. For example, if we increase the number of closest nodes contacted by the `ADD_PROVIDER` to 40. It means that the attacker needs at least 40 Virtual Machines to execute the eclipse attacks on CID. These attacks will cost 1,6\$ per hour, 38,4\$ per day, or 1152\$ per month to realize these attacks.

However, it implies that there is more to process on each node. If there are more requests, it will require more resources to threaten this number of client-side requests. The disadvantage of this idea is that the node must adapt to the resource required even for honest nodes to run IPFS.

5.2 Avoid getting closest peer of CID

Currently, the information of the closest node from a CID is given with the CLI command `./ipfs dht query <PeerID>`. For the attacker, it makes it easy to find this information to create Sybils close to a CID. At the origin, this command is dedicated to finding the closest node to a PeerID. But, it works also with CID because it is in the same address space.

This is the reason why we should never believe in the input of the CLI command. If we always sanitize our input to avoid letting malicious nodes get the information about the closest node from a CID. It will have more difficulties retrieving this information. It basically means that we could verify the entry to the CLI command for the request `./ipfs dht query <PeerID>` in order to avoid the input `<PeerID>` being filled with a CID.

we can assume that the attacker may modify the client as it wishes, so this seems rather weak. However, this counter-measure increases the difficulties for the attacker. For example, we could print the closest peer contacted when we add the content like in section 4.4.

5.3 S/Kademlia

The weaknesses we exploited in our attack were not solely due to a breach we found in the IPFS process of getting the file. It is also due to how Kademlia works and its inherent vulnerabilities.

Baumgart and Mies [2] have proposed to upgrade the original Kademlia protocol [16] with new concepts. They name this updated version *S/Kademlia*.

Implementing S/Kademlia into IPFS may be able to reduce the impact of our attacks thanks to new mechanisms modifying how node identifiers are generated, as well as the behavior of the lookup inside the DHT.

- **NodeID generation** - To mitigate Sybil attacks, S/Kademlia introduces a challenge when someone generates a new ID. This challenge has a certain complexity that will make an attacker spend a considerable amount of resources when generating a large number of Sybil identities.
- **Sibling list** - The concept of *Siblings* is introduced by S/Kademlia. Siblings are all nodes that will be responsible for a key in the DHT. S/Kademlia proposes a way so that every node in the network knows at least s siblings in its close peers. S/Kademlia proposes a way to make the siblings number different from the bucket size k , each node needs to know s closest nodes

(siblings) to an identifier. We illustrate in figure 5.1 that these siblings must be within a defined range of the node ID, we call this range of action *sybil range*. This is a coherent modification since the bucket size reflects the redundancy in connectivity and not the number of replications needed to be stored by the DHT.

In order to respect the constraint of each node knowing s siblings with a high probability, the lookup algorithm for a given key needs to converge to a sibling list of size $\eta \cdot s$, where $\eta \geq (2 * c) \geq 5$. This can be simplified to a siblings list of size $> 5 \cdot s^1$.

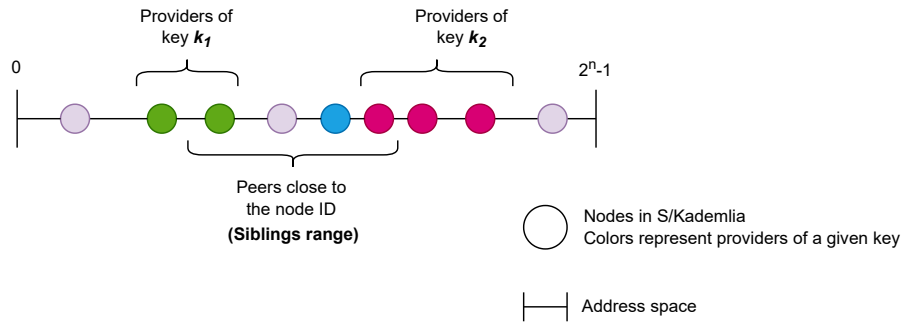


Figure 5.1: Example of a sibling list²

- **Disjoint paths** - In Kademlia, when one user wants to retrieve a key in the DHT, it will have to contact multiple nodes in order to find a path to access this item. Paths to a given key will converge. This makes the routing protocol more vulnerable when malicious nodes are placed on the path to a key. S/Kademlia proposes to make d parallel lookups where nodes on one path will not be present in the others. It makes the lookup procedure more robust to malicious nodes.

