Monitoring and Workload Characterization in the IPFS Network

Master Semester Project

Simon Jacob

June 9, 2023

Supervisors: Dr. Vero Estrada-Galiñanes, Pasindu Nivanthaka Tennage,
Advisor: Prof. Dr. Bryan Ford

DEDIS Lab, EPFL

Acknowledgments

This semester project would not have been possible without the help of my supervisors, Dr. Vero Estrada-Galiñanes and Pasindu Tennage. Specifically, I want to thank Vero for shaping the general direction of the project and Pasindu for contributing with his input on statistical methods.

I would also like to thank Prof. Dr. Bryan Ford for supervising this semester project.

Finally, I would like to acknowledge the help of Leo Balduf, who is the original creator of some tools used in this work and professionally answered many questions about their usage.

Contents

Chapter 1

Introduction

Decentralized networks offer the promise of a more resilient and egalitarian landscape for data storage and content distribution. They enhance security and redundancy by dispersing data across multiple nodes, and empower users by eliminating reliance on centralized authorities. In an era characterized by growing data privacy concerns, the vulnerability of centralized networks to outages has become increasingly apparent. In this context, understanding and enhancing decentralized networks like IPFS becomes critical.

A key player driving this transformation is the InterPlanetary File System (IPFS), an innovative protocol and network forging the way towards a decentralized future.

This work aims to contribute to the ongoing discourse surrounding IPFS and its role in shaping the future of decentral networks by making the following concrete contributions:

1. Provides an analysis of a publicly available dataset, offering new insights into the practical applications of IPFS (chapter 3), which can help identify potential improvements.
2. Presents a method for measuring the level of decentralization and the availability of resources of an important project that is using IPFS, Wikipedia on IPFS (chapter 4). This helps us assert and measure the independence and robustness of this project.
3. Repeats the data collection and evaluation based on a previously published data request monitoring setup (chapter 5) from two years ago. Having current results is important, as IPFS is constantly evolving. For this part, we also make code modifications to the original setup available to make future work easier.

We also make the entire source code (mostly Python) that we wrote throughout this work available, together with some additional results[1].

1.1 Internet Regulations and Online Censorship

While internet regulations have always been crucial in curbing access to illegal or illegitimate content, the rise of recent technologies increases the risk of excessive internet censorship, threatening the freedom of information. A particularly concerning aspect is that these systems are often deployed without public transparency [1].

Even the open accessibility of Wikipedia, an important project for accessing free knowledge all over the world, cannot be taken for granted as most of its control is placed in the hands of a few, who oversee the domain name, the database, and the front-end. A 2017 study found evidence of unexplained, sudden changes in article access rates in 15 countries, suggesting the influence of authoritarian governments on knowledge accessibility [2].

Moreover, access to Wikipedia appears to be influenced by government entities even within Western democracies. A study conducted after the June 2013 NSA/PRISM surveillance revelations noted a decline in the traffic of Wikipedia articles on privacy-sensitive topics. This decline suggests the chilling effect of government surveillance, impacting the public's willingness to access certain information, despite its technical availability [3].

In contrast to the normal internet, as a decentralized system, IPFS inherently resists censorship through its P2P architecture. If a single peer is taken down or blocked, the shared content remains available on the network as long as at least one peer provides it. With these circumstances in mind, exploring potential solutions, such as decentralized networks, becomes even more crucial in the ongoing fight for information freedom. This reinforces the significance of our research on IPFS, as it offers a potential solution to counteract the increasing trend of information control by centralized entities.

1.2 Roadmap

Chapter 2 outlines the motivation for this research, and provides an overview of IPFS's key concepts and more technical aspects necessary to understand the subsequent chapters of this report.

Chapter 3 dives into a detailed analysis of a publicly available gateway dataset published in a previous paper, shedding new light on the practical applications and real-world use-cases of IPFS.

[1] https://github.com/dedis/student_23_ipfs_workload_analysis

Chapter 4 zooms in on an important project that is using IPFS, namely Wikipedia on IPFS [4]. Here we propose and apply a method to measure the level of decentralization and to assess the availability of Wikipedia articles on IPFS by querying the *DHT*, a table that has a mapping between content and nodes storing the content.

Finally, chapter 5 details our efforts to reproduce results from a data request monitoring setup, originally published in a paper two years ago [5]. To achieve this, we collect and analyze messages from the *BitSwap* protocol, a crucial component for data transfer in IPFS.

Chapter 6 concludes this work by giving a short summary of the key findings and sketching ideas for future work.

Chapter 2

Background and Related Work

This chapter gives an overview of relevant implementation and usage details of IPFS that are important to keep in mind while reading this work and the related code. For further information, we refer to the official documentation[1] or documentation from the IPFS community[2]. It also introduces the *Wikipedia on IPFS* project here and summarizes previous work related to our project.

2.1 IPFS

The *IPFS* (**I**nter**P**lanetary **F**ile **S**ystem) network[3] is a peer-to-peer (P2P) distributed file system created in 2014 [6]. Its key features include tamper-resistance due to content-based addressing of files, no single point of failure due to decentralization and improved resilience against censorship due to its P2P network structure, allowing content to persist across multiple nodes even if some are taken offline or blocked. In addition, it strives to optimize bandwidth, a resource that is critical in the current age of data.

Moreover, the architecture of IPFS inherently ensures data persistence. Even if the original provider of the data goes offline, the data will continue to be accessible on the network as long as any other node has the content stored or cached, creating a robust system that is resistant to network failures.

The company behind IPFS, Protocol Labs[4], is actively developing the project and creating various services around it.

Several projects already make use of IPFS[5]. A notable example that we will explore in more detail in chapter 4 is the Wikipedia on IPFS project, which

[1] https://docs.ipfs.tech/concepts/
[2] For example, https://dweb-primer.ipfs.io/
[3] https://ipfs.tech/
[4] https://protocol.ai/
[5] A list is available here: https://ecosystem.ipfs.tech/

seeks to provide an uncensorable version of Wikipedia[6].

While IPFS markets itself as a file system, it is typically not used like a traditional file system where files (e.g. a text document) are often changed and overwritten. Instead, it is mainly used to store and share files that are not expected to be changed frequently, as any changes to a file will generate a new unique hash, thereby treating it as a different file entirely.

There are several clients available to interact with the IPFS network, the most popular one being *Kubo*, a Go-based implementation that we also use for our work [7].

2.1.1 CIDs

Unlike traditional file systems, IPFS uses content-based addressing instead of location-based addressing. Content is addressed based on the content itself, not the location of it. For each uploaded file, IPFS creates a so called *CID* (**C**ontent **ID**entifier), a cryptographic hash based on the content of the file. CIDs have two important properties:

1. Any difference in the content results in a different CID.
2. If two users upload the exact same content, the corresponding generated CID will also be the same.

Note that CIDs come in two different versions:

1. The old version (CIDv0) is a base58-encoded hash, and easily recognizable as a string of length 46 starting with ”Qm”.
2. The new version (CIDv1) is more flexible and can many different encodings, although the default is a base32 encoding. CIDs created with the new version tend to be a bit longer and always start with ”ba”.

The ipfs toolkit has a built-in command to convert from v0 to v1: `ipfs cid format -v 1 -b base32 <CID_v0>`

CID types

In IPFS, files and directories are handled in a unique way to optimize decentralized data storage and retrieval. When a directory is uploaded to IPFS, it is represented as a special object with links to the contents of the directory. Each of these links have a unique CID, that corresponds to a file or subdirectory within the uploaded directory. When a larger file is uploaded to IPFS, it is automatically split up into smaller chunks (the default chunk size is 256KB, but can be changed) and unique CIDs are generated for each

[6] https://blog.ipfs.tech/24-uncensorable-wikipedia/
[7] https://docs.ipfs.tech/install/command-line/

chunk. Similar to a directory, a large file is represented by an object with links to these chunks. The chunks together then can be used to reassemble the original file.