Implementing S/Kademlia into IPFS is not a new concept, there are numerous discussions in the IPFS community about it. Some parts of S/Kademlia have been implemented by the IPFS team. However, we are not able to find all modifications in the latest release of the IPFS Client³. Henningsen et al. [12] have also verified the

¹This simplification comes from a course given by Karlsruhe University https://telematics.tm.kit.edu/publications/Files/267/SKademlia_slides_2007.pdf

²Illustration inspired by https://telematics.tm.kit.edu/publications/Files/267/SKademlia_slides_2007.pdf

³The implementation of S/Kademlia, and some parts of it are discussed in pull requests #204, #436, #473 and issue #290 on the git repository of the IPFS DHT implementation - <https://github.com/libp2p/go-libp2p-kad-dht/>

IPFS codebase for the need of their paper and have come to the same conclusion.

5.4 Certified nodes

An attack affecting all decentralized systems is the Sybil attack. In IPFS, changing the current PeerID can be done by changing two lines in the configuration file. We can even change it using the IPFS CLI. This makes it easy for anyone to create a large number of PeerIDs easily.

What makes Sybil attacks hard to tackle is that Sybil nodes cannot be differentiated from other honest nodes. To distinguish these 2 types of nodes, we could introduce the concept of *certified nodes*.

To make the difference between honest nodes and Sybil nodes, we can make a more complex authentication process. This authentication is inspired by permissioned blockchain networks. This system requires everyone that wants to participate in the network to prove their identity [13].

Dinger and Hartenstein [8] make a differentiation between network participants and network nodes. Each participant must only be able to generate a limited number of nodeIDs. To do this, they propose 2 approaches:

1. Use a centralized entity like a Certification Authority that assigns an ID and signs identification requests. The certification Authority should not be part of the P2P network. In this process, we could ask for an identity card from the network participant, to prove the validity of the request. It is a good thing because we could assign a limited number of peers by ID card. This should heavily limit the possibility of Sybil attacks.
2. Generate a limited number of nodeIDs based on a hash of an external identifier, for example, an IP address. They also propose a way to verify these nodeIDs and an algorithm able to verify when somebody cheats the system. Of course, we assume that each participant has a limited number of external identifiers.

The solution involving a centralized entity could even take care of a database where we regroup the whole set of nodes. The peer could ask for the list of valid PeerIDs to verify the peer with which it communicates. However, it implies that we integrate the mechanism of centralized authority in the IPFS system, and the fact that we are using an ID card to identify the node could not be a desired property for IPFS. Having a centralized authority doing the proposed solution also incurs additional costs to network participants and reduces the barriers of entry in the p2p network.

The second solution is not *Sybil-proof* either. This approach cannot work when nodes are running behind a NAT and is not entirely compatible with the new properties from IPv6 addresses.

By taking into account advantages and disadvantages, we still think that the algorithm exploiting external identifiers for nodeID generation is a good idea to explore.

5.5 Scoring

The introduction of a score for each node could manage the distinction between Sybils and regular peers. The principle consists of X closest nodes keeping the information about the score of their X closest node. It has the benefits to avoid the Sybils or peers keeping the information about their score. It is more difficult for Sybil to change its score if it cannot directly access it through its code source. The score is an integer where it increases or decreases if the node realizes a good or bad operation for the system. In fact, we increase the value of the score when the node provides effectively the CID announced. On the opposite, we decrease the score if there is no response to that kind of request. Another example of increasing the score is when information about a provider is requested. If it provides the same answer as the other closest node or if the request cannot retrieve the provider of a CID.

To update the score of a node stored by its closest nodes, each node that has made a request to it needs to contact the x closest nodes and say that the request was successful. Indeed, this node does not profit from the increasing value of other nodes and will be honest.

To establish if a node is a Sybil or a regular node, we compare the score to a threshold. This threshold would be defined in the source code by a default value. So, the score would define if the node works effectively and are reliable. If it is not reliable, we pay not too much attention to its response.

The drawback is the increase in the number of messages and processes realized in each node. It introduces more complexity to the system because it must carry out the increase or decrease of the score on every closest peer.

Zhuang et al. [26] presents the idea of a reputation-based consensus protocol for Blockchain. Their idea is similar to our countermeasure. The reputation is based on the transaction activity, consensus participation, and assets of a node.

Chapter 6

Future Work

In the previous chapters, we demonstrate that it is possible to block access to a given content on the DHT. But, the IPFS network is always evolving with new technologies and with it, new types of vulnerabilities. For this reason, the security of this system has to continuously be evaluated to enhance its reliability.

In this chapter, we will discuss different types of potential vulnerabilities that we noticed during our research. These vulnerabilities will need to be evaluated in order to estimate their impact on IPFS.

6.1 Priority want attack

Until now, we only considered attacking the DHT system. But it is not the only system of IPFS, we could look also at the security of Bitswap. One of our interests is to look at the "Priority want". The priority-want is a 32-bits integer announcing the importance of the request. It allows informing the node getting the query to introduce it in a right place in its queuing request. The queuing request is the list of requests waiting to be treated in every receiving query.

As said in section 2.2.1, Bitswap responds automatically to a request by a **HAVE** when a node requests a CID, that it effectively provides. So, if we manage sybils in order to flood the victim Peer, it will have its queuing request full after a certain amount of time. The possible weakness of this approach consists of two points :

- Limited amount of live requests on the victim node
- High priority on our request

As there is a limited number of requests, we could fill the queue of the victim. It should have as consequence to drop all the new incoming queries.

Moreover, the priority-wait could imply starvation on requests with lower priority. Starvation defines the fact that a request with a high priority never gives room to a request with a lower priority to proceed. Then, Sybils could use this idea to block access to the content of a node.

Due to this weakness, we could fill the list of the victim node and ensure that no request from other nodes is resolved by the victim except the request from our Sybils.

However, it should be confirmed by an analysis of the Bitswap implementation. We already know that there is not a single queue implemented but there is a queue per neighbor as represented in figure 6.1.

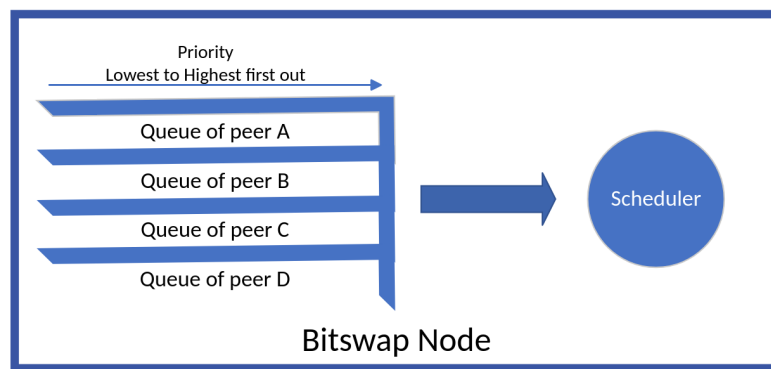


Figure 6.1: Bitswap Queuing

The appeal would be to analyze the feasibility and consequence of such attacks. This attack could be annoying for the neighbors connected to the victim node, but it would not affect the whole system. As one node has a maximum of 900 peers, it may affect 900 peers on the 200 000 nodes connected weekly on the IPFS network¹. It represents only 0,45% of the entire network impacted. This attack can effectively eclipse a node from the rest of Bitswap.

6.2 Privacy Attack

Privacy is also a concern in security. In IPFS, the usage of the system by users could be intercepted by other nodes. The queries inform about the kind of content searched by a user and attackers could gather this information to learn more about the user.