A CID can therefore correspond to any of the following:

- A single file or chunk of a file.
- A directory, represented by an object with links to the CIDs of these parts.
- A file, split up into smaller parts, represented by an object with links to the CIDs of these parts.

Similar to the CID of a file, the CID of a directory changes when a file in it is changed, or if a file is added/removed.

To gain information about the type of CID and its size, the following command can be used: `ipfs files stat /ipfs/<CID>`.

There is also a special type of directory, called a *HAMT sharded* directory, which is optimized for fast indexing of large directories. However, it seems that this type of directory is only used for the Wikipedia on IPFS Project [8]

Finally, to download a CID use `ipfs get CID`.

2.1.2 Peers and Providers

A peer is simply put a user running an IPFS node and therefore participating in the network. When a peer initially connects to the network, he trusts a few initial nodes known as "bootstrap" nodes to connect to other peers. The list of these bootstrap nodes can be inspected with the command `ipfs bootstrap list` and it is possible to modify this list [9]. Once the user is connected to the network, the command `ipfs swarm peers` returns a list of peers that are directly connected to the user.

If a peer has some CID stored and announces this to the network, he becomes a *provider* for this CID. Note that a peer might be a provider for a directory without being a provider for the contents of the directory. In this case, the peer would simply provide the object representing the directory, but not the contents of the directory itself.

Each peer is associated with a unique cryptographic hash called the *PeerID*. Similar to the CID hash, the PeerID hash seems to be available in different versions:

[8] https://ipfs-search.readthedocs.io/en/latest/ipfs_datatypes.html
[9] https://docs.ipfs.tech/how-to/modify-bootstrap-list/

1. A base58-encoded hash, starting with "Qm" and total length of 46. Careful: these PeerIDs can easily be confused with CIDs encoded with CIDv0, as they look exactly the same.
2. A string starting with "12D3" of length 52. This is the most common format and probably also the new default.

Some useful commands that we used for our work:

- To find providers for a CID, the following command can be used `ipfs dht findprovs <CID>`.
- To find more information about a specific peer, such as the IP address and the protocols this peer supports, the following command can be used `ipfs dht findpeer <PeerID>`. Note that this command can fail sometimes (e.g. with `Error: routing: not found`) because peers can be behind a firewall or unreachable for other reasons. In chapter 4, we simply call such a peer *unreachable*.
- The command `ipfs id <PeerID>` returns additional information, such as the client software the peer uses.
- The command `ipfs swarm connect /ip4/<IP_ADDRESS>/tcp/4001/p2p/<PeerID>` directly connects to a specific peer. Note that the connected peers change constantly, due to *churn* (peers joining and leaving the network), or, when requesting content, a connection to the corresponding providers is build.
- The command `ipfs config Identity.PeerID` returns the peerID of the current IPFS node.

With default network settings, i.e. Network Address Translation (NAT) enabled and without forwarding any ports, a peer on IPFS is generally connected to a few hundred peers at most, as evidenced in previous work[7]. This number can be massively increased up to 25,000-30,000 simultaneously connected peers[10] by exposing the machine to the internet, e.g. by disabling NAT or by forwarding the port on which the peer announces itself to the network (usually 4001). However, extra care must be taken, as exposing a machine to the public internet leads to increased security risks. For a tutorial on how an optimal configuration looks like, we refer to the official documentation[11].

2.1.3 IPNS

IPNS (**I**nter**P**lanetary **N**ame **S**ystem) is a naming system that allows the mapping of human readable addresses to CIDs in IPFS. We only make use of this

[10] https://grafana.monitoring.ipfs.trudi.group/d/E0amoF3nk/ipfs-realtime-monitoring

[11] https://docs.ipfs.tech/how-to/nat-configuration

feature in 4 to get the CID belonging to a Wikipedia article. Here is a simple example of how we use IPNS to resolve a Wikipedia URL to the corresponding CID: `ipfs resolve -r /ipns/en.wikipedia-on-ipfs.org/wiki/Book`

2.1.4 Gateways

A gateway in IPFS provides access to content on the IPFS network through traditional HTTP protocols, serving as a bridge between the traditional web and the IPFS network. A basic gateway consists of two parts: A web server (e.g. nginx) and an IPFS node. Note that these components may run on different machines, i.e. the gateway web server and the associated IPFS peer can have different IP addresses.

There are *public* gateways that are accessible to anyone on the internet and *private* gateways, which can be configured to only accept requests from a specific group of users or to retrieve a specified set of CIDs.

Protocol Labs maintains a list of available public gateways[12].

Most public gateways implement some form of content blocking to prevent misuse and legal issues. There are publicly available denylists available[13] that contain CIDs associated with potentially harmful or illegal content. This may include copyright-infringing material, content subject to DMCA takedowns, malware/phishing, or defamatory content.

Gateways can be used directly (e.g. by going to `https://{gateway URL}/ipfs/{CID}/{optional path to resource}` in any modern browser) or indirectly, by browsing a web site that loads content from the gateway (e.g. via an HTML tag `<img src=https://{gateway URL}/ipfs/{CID}/{optional path to resource} />`).

Caching

Due to the immutable nature of content on IPFS (changes to a file result in a new CID), gateways often have a large web server cache. One of the largest IPFS gateway operators, Cloudflare, has claimed[14] to achieve a cache hit ratio of 97%, meaning that only 3% of all requests to the gateway are actually forwarded to the IPFS network.

Caching in the web server of a gateway has one other key benefit besides improving performance: If all providers for a specific CID are offline at some point, the CID might still be available in the cache of a web server and hence can still be served to users. This further enhances the robustness and availability of content on the IPFS network.

[12] https://ipfs.github.io/public-gateway-checker/

[13] https://github.com/ipfs/infra/blob/master/ipfs/gateway/denylist.conf

[14] In a keynote at the DI2F workshop at IFIP Networking 2021.

2.1.5 DHT

At its core, IPFS is a *DHT* (**D**istributed **H**ash **T**able), a huge table that maps content to the corresponding providers. This hash table is distributed, i.e. no single peer has the entire table, but each peer has a subset of the table stored as well as information on which peers have other subsets stored. The process of uploading a file in IPFS can be thought of as creating a new entry in the DHT that maps the CID of the file to the PeerID of the uploader. IPFS uses a specific type of DHT known as Kademlia DHT [8], which has some optimizations for network efficiency and speed.

2.1.6 BitSwap

BitSwap is a message-based protocol in IPFS, and plays an essential role in exchanging (”swapping”) data blocks with other peers in the network. If a user requests a CID, he communicates via so-called ”want-lists” to the directly connected peers what CID he wants. The process goes through the following steps, also displayed in 2.1:

- User sends ”want-have” containing the CID to all connected peers.
- Connected peers reply either with ”have” or ”dont-have”.
- User sends ”want-block” to the peers that responded with ”have” in the previous step. If none of the connected peers replied with ”have”, BitSwap queries the DHT to find providers of the CID and connect with them (in an endless loop until a provider is found).
- Peers finally respond with the requested data.

2.1.7 Comparison between DHT and BitSwap

Table 2.1 summarizes the key differences between DHT and BitSwap.

Figure 2.1 shows the entire process that a peer goes through when attempting to fetch content from the IPFS network.

2.2 Wikipedia on IPFS

As mentioned in the introduction, the influence of governmental entities on information access underscores the need for a decentralized solution to ensure the ongoing availability of free information.