¹The number of 200 000 nodes connected weekly is provided by the Protocol AI

6.2.1 Analyzing current and future requests

With Bitswap, every node advertises the CIDs that they need to all their neighbors. As a reminder, the neighbors are all the peers connected to a given node. It opens the door for a privacy attack. An attacker could easily create and connect a node to a target. The attacker only has to collect the requests received from its neighbors. Then, the attacker could analyze the requests by mapping each CID requested to the content in order to gather information about the target. This attack is based on the current request made by a node, it does not allow us to know the request history of one user. This attack was also mentioned by Balduf et al. [1].

With the new trends, more and more websites are wanting to use a more decentralized way to access them. This is possible via IPFS by combining 2 functionalities: IPNS and DNSLink. Also by using for example the Cloudflare IPFS gateway². IPFS will become more and more used. One example of a website that uses all these technologies is Uniswap, it is the biggest decentralized exchange, as of May 2022³.

If we combine the privacy attack and the use of IPFS to resolve the content of the website, we could make a history stealing without protection on IPFS. History stealing has the purpose to steal information about the websites visited by users [7][5].

It is interesting to analyze this feasibility, the severity, and the consequences of these attacks. Especially, if the resolution of websites through IPFS becomes more frequently used.

6.2.2 Analyzing past requests

When a node asks for a CID, it will become a new provider of the content. It is another interesting property for privacy attacks. We could easily analyze the different CIDs downloaded by a given peer. The attacker only has to connect to the target and request a CID to see if the target provides it to him or not.

If the node responds with a `HAVE`, we know that it has downloaded it in the past. Otherwise, it responds with a `DON'T HAVE` or never response and we know it has not downloaded it.

The attacker could also make history stealing with this behavior and be more precise on the past request realized about a list of CID that we want to test like the popular ones or a specific CID. What makes this attack powerful is the fact that we can know an entire node history[1].

²<https://developers.cloudflare.com/web3/ipfs-gateway/>

³<https://uniswap.org/blog/ipfs-uniswap-interface>

This attack looks like the previous one. But the interest is in past content resolutions, whereas the previous one is more focused on the current and future resolutions.

6.3 Consequences for Filecoin

IPFS allows storing and transferring files in a decentralized way. Unfortunately, it has no mechanism that incentivizes its users to store files for the longer term⁴.

Filecoin is one of the biggest projects built on top of the IPFS network. It enables longer-term data storage on IPFS by adding economic incentives when someone stores a file.

IPFS and Filecoin are two complementary protocols. The more users there are on IPFS, the more users will be interested in Filecoin and vice-versa.

When a user pays for storing its data using Filecoin, it can ask for fault-tolerant storage. The user will ask for a certain quality of service and wants his data to be available at all times.

It would be interesting to analyze the impact of our attack on the different mechanisms of Filecoin and what happens when a file from Filecoin, stored on IPFS, is eclipsed.

⁴<https://docs.filecoin.io/about-filecoin/ipfs-and-filecoin/>

Chapter 7

Related Work

Since the launch of IPFS in 2013 IPFS has gained popularity, but the scientific literature on IPFS is still limited. We can easily find the original whitepaper of Benet [4], but the IPFS documentation is not maintained at the same pace as the evolving ecosystem. In this section, we briefly discuss the articles used and the link they have with our research.

7.1 The Sybil Attack

Peer-to-peer systems have many advantages and disadvantages. They also have vulnerabilities inherent to the way they work. Large-scale p2p systems rely on a large number of distinct nodes in the network to mitigate the effects of hostile nodes or ones having dysfunctions. In his seminal paper, Douceur [9] defines an attack where one entity forges multiple identities (nodes) to have a larger influence in the system. He named this the *Sybil attack*.

To mitigate this attack, peer-to-peer systems need to put in place an identification authority that will verify that each node corresponds to a different entity. An alternative to this identification authority is to impose resource-demanding challenges to validate identities. Moreover, Douceur [9] explains that in peer-to-peer systems, Sybil attacks cannot be prevented by only impeding.

7.2 Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network

Marcus et al. [15] describe an eclipse attack on the Ethereum cryptocurrency. It is an interesting paper for our work due to the use of Kademlia in Ethereum. Even

more, the eclipse attack is like ours because it targets a peer-to-peer network. However, the difference is in the target of their attack eclipse.

In their document, they eclipse a victim node by monopolizing all its connections. It allows the attacker to filter all the packets received or sent by the target. The target gets a partial view of the blockchain and global information due to the attack. This paper raises multiple vulnerabilities about the use of Kademlia in Ethereum and a way of deploying an effective eclipse attack on peers.

So, this paper is one of our guidelines to understand the weaknesses of Kademlia and the establishment of an eclipse attack in a such system.

7.3 Mapping the Interplanetary Filesystem

In order to discover more about the Interplanetary Filesystem, we use this article of Henningsen et al. [12] that describes the uses of DHT by this system. The paper evaluates the performance of DHT in IPFS. It is really interesting because it provides an in-depth overview of the usage of DHT on IPFS by collecting pertinent measurements.

We have also reused CIDs they collected when crawling the IPFS network, for our script calculating the ratio of the resolution of queries between DHT and Bitstamp.

7.4 Disrupting the InterPlanetary File System

Prünster et al. [19] gathered the knowledge from previous eclipse attacks in systems such as Bitcoin [18] and Ethereum [10] and made it work on the live IPFS network. They can poison the routing table of a node and eclipse it from the IPFS network. To accomplish this, they have severely analyzed the IPFS codebase and how the routing works in IPFS.

They have exploited a vulnerability in the way IPFS drops peers from its routing table in order to replace honest peers with malicious ones. They have also demonstrated that this attack can be mounted with a reasonable amount of resources, and could target bootstrap nodes of IPFS. The bootstrap nodes are the first nodes that every IPFS node will connect to when launched and are essential to the correct functioning of IPFS.

Their research is focused on the eclipse of a node while our research focuses on blocking content on the IPFS network. Our work is complementary as we each focus on different vulnerabilities in the routing protocol of IPFS.

Chapter 8

Conclusion

The web is moving towards Web3. This transition gives importance to a new type of system: decentralized networks. It is challenging to assure the security of this type of network due to the lack of a central authority. IPFS responds to a big demand from Web3, by allowing a decentralized way of storing content and hosting websites¹. IPFS is a popular system of the Web3 and has many applications and services running on it². With the rise in popularity of IPFS, which already has millions of users weekly through the IPFS HTTP gateway³, it is essential to keep this system secure.

Our goal is to help increase the security of IPFS, by doing a security analysis of its content resolution system

During our thorough analysis, we discover weaknesses in the broadcasting of a CID in the DHT. Adding a new CID in IPFS relies on the 20 closest nodes to this CID. These 20 nodes are the same ones that answer get requests about the given CID. We successfully deploy an attack targeting this vulnerability, by replacing the 20 closest nodes to a CID, with malicious ones. These 20 malicious nodes do not process messages concerning the addition of a CID. Consequently, they will not be able to answer queries from other nodes in the network, thus making the CID inaccessible.

During the realization of this work, we faced multiple challenges due to the implementation of IPFS itself and its documentation. IPFS is open-source and the development is updated every month, the development is progressing quickly given the new ideas provided by the large community available on different media like Github, Discord, or the IPFS forum. This is a bazaar. Moreover, the lack of scientific articles and precise documentation involves a deep analysis of the current

¹<https://blog.cloudflare.com/cloudflare-pages-on-ipfs/>

²<https://awesome.ipfs.io/apps/>

³<https://blog.ipfs.io/2020-02-10-our-focus-for-2020/>

implementation. We discover that there was incoherent information given in the documentation and the IPFS source code which makes it a real challenge for us to correctly understand certain aspects.

While performing our security analysis and estimating the feasibility of our attack, we discover some interesting information required in the realization of our attack. We measure the amount of time taken by IPFS to retrieve content and the average is 2,81s. Another interesting result is the number of requests resolved by Bitswap or DHT in IPFS. We conclude that queries are mostly answered only with Bitswap and by not relying on the DHT.