Wikipedia on IPFS[15] aims to preserve the accessibility of blocked articles through by storing them decentral on IPFS. The Wikipedia on IPFS project, therefore, presents substantial advantages over traditional Wikipedia, offering

[15] https://github.com/ipfs/distributed-wikipedia-mirror/

	DHT	BitSwap
Purpose	Data structure used for storing and locating stored data, acts as a lookup service in IPFS to find providers for CIDs	Protocol used for exchanging blocks of data in IPFS
Data collection	Active, to gather data we make requests to the IPFS network to locate data storage locations	Mostly passive, runs in the background and collects requests from other peers
Part of the Network	Considers the entire network for providers	Primarily interacts with the directly connected peers for data exchange, but can reach out to other peers during a content request if the directly connected ones do not have the requested data
What we measure	- Decentralization - Redundancy - Content availability	- CID popularity - Peer activity - Other things [5]

Table 2.1: Comparison between DHT and BitSwap in IPFS.

Language	Language Code	Snapshot Date
English	en	09.03.2021
Turkish	tr	19.02.2021
Myanmar / Burmese	my	22.02.2021
Arabic	ar	26.03.2021
Chinese	zh	16.03.2021
Ukrainian	uk	09.03.2022
Russian	ru	12.03.2022
Farsi / Persian	fa	18.08.2021

Table 2.2: The 8 different languages currently available in Wikipedia on IPFS together with their snapshot version.

a more robust mechanism for supporting Wikipedia's mission of providing universal access to information.

Wikipedia on IPFS is based on snapshots available as *ZIM* archives[16], an open format used to store compressed versions of Wikipedia. It is also possible to view Wikipedia offline with a appropriate software.

Wikipedia on IPFS is currently available for 8 different languages, as shown

[16] https://wiki.openzim.org/wiki/OpenZIM

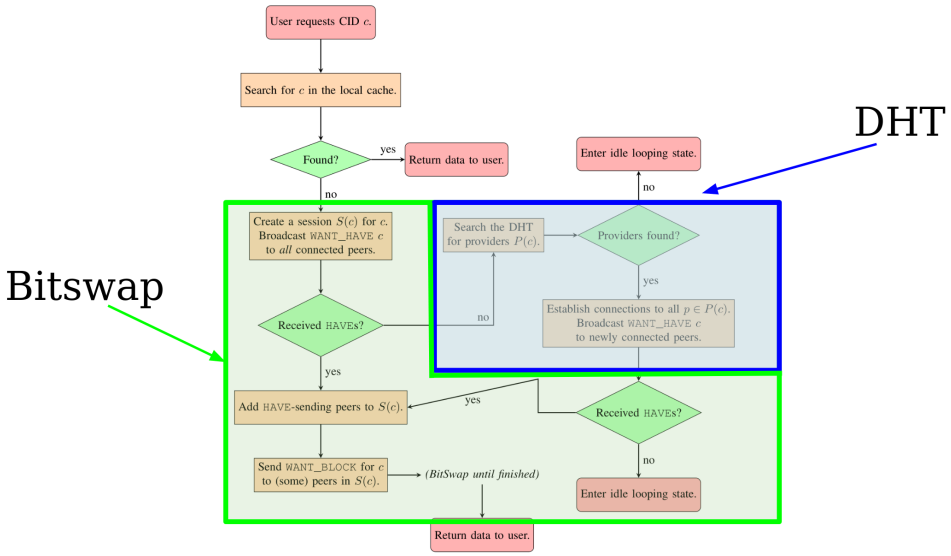

Figure 2.1: The entire process of content retrieval in IPFS, adapted from [5].

in table 2.2.

Some of these languages were added to the project in response to current world events, e.g. the Turkish Wikipedia was added after the Turkish government issued a court order in April 2017[17], permanently restricting access citizen's access to Wikipedia.

2.3 Related Work

The creators of IPFS have analyzed various aspects of IPFS and have published three public datasets in a SIGCOMM paper [9]:

1. **Peer Data**[18]: A SQL database containing information about peers that are engaged in the DHT and their connection behaviour. We looked into the data, but couldn't find any use for this beyond what was already published in the paper. For those interested, we provide the detailed database structure in our code repository.
2. **IPFS Gateway Usage Data**[19]: A nginx log file from a single day of the web server behind the public gateway `https:ipfs.io/`. It contains about 7 million requests. As gateways are responsible for a large portion of IPFS traffic, this is definitely the most interesting dataset out of these three and we use it in chapter 3.

[17] `https://turkeyblocks.org/2017/04/29/wikipedia-blocked-turkey/`
[18] Available on IPFS as `bafybeigkawbwjxa325rhul5vodzxb5uof73neszqe6477nilzziw5k5oj4`
[19] Available on IPFS as `bafybeiftyvcar3vh7zua3xakxkb2h5ppo4giu5f3rkpsqgcfh7n7axxnsa`

3. **Performance Data**[20]: A dataset containing data from performance experiments related to the DHT lookup. While it might be interesting to repeat this experiment with different parameters (e.g. different CID sizes instead of always 0.5MB), there probably isn't anything to learn beyond what was published already.

While the paper contains a lot of information overall, it falls a bit short on the detailed analysis of the public gateway.

A student in the 2022-2023 academic year further analysed the public gateway dataset, providing more details than were published in the original paper [7]. However, the focus of her work lies on the entire dataset as a whole. Our work aims to give new insights by focusing more on individual parts of the dataset and finding patterns between related request. Another part of this project was a performance analysis of an idle and empty IPFS and an idle and empty Swarm node. In our work, we do not analyze the performance of IPFS nodes, but rather try to learn something from the IPFS network. Our IPFS nodes were also empty, because we did not attempt to share content, but not idle, as they were collecting data from IPFS either active (DHT, chapter 4) or passive (BitSwap, chapter 5).

A paper published in 2021 [5] contains a detailed methodology and related tools to monitor data requests on IPFS by collecting BitSwap messages[21]. In our work, we try to reproduce these results. However, this task proved more complex than merely downloading and running the tools: In the last two years the authors of the code and the paper have moved from the old setup to a new setup which is more robust, but instead focuses on collecting metrics related to BitSwap, not on the actual content of the BitSwap messages. Consequently, some tools were left behind and no longer suitable to use with the new format. We made adjustments some to integrate the tools compatible with the older setup into the new one.

Other work focuses on different P2P networks, such as BitTorrent [10]. The comparison to BitTorrent can be useful because it is a widely used P2P system. While IPFS and BitTorrent have many differences, one could, for instance, draw parallels between the role of BitTorrent *seeders* and content providers in IPFS. The main conclusions of the paper highlight that decentralization has its trade-offs, and that incentives play an important role in ensuring content availability on P2P platforms. The incentives in BitTorrent often come from community rules, reputation systems, and ratios maintained on torrent sites, so called *trackers*. In contrast, IPFS provides an optional incentive mechanism through Filecoin [11], a cryptocurrency developed by the same organization

[20] Available on IPFS: `bafybeid7ilj4k4rq27lg45nceq4akdpetav6bcujgiym6vch5ml24tk2t4`

[21] The tools are available here https://github.com/trudi-group/ipfs-tools and https://github.com/trudi-group/ipfs-metric-exporter

as IPFS, where users can pay storage providers to ensure the persistence of content.

Chapter 3

Analysis of a Public Gateway Dataset

Protocol Labs published a nginx log file[1] from the web server running the official gateway[2] and provided a very basic analysis [9].

Other previous work [7] already analyzed the dataset in more detail. Our analysis focused on requests originating from third-party websites, as these were insufficiently explored or inadequately detailed in previous work.

3.1 Overview

The log file contains data collected over approximately a single day on January 2nd, 2021. After filtering out irrelevant entries and requests unrelated to IPFS, the log provides information on nearly 7 million requests.

The nginx log file has 15 columns:

1. Encrypted IP address of the client: This column is not useful, because the encryption is salted, i.e. even if the same IP address appears twice, the encrypted IP address will be different. We will remove this column
2. Timestamp of the request (on the web server).
3. HTTP request information (method path, version): We kept only `GET` requests.
4. HTTP response status code: A 2xx status code indicates success, while a 4xx or 5xx status code indicates failure.
5. Bytes returned (body bytes sent).
6. Request length (in bytes).
7. Request time (in seconds).

[1] https://docs.nginx.com/nginx/admin-guide/monitoring/logging/
[2] https://ipfs.io/ipfs/

8. Upstream response time (in seconds).
9. Upstream header time (in seconds).
10. Cache hit/miss: This is either "HIT", "MISS", or "EXPIRED", indicating that this entry used to be in the cache some time ago, but no longer is.
11. HTTP referrer: This column indicates if the request is coming from a third-party website or if it is made directly.
12. User agent: The software and version the client used. For our analysis, we assume that each user agent corresponds to a unique user. However, in reality it is highly likely that multiple different user have the same user agent.
13. Server name: Can contain the CID.
14. HTTP host: Can contain the CID.
15. HTTP schema: Either http or https.

As previous work has explained, and mentioned in the list above, the relevant CID may appear in one of three columns [7]:

- As part of the HTTP request information (`GET ...`).
- As part of the server name.
- As part of the HTTP host.

However, for our purpose, we treated these equally. We filtered out any rows that did not contain any CID.

3.1.1 Third-Party Websites using the IPFS Gateway

We partitioned the dataset into two categories: requests generated directly and those emanating from third-party websites. We focused our analysis solely on the latter. Figure 3.1 shows that this is approximately a half-half split.

We conducted a three-step manual investigation to classify the nature of the top 100 websites:

1. 1. Check https://www.similarweb.com/.
2. 2. If no clear description is found in the previous step, use a popular search engine to find information about the website.
3. 3. If no clear description is found in the previous step, visit the website.

We found that some NFT platforms were no longer accessible[3].

[3]For example https://emoon.space/, which closed after a security incident

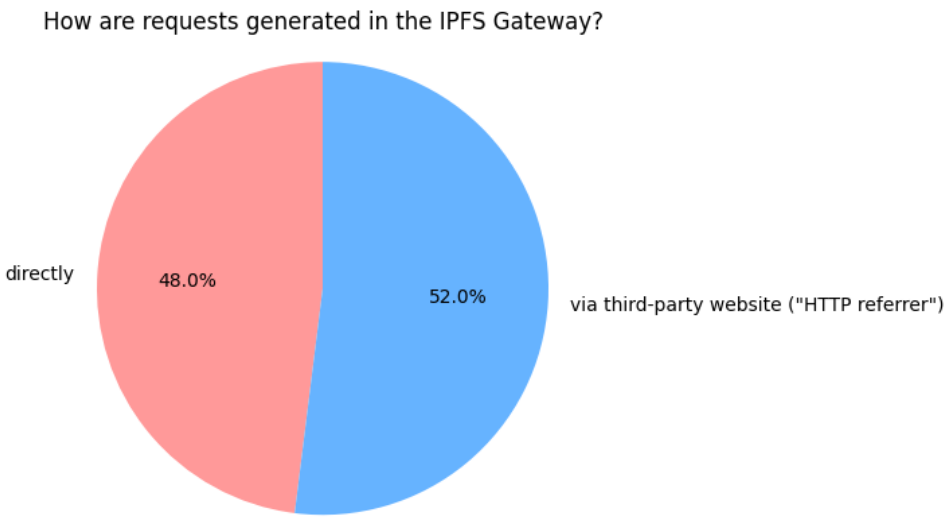

Figure 3.1: Approximately half of the requests are coming from third-party websites, the other half are generated directly.

Figure 3.2 shows the results of this manual investigation. Among the seven websites related to movie streaming, the most popular one is a Chinese movie streaming site, `nunuyy.top`, which alone accounts for 60% of all requests originating from third-party websites.

3.2 Grouping Related Requests

We can group requests together that we believe to be related in the sense that they together access a bigger resource, such as a movie or a folder with images. To detect these grouped requests, we only consider data for a single website, and then sort the dataset by user agent. We then use the following criteria to group related requests together:

- The user agent is the same.
- The requested CID is the same. Note: In this context, a CID can either be a larger file or a folder that is divided into multiple resources, each with associated requests.
- The HTTP referrer is the same.

For this report, we only apply this strategy to the most popular website in the dataset according to the number of requests, `nunuyy.top`. However, our approach can be generally applied and extended to other websites as well.

Figure 3.3 shows for how long users generate streams of requests. In most cases it is rather short, indicating that users rarely watch a movie for the

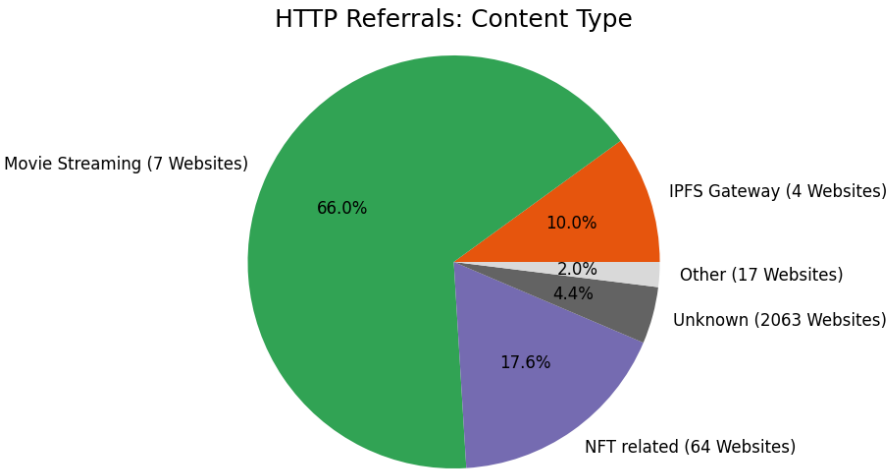

Figure 3.2: Classification of the types of websites that use the IPFS Gateway in their website source code.

entire duration.

Figure 3.4 shows the cache hit ratio of the grouped requests. The U-shaped distribution indicates that the gateway cache does not proactively preload content into the cache, but rather only caches what it has served before. These results show that the gateway cache hit ratio can be decently improved by detecting these request patterns and prefetching content when such a pattern is detected. As shown in previous work [9], cache misses negatively effect the latency by up to four seconds.

3.3 Conclusion

When it comes to the classification of third party websites using IPFS, our findings align with those of previous work [9], but give a much clearer view on the distribution of the websites. By focusing on requests coming from a single website and grouping related requests together, we discovered new patterns in user behavior on this website. We have also found a potential improvement to increase the cache hit ratio on the web server which works by detecting request patterns and predicting the next requests. To verify that this would indeed improve the cache hit ratio, this strategy would need to be implemented and evaluated on a gateway for verification.

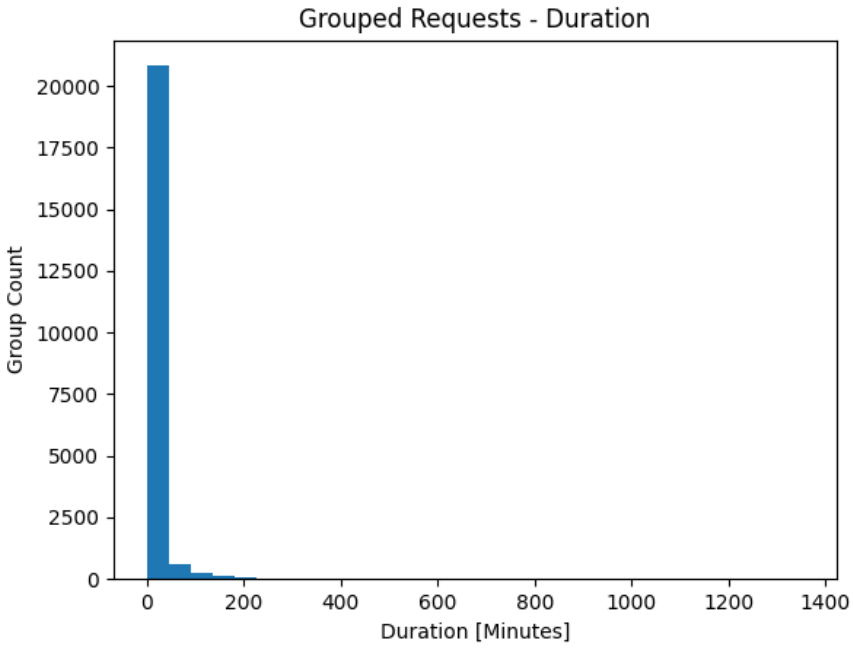

Figure 3.3: Most users only watch movies for shorter amounts of time. The vast majority of users spend fewer than 200 minutes watching a movie. Some users also spend a much longer time on a movie website, this happens most likely due to the users pausing a movie and continuing several hours later.

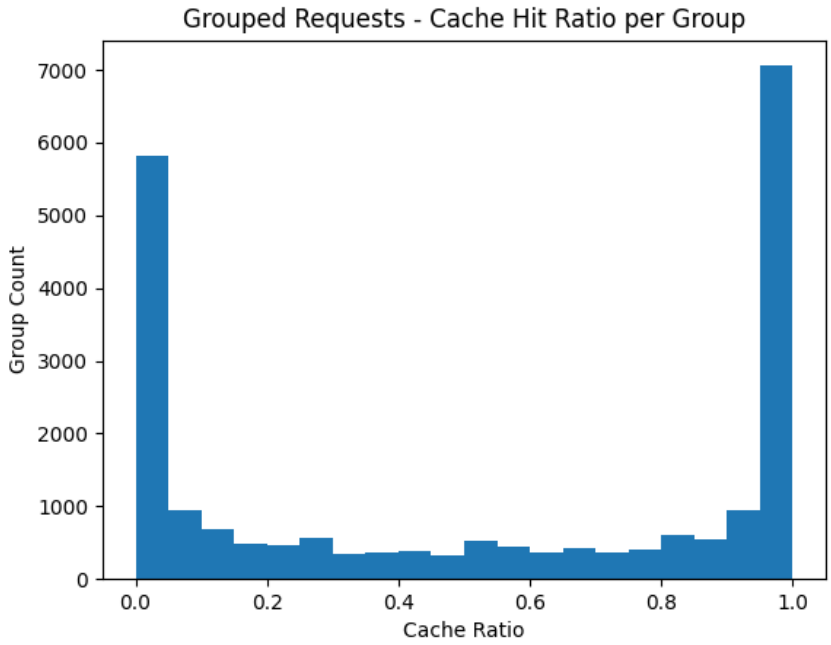

Figure 3.4: The cache hit ratio for this website exhibits a U-shaped distribution.

Chapter 4

Measuring Wikipedia on IPFS

In this chapter, we present our approach to measure Wikipedia on IPFS [1] and evaluate its current state as a decentralized project. We have also come up with possible explanations for the results.

4.1 Overview

4.1.1 Experiment Setup

Hardware and Software details

We ran the scripts described in this chapter on a server equipped with an Intel(R) Xeon(R) Silver 4216 CPU. We used a virtual machine with 4 vCPUs and 16 GB RAM running Debian 10. We used the go-ipfs client[2] version 0.20.0 with default configuration.

Strategy

We started the experiment on 09.05.2023 in the evening and kept it running for approximately one month until 09.06.2023. In total, we repeated the main loop approximately 680 times for each language[3]

The whole process is divided into three phases, the first and last of which only need to be executed once:

1. **Preparation**: The preparation phase consists of two steps. For each language:

[1] https://github.com/ipfs/distributed-wikipedia-mirror/

[2] https://docs.ipfs.tech/install/command-line/

[3] This is only an approximation, because we sometimes had to repeat the loop due to the IPFS daemon crashing.

a) Scrape URLs using recursion with a depth of one level on the Wikipedia main page. We did this to reduce the total size of articles, as processing millions of articles is infeasible. Even then, for performance reasons we further had to reduce the set of URLs by uniformly sampling a random set of URLs, based on the total size of this set, see next step.

b) Convert URLs to CIDs using *IPNS*, as mentioned in the introduction. We only kept URLs that successfully resolve to a CID. This unfortunately seems to be a limitation of IPNS, and is especially likely to happen if obscure characters appear in the URL. Figure 4.1 shows how many articles we were sampling from in the end.

2. **Main loop**: The main loop consists of four steps. For better performance, the main loop is highly parallelized. For each language:

   a) Randomly sample 2.5% of articles (together with the corresponding CID).

   b) Check if the article is available on the website by requesting it and examining the HTML response.

   c) Run the command `ipfs dht findprovs <CID>` to find providers.

   d) Check if providers are reachable using the command `ipfs dht findpeer <PeerID>`. We consider providers unreachable if this command is unsuccessful.

   e) Check if the ipfs daemon has crashed as this has indeed happened occasionally. In that case, rerun the previous steps.

   We also built in some failure tolerance: if `ipfs dht findprovs <CID>` fails to produce a reasonable output, we give it another try. If the article website is not reachable on the website, we also retry once. Similarly, if `ipfs dht findpeer <PeerID>` fails, we allow up to two additional retries.

3. **Post processing**: For each language, we remove unreachable providers from the list of providers.

4.2 Results

To analyze the data, we skipped files that didn't have any providers for any article. We know that the IPFS daemon process crashed occasionally, which could be one reason for the existence of these files. However, the existence of these files in cases where the daemon didn't crash indicates that there may be other factors at play. We suspect some other issue with our client setup, as results with zero providers do not match with the rest of our results.

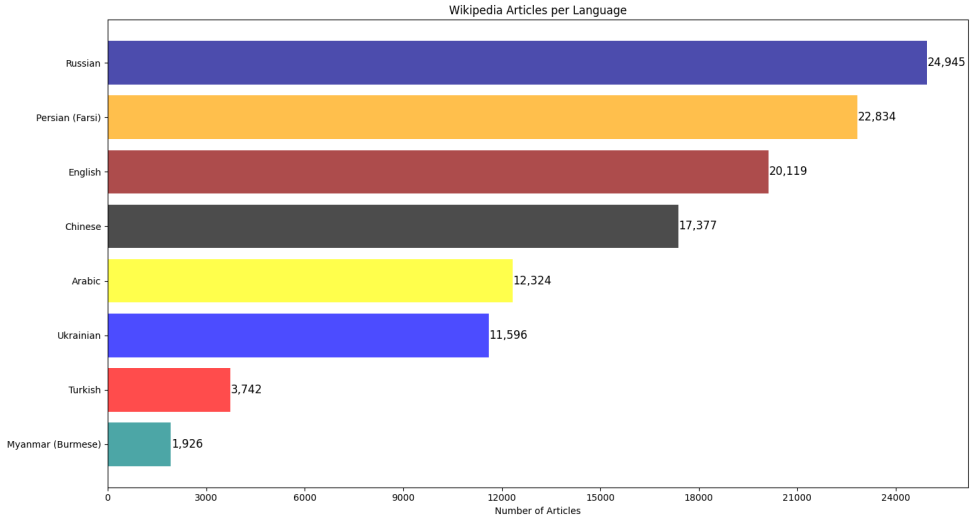

Figure 4.1: The number of articles we were sampling from. A script that recursively collected links from the main pages and one level below it found these articles. Then, we filtered out articles that couldn't resolve to a CID with IPNS. We also tried to compute the coverage to compare these numbers with the current Wikipedia, however that wasn't quite possible as the Wikipedia website contains links to articles that don't exist in the snapshot versions.

Figure 4.2 shows the number of unique peers that are contributing to the Wikipedia on IPFS project, i.e. that have at least one article stored. We can see that the English version of Wikipedia has the most contributing peers with mostly between 16 and 18 in May, with a sudden decrease starting at the end of May and continuing to decrease until the end of the experiment with only 6 contributing peers. The situation is more concerning for other languages at the end of the experiment, as only one to four peers were still contributing. The only possible explanation we could come up with is that several providers for the English Wikipedia must have disconnected in quick succession.

Figure 4.3 shows the *article availability on IPFS*, which is given by the expression $\frac{\texttt{number of available articles}}{\texttt{total number of sampled articles}}$. Available articles are articles that have at least one reachable provider. We can see that it usually fluctuates around 99%-100% with a notable drop near the end of May. For the English language, the article availability drops down to 65%, the lowest that we have measured in our experiment. Interestingly, this drop seems to happen at around around the same time as the decrease in peers contributing to the English version, indicating that several significant providers may have gone offline.. However, unlike the number of contributing peers, the article availability ratio quickly recovers again.

Similarly, figure 4.2 shows the *article availability on the website*, which is again given by the expression $\frac{\texttt{number of available articles}}{\texttt{total number of sampled articles}}$, however, this

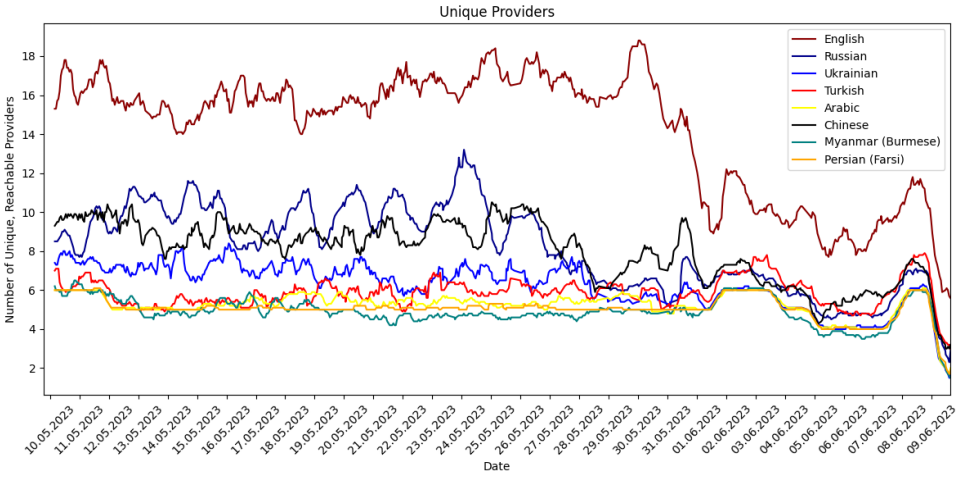

Figure 4.2: The number of peers contributing to the Wikipedia on IPFS project for different languages. To smooth out short-term fluctuations and highlight longer term trends, the values for the y-axis are calculated with a moving average of size 10.

time we define available articles as articles that are accessible on the website. The website maintains a very high article availability throughout the entire experiment: between 99% and 100% of Wikipedia articles remain accessible at any given time. This indicates that even if articles are not accessible through IPFS, they are likely to remain available on the website. We strongly suspect that this high availability is due to caching on the web server running the gateway. Another possible explanation, since we removed unreachable peers, is that a provider of an article might have been unreachable for our IPFS client for whatever reason. It could still have been reachable for the IPFS client sitting behind the gateway. We believe that these results are reasonable, as when manually browsing the website, it only happens very occasionally that an article is inaccessible. Nonetheless, we encountered such a scenario multiple times and show an example screenshot of such an occurrence in figure 4.5.

Most of the time, when an article was unavailable on the website, it was also not available on IPFS. However, on rare occasions, some articles that were available on IPFS, but not on the website. One of these cases comes from an article where our script seems to have stored the URL in a wrong format. In other cases we believe that a temporary overload on the web server serving Wikipedia on IPFS is the most likely reason for that.

4.3 Conclusion

The Wikipedia on IPFS project can be considered somewhat decentralized with several peers contributing, especially for the English language, as it had

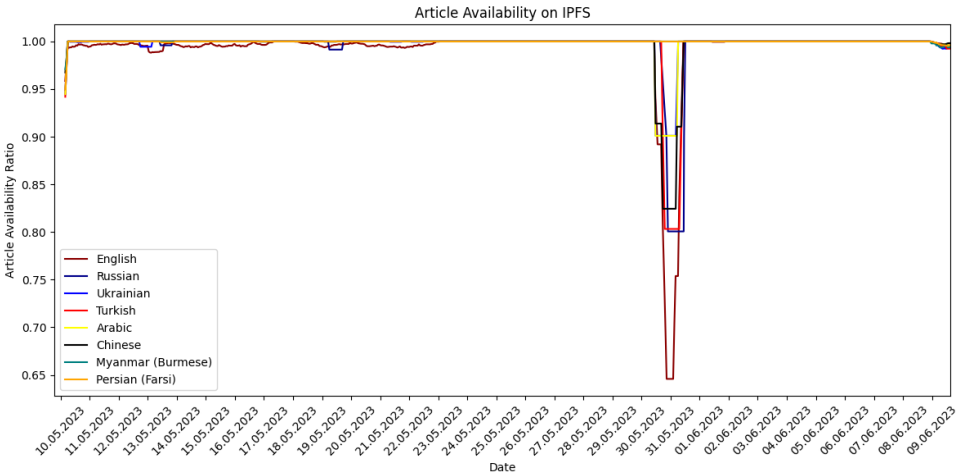

Figure 4.3: The article availability on IPFS for different languages. To smooth out short-term fluctuations and highlight longer term trends, the values for the y-axis are calculated with a moving average of size 10.

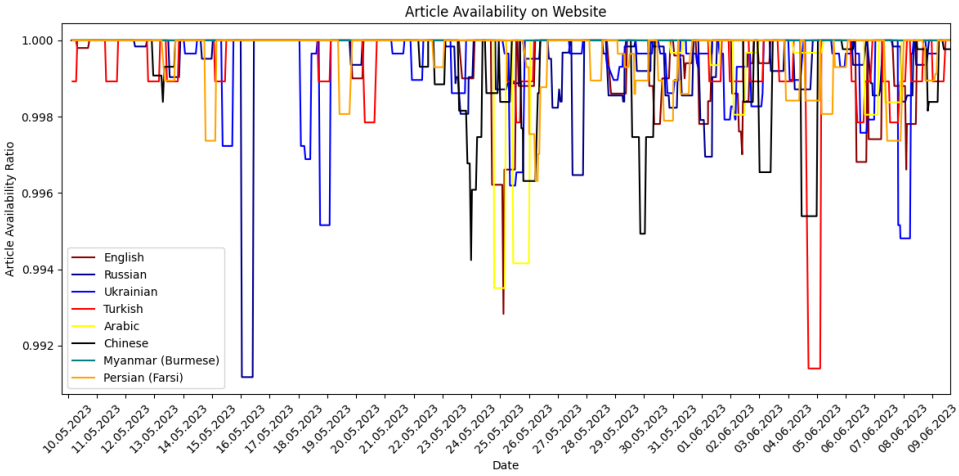

Figure 4.4: The article availability on the website for different languages. To smooth out short-term fluctuations and highlight longer term trends, the values for the y-axis are calculated with a moving average of size 10.

Figure 4.5: An example screenshot showing what happens when a Wikipedia article is not available on the Website. In this case, we manually verified that the article about the "Blue whale" became temporarily unavailable both on the website and on IPFS for a short period of time

.

up to 18 providers. The trend at the end of the experiment is a bit concerning, with only one or a few peers contributing for many languages and we have decided to continue our measurement for some time after the report to further analyze this trend. The new results will be available in our code repository. Both the website and IPFS manage to achieve an article availability of over 99% throughout the entire experiment, with a small exception in the availability on IPFS. We also believe that the website's gateway cache plays a critical role in maintaining constant article availability by offsetting any drops in IPFS availability through its cache storage. Overall, we think that the project could achieve even higher article availability ratios for both IPFS and the website if more peers contributed to it.

Some questions that remain:

- Would using a different sample strategy and/or different sample size yield different results?
- How much redundancy is there, i.e., what is the average number of peers that have stored an article?
- Similarly, how is the content distributed over the peers, i.e. do all peers contribute equally or do some peers contribute more than others?

Chapter 5

Monitoring Data Requests with BitSwap

In this chapter, we delve deeper into the BitSwap protocol, showing how we collect and analyze data requests from the IPFS network. By monitoring BitSwap messages, we can measure how often CIDs are requested and how peers behave.

This entire chapter heavily draws upon tools and methodologies from previous work [5]. However, due to the outdated nature of some of these tools, we made some necessary adaptations[1].

5.1 Overview

When a peer requests a CID, they follow the process depicted in figure 2.1 in the introduction.

5.1.1 Anatomy of a JSON BitSwap Message

Listing 5.1 presents a BitSwap message example received in JSON format. Under normal circumstances, we expect to receive each BitSwap message twice:

- First, as a `WANT_HAVE` broadcast indicating that a peer wants a specific CID and asks the connected peers if they have this CID stored. This is indicated both by the field `"cancel": false` and by the field `"priority"` being larger than zero.

[1] Our adaptations used for this report are currently available as a branch here: https://github.com/S-u-m-u-n/ipfs-tools and https://github.com/S-u-m-u-n/ipfs-metric-exporter

- Second, as a `CANCEL` broadcast, indicating that the peer is no longer interested in this specific CID. This may be due to one of two reasons:
  1. The peer has successfully retrieved the CID
  2. The peer has manually cancelled the request and is suddenly no longer interested in the CID, e.g. a user pressing CTRL+C after running `ipfs get <CID>`

However, it's also possible to receive multiple `WANT_HAVE` requests without any CANCEL in between.

Although we can analyze this JSON format, we further process the received data using a BitSwap unification tool for two reasons:

1. To match the related `WANT_HAVE` and `CANCEL` BitSwap requests.
2. Since we are using two monitoring nodes, to mark received duplicates.

```
{
  "timestamp": "2023-05-16T10:15:03.921388813Z",
  "peer": "12D3Ko****",
  "bitswap_message": {
    "wantlist_entries": [
      {
        "priority": 2147483284,
        "cancel": false,
        "send_dont_have": false,
        "cid": {
          "/": "bafkre****"
        },
        "want_type": 1
      }
    ],
    "full_wantlist": false,
    "blocks": [],
    "block_presences": [],
    "connected_addresses": [
      "/ip4/65.1**.**.***/udp/4001/quic-v1"
    ]
  }
}
```

Listing 5.1: Full example of a BitSwap message in JSON format.

5.1.2 Explanation of Tools

This subsection covers the crucial concepts for understanding the BitSwap monitoring setup. However, for more details, we recommend exploring the repositories as they both contain decent documentation.

We use two different code repositories:

1. `ipfs-metric-exporter`[2]: This repository mainly contains a modified version of *Kubo* (the go-ipfs client) to collect and export some metrics. We refer to each instance of this modified Kubo as a *monitoring* node, or simply *monitor*, throughout the remainder of this section. This

[2] https://github.com/trudi-group/ipfs-metric-exporter

repository also contains the Docker setup which runs the entire setup (see below). One handy feature of this plugin: Instead of using `ipfs swarm peers` on each monitor to get the peers connected to each monitor, we can make use of the HTTP API endpoints and simply call `curl http://127.0.0.1:8432/metric_plugin/v1/sample_peer_metadata` on the host machine running the docker setup (or `:8433` for the second monitor) to get a list of connected peers (and more detailed information about these peers).

2. `ipfs-tools`[3]: This repository contains code for Docker images that are required for the setup. It also includes additional tools, some of which we use and explain in the following sections.

The setup to collect BitSwap messages includes the following components:

- An AMQP server, in this case RabbitMQ.
- At least one monitoring node connected to the IPFS network pushing messages to the AMQP server (in our setup, we use two monitoring nodes).
- A BitSwap monitoring client: a TCP server that connects to the AMQP server and receives the BitSwap messages.

BitSwap monitoring (`bitswap-monitoring-client`)

The BitSwap monitoring client is essentially a TCP server that connects to the AMQP server and collects the received messages.

It also (optionally) logs the messages to disk as compressed JSON (`.json.gz`). Note that it logs both BitSwap messages and connection events; an example of a connection event is provided in listing 5.2. Connection events can be used to better understand BitSwap messages, for example because a peer might disconnect due to some network error and reconnect immediately later.

```
{
  "timestamp": "2023-06-06T13:28:48.075007897Z",
  "peer": "12D3Ko****",
  "connection_event": {
    "remote": "/ip4/65.1**.***.***/udp/4001/quic",
    "connection_event_type": 0
  }
}
```

Listing 5.2: Full example of a connection event in JSON format. The 0 indicates that this peer connected to us, a 1 would indicate that he disconnected.

To get the number of requested CIDs in a `.json.gz` file, the following bash command can be used:

[3] https://github.com/trudi-group/ipfs-tools

```
zcat 2023-06-05_23-55-24_UTC.json.gz |
jq 'select(.bitswap_message.wantlist_entries != null) |
.bitswap_message.wantlist_entries[] |
select(.cancel == false) |
.cid."/"' | wc -l
```

Identifying Public Gateways (`ipfs-gateway-finder`)

This tool downloads a list of public gateways[4] and creates a small CID with filled with random bytes for each gateway. It then sends requests for these CIDs via the gateways, checking for the appearance of any BitSwap messages containing this CID. If such a message appears, we conclude that it must be coming from the gateway, as it is extremely unlikely that any other peer would request the random CID.

Unifying BitSwap traces (`unify-bitswap-traces`)

This tool takes BitSwap traces collected from multiple monitors and 'unifies' them, meaning it identifies and labels duplicate entries. The tool also matches `WANT_HAVE` requests with their corresponding `CANCEL` request.

The resulting CSV has 17 columns:

1. **monitor_id**: Which monitor received this BitSwap message.
2. **matched_to_monitor_id**: For inter-monitor requests, the monitor that received it first.
3. **match_time_diff_ms**: For inter-monitor requests, the time difference between the requests. In our case, this will usually be 0, because we were running the monitors on the same machine.
4. **global_duplicate_time_diff_ms**: For inter-monitor requests, the same as the column before. However, for intra-monitor duplicates, the time difference between this request and the last one.
5. **message_id**: A simple ID, starting from 1 and counting up. Unique for each row.
6. **message_type**: Usually set to 1, indicating a normal message[5].
7. **timestamp_seconds**: Timestamp when the message was received in seconds.
8. **timestamp_subsec_milliseconds**: Timestamp when the message was received in milliseconds.

[4] Obtained here: https://ipfs.github.io/public-gateway-checker/

[5] Check the source code for the actual meaning: https://github.com/trudi-group/ipfs-tools/blob/master/common/src/wantlist.rs#L54-L56

9. **peer_id**: The peer that sent this message.
10. **address**: The IP address and protocol this peer used.
11. **priority**: The priority of this message. Usually it's either a very large number (e.g. 2147483284), indicating the start of a new request or 0, indicating that this message `CANCEL`s an earlier request.
12. **entry_type**: Usually either 1, indicating a `CANCEL` or 4, indicating a `WANT_HAVE`[6].
13. **cid**: The requested CID.
14. **duplicate_status**: Whether this message was received multiple times without a `CANCEL` in between. [7]
15. **sliding_window_smallest_match**: Indicates the smallest sliding window that matched for a sliding window duplicate request entry. We refer to the documentation of the unification tool to understand how it uses sliding windows.
16. **secs_since_earlier_message**: If the message `CANCEL`s an earlier request, the time between them in seconds.
17. **upgrades_earlier_request**: Whether this message upgrades an earlier request. Usually set to `false`.

There are two kinds of duplicates:

1. **Inter-monitor**: Monitor A received a message and monitor B (possibly a bit delayed) receives the same message or vice-versa. This usually happens when a peer is connected to both monitors.
2. **Intra-monitor**: A monitor receives the same request (i.e. same peer requesting the same CID), without a CANCEL or disconnect message in between. These duplicates depend on the connection events and are a bit unreliable to find, as it is possible that we have missed the disconnection event of the peer and they also appear way less often. Intra-monitor duplicates have a `duplicate_status` value different from 0.

5.1.3 Results

We implement the following measurements, partially based on the methodologies outlined in previous work [5]:

[6] Check the source code for the actual meaning: https://github.com/trudi-group/ipfs-tools/blob/master/common/src/wantlist.rs#L59-L66

[7] Check the source code for the actual meaning: https://github.com/trudi-group/ipfs-tools/blob/master/common/src/wantlist.rs#L88-L93

- **Received** `WANT_HAVE` **Requests**: The number of received requests for CIDs, excluding `CANCEL`s.
- **CID Popularity**: We quantify this by defining two different popularity scores [5]:
  1. **RRP** (**R**aw **R**equest **P**opularity): Total number of requests received for this particular CID. The focus here is more on the popularity of a CID in IPFS.
  2. **URP** (**U**nique **R**equest **P**opularity): Number of distinct peers that requested this particular CID. Here the focus lies more on the peer behavior.
- **IoU** (**I**ntersection **o**ver **U**nion) of the peers connected to the monitors.
- **IoU** (**I**ntersection **o**ver **U**nion) of the received BitSwap messages.
- **Gateway peers vs. non-gateway peers**: To compare how much traffic is generated by gateways compared to normal peers.

5.1.4 Conclusion

We first ran our setup in a sub-optimal way and learned that port forwarding is essential to be connected to a lot of peers and to collect numerous BitSwap messages. We were unable to gather sufficient useful data before the report deadline. Therefore we have decided to repeat the BitSwap monitoring setup and properly finish the rest of this chapter in a PDF in part 3 of our code repository: https://github.com/dedis/student_23_ipfs_workload_analysis.

Chapter 6

Conclusion

In chapter 3, we gained new insights into the gateway usage of IPFS by continuing where previous work had stopped. We learned, in more detail, what kinds of web services use IPFS, and how requests coming from these web services relate to each other.

In chapter 4, we measured the decentralization and the article availability of Wikipedia on IPFS using the DHT. We learned how many peers contribute to Wikipedia on IPFS and explained how articles remain available on the website through the gateway cache, even when no provider is currently online.

Finally, in chapter 5, we modified the code of an existing project that monitors BitSwap messages, in order to analyze the results from the monitoring.

6.1 Future Work

While it is always possible to spend more time analyzing the public gateway dataset, we believe that future work would be better served trying to obtain and analyze a different gateway dataset, either by contacting gateway operators, or, even better, by running their own gateway and collecting data.

The script tools used in chapter 4 can be further improved, as they proved to be not very robust. This experiment could also be repeated by sampling different Wikipedia articles, e.g. articles from the top-100 or top-1000 list of all times [1]. However, it is unclear whether such an undertaking would really lead to new insights into Wikipedia on IPFS. What is probably more interesting is to see the methodology of measuring decentralization and article availability applied to a different project that markets itself as decentralized or heavily

[1] Such a list can for example be obtained here: https://en.wikipedia.org/wiki/Wikipedia:Top_100_most_viewed_articles

depends on IPFS. Perhaps the same ideas can even be adapted to work with a different P2P platform[2].

Especially the BitSwap monitoring setup provides plenty of opportunities for future work. Here are some concrete ideas on how future work could use the BitSwap setup to improve the results from our work:

- The setup can be run with more monitors instead of just two. This is as simple as adding some lines in the `docker-compose.yml` file and making sure that the ports from the new monitor(s) are correctly set. By how much can we increase the coverage of the IPFS network with more monitors?
- Actively influence the connected peers of the monitors by e.g. requesting content, offering content, manually connecting, modifying the bootstrap list, ... Can we also increase coverage of the IPFS network with this strategy? How does it compare to adding more monitors?
- A more advanced concept: similar to how we find public gateways, can we learn more about other peers (e.g. can we find restricted gateways?) by influencing their behavior outside of the BitSwap protocol?

Another intriguing, yet more challenging task (which would require some familiarity with Rust), involves modifying the code of the BitSwap monitoring client to enable a real-time analysis and processing of BitSwap messages as they are received. For example, this modification would allow us to determine why a peer has sent a `CANCEL` request - is it because the peer successfully downloaded the CID? We can find this out by checking if the peer has the CID stored directly after the `CANCEL`.

[2]For example, there was a bounty for putting Wikipedia on Swarm, see https://bounties.gitcoin.co/issue/28926. The best solution as of now seems to be https://bzzwiki.xyz/, which is not nearly as mature as Wikipedia on IPFS, and also isn't very likely to be further improved in the near future

Bibliography

[1] N. Koumartzis and A. Veglis, "Internet regulation: The need for more transparent internet filtering systems and improved measurement of public opinion on internet filtering", *First Monday*, vol. 16, Oct. 2011. DOI: 10.5210/fm.v16i10.3266.

[2] J. Clark, R. Faris, and R. Heacock Jones, "Analyzing accessibility of wikipedia projects around the world", *Berkman Klein Center Research Publication*, no. 2017-4, May 2017. DOI: 10.2139/ssrn.2951312. [Online]. Available: https://ssrn.com/abstract=2951312.

[3] J. Penney, "Chilling effects: Online surveillance and wikipedia use", *Berkeley Technology Law Journal*, vol. 31, no. 1, p. 117, 2016. [Online]. Available: https://ssrn.com/abstract=2769645.

[4] IPFS, *Distributed wikipedia mirror*, https://github.com/ipfs/distributed-wikipedia-mirror/, Accessed: 29 May, 2023, 2023.

[5] L. Balduf, S. Henningsen, M. Florian, S. Rust, and B. Scheuermann, "Monitoring data requests in decentralized data storage systems: A case study of ipfs", in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 658–668. DOI: 10.1109/ICDCS54860.2022.00069.

[6] J. Benet, *Ipfs - content addressed, versioned, p2p file system*, 2014. arXiv: 1407.3561 [cs.NI].

[7] S. Xu, "Dissecting IPFS and Swarm to demystify distributed decentralized storage networks", Master Semester Project Report, EPFL, 2023.

[8] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric", in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 53–65, ISBN: 978-3-540-45748-0.

[9] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: A storage layer for the decentralized web", in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22, Amsterdam, Netherlands: Association for Computing Machinery, 2022, pp. 739–752, ISBN: 9781450394208. DOI: 10.1145/3544216.3544232. [Online]. Available: https://doi.org/10.1145/3544216.3544232.

[10] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The bittorrent p2p file-sharing system: Measurements and analysis", in *Peer-to-Peer Systems IV*, M. Castro and R. van Renesse, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 205–216, ISBN: 978-3-540-31906-1.

[11] P. Labs, *Filecoin: A decentralized storage network*, Accessed: 2023-06-07, 2017. [Online]. Available: https://filecoin.io/filecoin.pdf.