We have contacted Protocols Lab to inform them about the security vulnerabilities highlighted in this paper. We also introduce some counter-measure to avoid this kind of attack and mitigate the influence of Sybils in the IPFS network.

Bibliography

- [1] Leonhard Balduf, Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. Monitoring data requests in decentralized data storage systems: A case study of ipfs. *arXiv preprint arXiv:2104.09202*, 2021.
- [2] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8. IEEE, 2007.
- [3] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
- [4] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. Technical report, 07 2014.
- [5] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Nordic conference on secure it systems*, pages 31–46. Springer, 2011.
- [6] Zoltán Czirkos and Gábor Hosszú. Enhancing the kademlia p2p network. *Periodica Polytechnica Electrical Engineering (Archives)*, 54(3-4):87–92, 2010.
- [7] Adrian Dabrowski, Georg Merzdovnik, Nikolaus Kommenda, and Edgar Weippl. Browser history stealing with captive wi-fi portals. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 234–240. IEEE, 2016.
- [8] Jochen Dinger and Hannes Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8–pp. IEEE, 2006.
- [9] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

- [10] Sebastian Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. Eclipsing ethereum peers with false friends. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 300–309, 2019. doi: 10.1109/EuroSPW.2019.00040.
- [11] Sebastian Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. Mapping the interplanetary filesystem. In *2020 IFIP Networking Conference (Networking)*, pages 289–297, 2020.
- [12] Sebastian A. Henningsen, Martin Florian, Sebastian Rust, and Björn Scheuermann. Mapping the interplanetary filesystem. In *2020 IFIP Networking Conference, Networking 2020, Paris, France, June 22-26, 2020*, pages 289–297. IEEE, 2020. URL <https://ieeexplore.ieee.org/document/9142766>.
- [13] Manlu Liu, Kean Wu, and Jennifer Jie Xu. How will blockchain technology impact auditing and accounting: Permissionless versus permissioned blockchain. *Current Issues in auditing*, 13(2):A19–A29, 2019.
- [14] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. *Proceedings of the 16th international conference on Supercomputing - ICS '02*, 2002. doi: 10.1145/514191.514206.
- [15] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum’s peer-to-peer network. *Cryptology ePrint Archive*, 2018.
- [16] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002. doi: 10.1007/3-540-45748-8\{_}5.
- [17] Gianluca Moro, Sonia Bergamaschi, Sam Joseph, Jean-Henry Morin, and Aris M Ouksel. *Databases, Information Systems, and Peer-to-Peer Computing: International Workshops, DBISP2P 2005/2006, Trondheim, Norway, August 28-29, 2006, Revised Selected Papers*, volume 4125. Springer, 2007.
- [18] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.
- [19] Bernd Prünster, Alexander Marsalek, and Thomas Zefferer. Total eclipse of the heart—disrupting the interplanetary file system. *arXiv preprint arXiv:2011.00874*, 2020.
- [20] Sean Rhea, Dennis Geels, Timothy Roscoe, John Kubiawicz, et al. Handling churn in a dht. In *Proceedings of the USENIX annual technical conference*, volume 6, pages 127–140. Boston, MA, USA, 2004.

- [21] Hosam Rowaihy, William Enck, Patrick McDaniel, and Thomas La Porta. Limiting sybil attacks in structured p2p networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 2596–2600. IEEE, 2007.
- [22] Hector Sanjuan, Samuli Poyhtari, Pedro Teixeira, and Ioannis Psaras. Merkle-crdts: Merkle-dags meet crdts. *arXiv preprint arXiv:2004.00107*, 2020.
- [23] Atul Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *IEEE INFOCOM*. Citeseer, 2006.
- [24] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *International Workshop on Peer-to-Peer Systems*, pages 261–269. Springer, 2002.
- [25] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005. doi: 10.1007/s10922-005-4441-x.
- [26] Qianwei Zhuang, Yuan Liu, Lisi Chen, and Zhengpeng Ai. Proof of reputation: A reputation-based consensus protocol for blockchain based systems. In *Proceedings of the 2019 International Electronics Communication Conference*, pages 131–138, 2019.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl