Bachelor's Thesis Computing Science


Radboud University Nijmegen

---

**WTFS - WebTorrent File System**

---

*A decentralized, resilient and web-friendly file system based on torrents*

*Author:*
Spyridon-Alexandros Banos
s1043910

*First supervisor/assessor:*
Prof. Arjen De Vries

*Second assessor:*
Prof. Djoerd Hiemstra

July 1, 2022

## Abstract

Big data has become an integral part of organisations due to the high amount of data distributed through the internet. This need for big data solutions has made decentralized file systems a popular choice for organisations who look for more cost effective approaches. This decentralized approach, usually comes with problems such as lack of data availability, difficulty of use and high complexity. In this paper, we have designed WTFS (WebTorrent File System), a file system that has reduced costs and scaling proportional to its user base due to its decentralized architecture and use of torrents. We also integrated the use of WebTorrent, a torrent client that works in the browser, which makes the synergy with websites a lot easier and user-friendly. Additionally, we incorporated unique clients, called LocalNodes, that act as replicators, making the content available even if all clients disconnect. This component works with dynamic replication to keep the strain on the machines low and achieve data resilience. Finally, we conducted a case study that compares WTFS with file systems such as HDFS, IPFS and Academic Torrents.

# Contents

# Chapter 1

# Introduction

Big Data is an essential part of many organisations [13]. Particularly as Internet and ICT technologies prevail, organisations' decision-making has become data-centric, requiring processing large amounts of data [20]. A few technologies that aim to solve this challenge have been created over the years. One of them is distributed le systems such as GFS and HDFS. These use a distributed approach, where multiple servers replicate the data to achieve resilience and have the main server that handles all the metadata and orchestrates the data ow. Fundamentally distributed systems depend on multiple machines for storage due to the sheer size of data sets.

Consequently, as the demand and volume for data has grown so large, those approaches have become achievable only by larger companies because the hardware needed is costly and the chances of systems errors are higher [16]. The aforementioned le systems fall into the category of centralized distributed systems. That means that data control has a single point of entry, even if it is split across di erent machines.

Other popular approaches to storing and handling big data include BitTorrent-based File Systems [21] and InterPlanetary File System [9], which follow a decentralized architecture. They use the clients' machines to distribute and store data around and maintain the network; the clients hold control of data. As a result, the systems scale proportionally with the number of clients and demand for content, leading to potentially in nite scaling without needing to pay for expensive computer clusters for storage. In contrast, decentralization has limitations such as the lack of fault tolerance due to the dependency on the clients' machines, which could have low specs. Furthermore, the setup for such le systems is quite complex and the integration with websites is sub-optimal. A speci c le system of interest for that demonstration is Academic Torrents (AT).

Academic Torrents is an archive website that follows a decentralized architecture. This website runs on a le system using torrents to create an online archive for Academic Content that is purely hosted by the clients [15]. More speci cally, Torrents use BitTorrent, a peer-to-peer protocol that allows computers that download and upload the same torrent le, to form a group (swarm) and transfer pieces of the le between each other [14]. Hence, Academic Torrents manages to create a space where users from academia distribute their research without the cost of publishing it, while also scaling the availability and speed of the system. Although those claims are valid [25], their project falls into the common problem of decentralized systems: the lack of data resilience. While their approach has tried to have web servers to back up some of the data, a quick look at their content will show that some papers are unavailable since they have 0 seeders (clients hosting the data). Another problem acknowledged by the Authors of AT was the di culty of use [15] - Table 1. To Upload and Download data, the user needs to have a dedicated torrent client; overall, the platform is not user friendly. Then the question that arises from all the points is: How can a Decentralized File System be designed to be resilient, scalable and user friendly?

We derive some sub-questions from the research question above that will be considered throughout the paper.

- How can we reduce the cost of a le system while maintaining data resilience?
- How does such a le system scale with the use of Big Data?
- How can we make such a le system user friendly for web clients?

We are focusing on the torrent approach and suggest a design of a decentralized le system that uses a browser torrent client (WebTorrent [6]) to help with the ease of use in the le system and its integration in Websites, such that it can be more accessible. Also, we extend the usual torrent le system with the inclusion of Specialized Local Clients that act as our replication system, similar to how GFS and HDFS work. That way, we can achieve a dynamic replication system that ensures fault tolerance and persistence of data while also getting all the bene ts of low bandwidth and scaling through the nature of torrents. Finally, we conduct a case study comparing features and properties like resilience, speed, cost and more, with other le systems.

# Chapter 2

# Background

In the Big Data world, file systems are widely used for fault-tolerance, parallelism and performance. Google was the one that made distributed centralized file systems popular with their white paper about GFS [19] for Big Data solutions in 2010. Later, the Hadoop file system [29], which some could say is the successor of GFS, made it available cross-platform and is one of the most used Distributed File Systems to date. Additionally, we look into popular decentralized file systems such as BitTorrent-base File systems, IPFS and Tribler. Thus in this section, we describe some of the notions of HDFS, BitTorrent, IPFS, Tribler, WebTorrent and other technologies to familiarize the reader with the terminology that will be used throughout the rest of the document.

## 2.1 Distributed Centralized File Systems

Centralized file systems are based on the client-server architecture [23]. The clients request data from the servers, usually through a network. Then servers handle the requests through file access protocols and provide the data to the clients. More often than not, the servers are a cluster of multiple servers with a central metadata server orchestrating the data flow and replicating the data to achieve fault tolerance. The advantages of the distributed architecture are scaling, lower latency and erasing the problem of a single point of failure [12].

## 2.2 Hadoop Distributed File System

An example of a Centralized Distributed Architecture is the Hadoop File System [29]. HDFS consists of three main components: the clients, the namenodes and the datanodes. Users request the writing and reading of data. For data writing/storing, the blocksize and how many replicas are needed of the data need to be specified. Then the client splits the file

into blocks of the speci ed size and asks the namenode to store the blocks of data with the speci ed replication factor. Afterwards, the Namenode sends the data to the main datanode, and forwards the data to the other datanodes until the replication factor is achieved. The Namenode is a server that handles the metadata of the system and stands between the client and the datanodes. Lastly, after all the datanodes have received the data, the Namenode stores all the metadata on its hard disk.

Reading of data happens as follows. The user requests a le. The client node then asks the namenode for the le. The namenode answers with a list of all blocks and their coordinates. With that info, the client can directly request the le from the closest datanode.

Finally, we talk about the replication factor. That is the number of times a le is stored in di erent datanodes. With this strategy we ensure fault tolerance and data resilience in case some of our datanodes fail. From the paper of HDFS we know that the datanodes could be commodity hardware prone to failing or breaking down [29].

## 2.3 Distributed Decentralized File Systems

Within the category of Distributed Systems, we also have decentralized systems. That is a category where the users act both as clients and server [8]. Essentially there is no central point of storage and clients themselves are responsible for providing and distributing the data. This approach's main advantage is its potential in nite scaling depending on the number of users and its cost e ectiveness towards the organization that uses such an architecture [22].

## 2.4 BitTorrent based File System

BTFS is a decentralized, peer to peer le sharing system based on torrents [28]. Torrents are les that contain metadata about the les/folders that have to be distributed to other users. With that metadata the person with the torrent le can then look into the trackers pointed to by the torrent le and retrieve a list with peers that have the le. The user can directly ask di erent peers for le chunks and then combines them all to create the le. The process just described is automated through the bittorent protocol [1].

Another term we will see a lot throughout the paper is Magnet Link or Magnet URI. This extension to the protocol allows clients to immediately join a swarm and download les without rst downloading .torrent les. The structure of the link speci es the hashed metadata of the le, the tracker urls and the peer address [2]. In our design, we are going to be using the Magnet URI.

Some other notions of torrents are seeders and leechers. Seeders are users who have downloaded files from torrents and have them complete and available for downloading from other users (seeding). Subsequently, the end person who downloads a file but then does not make it available for other people to access (or has only some parts of it), is called a leecher [24].

Putting all those concepts together, we have a network of users sharing and distributing files, creating a self sustained file system. The advantage of this approach is that there is no upper limit to the replication factor, since the more people who seed the files, the more copies of that file exist in the system. Also, there is essentially no cost involved, since the storage is done on the users machines. Of course, these advantages have their downsides.

- Content becomes unavailable if there is no seeder for a particular torrent [18].
- The performance of users' machines and network connection cannot be guaranteed.
- In order to access the content, you need a dedicated torrent client.

## 2.5 InterPlanetary File System

The InterPlanetary File System is a different approach to decentralization, other than torrents. It is inspired by ideas from Git [11] for Version Control, BitTorrent [14] and Kademlia (distributed hash table) [26] to create a peer-to-peer protocol. The way it works is by allowing the clients to store and rely information/content from anywhere in the world through their storage network. How clients find the content, is through a distributed hash table (DHT), which maps data items to addresses [30].

What IPFS does different than BitTorrent is that it introduces a DHT and thus clients can immediately interact with each other to find where the content is hosted, instead of depending on a Tracker. Also, they use Merkle Directed Acyclic Graphs, where every IPFS client/Node has an id which is its hashed content. That way they can have multiple versions of the same file and also split the file into blocks (just like BitTorrent) so that a user can get the file by merging multiple blocks of the file from various peers [9]. Lastly we want to mention that IPFS is a pretty popular file system and has had a lot of integration with blockchain in order to achieve resilience.

## 2.6 Super Peer Decentralized Systems - Tribler

The design of peer to peer decentralized systems includes a special notion called superpeers. The superpeer is essentially a peer that acts both as a server and a client [10]. One file system that uses superpeers is Tribler

[27]. Tribler is built on top of the BitTorrent protocol and extends it with a social aspect. The creators aim to have BitTorrent swarms between non-anonymous peers that share common interests in order to form social circles within the network. The characteristic of Tribler we want to focus on is the superpeer. Pouwelse et al. note that when new peers connect to the system, they fail to discover and connect to already existing peers. Thus, with a superpeer that is already connected to multiple other peers, a new user can retrieve the list of peers from the superpeer [27].

## 2.7 Conclusion of Background Section

Based on that literature, we aim to solve the high cost of le systems, the lack of integration with the browser that makes the user experience di cult and the unavailability of content, by using multiple aspects from the di erent approaches mentioned above. First, we use a decentralized approach based on BitTorrent. More explicitly we use WebTorrent [6], a torrent client that is based on BitTorrent but extends the protocol to work with WebRTC which enables true peer-to-peer communication and transport of les in the browser. With their API we are able to create a le system that integrates perfectly with the browser while also allowing for processes to run purely in the backend through NodeJS. Secondly, we include superpeers, which we call LocalNodes, and implement them in a way that resembles datanodes from HDFS, to achieve data resilience (more on that in the Design Overview).

# Chapter 3

# Design Overview

## 3.1 General Overview

The WebTorrent File System (WTFS) can be conceptualized in two parts. The first part is to create a torrent-based file system using WebTorrent. This is achieved with the MainNode and Torrent Tracker components, as can be seen in figure 3.1. The second part is to add data resilience to the file system. We achieve that using superpeers called LocalNodes [see figure 3.1].

Figure 3.1: High Level architecture of WTFS

At first we shall describe the system from a higher level perspective. The main core of the system revolves around the MainNode, which acts as a coordinator between Tracker and Clients, based on metadata of both files and clients. The inspiration for the component comes from HDFS' metadata server, the NameNode. Then we have the clients, who are able to upload files as well as download them through a magnet link. This whole process is handled by the WebTorrent API and the tracker. As a second step our design adds the LocalNodes which act as clients that follow instructions

from the MainNode and also as a local data server, just like a DataNode would in HDFS.

The MainNode itself stands as an extra layer between the tracker and the clients/LocalNodes. It is also responsible of displaying a list of all the current Torrents that are present in the Tracker and keeping track of the seeders of all the torrents. Based on the number of seeders, the MainNode should either instruct the LocalNodes to replicate and seed the le (if the number of seeders is below a threshold) or stop seeding and delete the le from the local storage (if the number of seeders is above a threshold). In this way we achieve a hybrid system where the clients host most data, while also having the LocalNodes that would seed data that stop being seeded by other clients in the torrent network, in order to keep them available.

Below in Fig 3.2, we can see a general ow of the whole system and how each component interacts with one another. The arrows describe some of the interactions that happen between the di erent components. Those depict a general functionality and not a speci c case of uploading/downloading a le. An important remark is that arrows between all the LocalNodes and Clients are missing. This is because it would make the Schema too complex and it is more of an internal thing of how torrents work through the BitTorrent Network. These arrows can bee seen in Fig 3.3.

Figure 3.2: Low Level interaction between components of WTFS

Figure 3.3: Torrent interaction between clients

## 3.2 Tracker

The torrent tracker is an essential part of torrents. Its use is to keep track of all the peers in the network and to help them find each other [14]. Essentially when one peer requests a torrent, a message is sent to the tracker indicating the torrent they want and the tracker replies with a list of peers that have previously asked for the same torrent. After the client gets the peer list, it is its job to connect with the rest of the peers. In WebTorrent, when a new client is created, by default a custom BitTorrent-tracker made by the creators of the API [5]. The creator of WebTorrent and contributors have also created a package (bittorrent-tracker ) to setup your own BitTorrent tracker [3]. With a custom tracker, it is as easy as passing its URL as a parameter to the API when creating the client.

This gives a lot of customization capabilities. One could use already existing private trackers that are more trustworthy and could expect most of the clients to be seeders and not leechers. Another one might use open trackers which are free and thus reduce any costs associated with the tracker. For our Proof of Concept, we initially used opentracker.xyz and btorrent.xyz, which worked great. In the end, we opted for a custom tracker in order to have more control over the metadata passed to the MainNode [A.2.4].

## 3.3 MainNode

The MainNode keeps track of the torrents and manages their replication factor based on the number of peers. Firstly, if a new file is uploaded, the MainNode notifies both LocalNodes to download and seed it. We call this action of the LocalNode, replication. For already existing torrents, the MainNode should be able to adjust the local replication of the LocalNodes, depending on the seeders of the files. This is achieved by pinging the server and getting the metadata of torrents and their corresponding number of peers. Then a threshold is set which is compared to the number of peers

a torrent has. If the number of peers is bigger than the threshold MainNode, notifies the LocalNodes, by sending the torrent ID, to stop seeding that torrent [More info: section 3.5]. Similarly if the number of peers has suddenly gone below a threshold, due to peers disconnecting from the swarm or not seeding a file anymore, the MainNode should inform the LocalNodes to replicate the file by sending the torrentID.

## 3.4 LocalNodes

LocalNodes are a part of the implementation of the file system that is responsible for the replication of data. Essentially, they act just like a downloading only client, but the decisions made for it are orchestrated by the MainNode. This component is inspired from both HDFS [29] and the Tribler paper [27] in regards to the replication factor and the superpeer design respectively.

Design choices for the LocalNodes follow the three design principles introduced by Beverly Yang and Hecto Garcia-Molina [10]. The first principle is: A superpeer should always accept new clients. We adapt this in our design by immediately seeding any file uploaded to the system. That way we can discover any new clients and any new torrents that get uploaded. The second principle is: A superpeer should aim to maximize its outdegree. Outdegree in this case means the connection of the superpeers with the normal clients of the network. This follows from the design choice of the previous principle. When a new client connects and uploads a new torrent that is not seeded by anyone else, we immediately seed it and thus discover that new peer, increasing our outdegree. Thus, even if all client stop seeding the torrent, there is no high risk of it disappearing from the system. The third principle is: A super-peer should aim to Minimize TTL. TTL stand for Time-To-Live and in the paper of Yang et al. they use it to refer to the query messages happening between superpeers and regular peers. They aim to decrease the TTL when there is enough outdegree, so that they reduce redundant queries. We interpreted this principle a bit more freely and adapted it to the design of the dynamic replication that our LocalNodes are based on. With the dynamic replication we keep the resources of the LocalNode machines as low as possible without compromising a lot of the outdegree, because we only stop seeding a torrent when the number of seeders is above the threshold.

Now that the design principles of our LocalNodes are established, we will describe in more detail how they work and interact with our system. When a torrent falls below the threshold set for replication, the LocalNode receives a signal and the torrentID or hash from the MainNode. Then the functionality equals that of the client; the LocalNode downloads the torrent through the API call \ add", but with a slight difference. They are implemented in such a way that they are able to seed the files without having a browser open.

This is possible using the API packagewebTorrent-hybrid [7] which can be used for purely backend functionality through Node JS.

The important thing about the LocalNodes is that depending on the use case, the number of them can be tweaked. For example in a smaller company with less nances they could have only one LocalNode, because the scaling with the clients seeding the les, would be enough to raise the replication of said les. We consider these the nice side e ects of using torrents. Another company could use two LocalNodes for increased safety if one of them Fails. Someone would argue that having 3 LocalNode defeats the purpose of the design, since we could use an HDFS implementation with 3 replication factor, due to similar costs. But here we should note that in HDFS, most of the time, all 3 nodes have replicas of all the les, while in WTFS (our implementation) the most popular and downloaded les are seeded by other clients. Ultimately this increases the bandwidth and we can stop seeding the torrents at the LocalNodes, thus achieving lower strain on our machines. Based on the hierarchy paper [17], we believe that 2 or more is better than just one, due to the redundancy we can achieve. In the case that a LocalNode fails, we can rely on the other one to keep seeding the content. Finally, the storage of LocalNodes can be implemented as database like systems or normal disk le system storage, but could also be seedboxes, which are remote servers speci cally designed to download and upload les from peer-to-peer networks.

## 3.5 Client

A client has two main functionalities: upload and download. A client is able to upload a le from their system, upon which a torrent will be created through the API, and the client will start seeding that le. A client can download any other torrent that is being seeded and made available by the system, either by specifying that torrents ID, info hash or MagnetURI.

## 3.6 Dynamic Replication

The dynamic replication is a core component to our design since it minimizes the load on the LocalNodes while maintaining resilience of the les. Firstly, we have a threshold that is the minimum number of seeders we want a le to have. Let us call that threshold and assign it the number 3 (Since that is a replication factor su cient in the HDFS system). We also keep track of the number of seeders per torrent. Let us call thatnum_seeders In our system the MainNode tries to balance those two variables such that the number of seeders of the torrent is bigger or equal to the threshold(num_seeders >= threshold).

Let us show a practical example: When a client uploads a new file, the num_seeders is equal to 1. So the MainNode sees that the num_seeders < threshold, thus it notifies the two LocalNodes to replicate the torrent, making the num_seeder equal to 3, the threshold. The result of that can be seen in the Initial State 1 of Fig 3.4. Then a second client joins the swarm and seeds the same file as client 1. So now there 4 seeders. The MainNode sees that num_seeders > threshold by 1, and thus we can stop seeding the torrent from one of our LocalNodes. Thus the MainNode notifies the LocalNode 2 to stop seeding the torrent (State 2 of fig 3.4), achieving the. Then another client, client 3, starts seeding the same torrent ID, and thus again the num_seeders is 4 > threshold. So now MainNode notifies the other LocalNode to stop seeding as well (State 3 of fig 3.5).

Figure 3.4: Stop replicating on new clients

Another example is when a file is already seeded by more or the same number of clients as the threshold. In that case non of the two LocalNodes seed it. But if a client decides to stop seeding it, then num_seeders < threshold, then the LocalNodes get notified once again to replicate it to match the threshold. This can be seen in fig 3.5.

When clients stop seeding a torrent, LocalNodes intervenes and replicates the file.

Figure 3.5: Clients stop seeding. LocalNodes replicate

## 3.7 Data Resilience

The data resilience is strongly based on the replication factor. By having enough replicas of the files we ensure that even if a system fails, or if all clients stop seeding, we have a point of control in the LocalNodes that surely seed the files with low or even 0 seeders. In this section we are also going to discuss about the torrent side effects that become beneficial to the scaling of the system and its resilience.

Imagine a very popular torrent that wants to be downloaded/seeded by a lot of people. That torrent will inevitably reach a very high replication factor due to a lot of client downloading it and seeding it. Thus there is no need for the LocalNodes to seed it as well. Due to that nature the system scales accordingly to the demand of the data/files. Furthermore, it is unlikely that a substantial amount of clients will suddenly stop seeding a file simultaneously, or want to download a very unpopular file.

In a similar scenario that a file is seeded by one or two people (or even 0), the LocalNodes seed it so that an eventual client that wants the file can still access it and download it. In this scenario the low number of seeders also means a low number of requests for it and thus there is no real throttling or bandwidth issues. As a final remark we will discuss about fault tolerance of the LocalNodes. This is particularly easy to handle, since when a LocalNode breaks down, all that happens is the files being seeded by it, drop their number of peers by one. If the new number is lower than the threshold then the rest of the LocalNodes will try to replicate the file and no data loss would have happened.

## 3.8 System Interactions and Data Flow

One of the core uses of distributed systems is an upload of a file. Below we depict how the data flows through the system interactions in an upload

sequence. Let us now look at how an \Uploading sequence" looks like in our system [See gure 3.6].

1. The client chooses a le that they want to upload to the le system.
2. On the le upload procedure, the API call \ client.seed' is issued - which creates a new torrent from the le and seeds it. Simultaneously with the API call, the Client noti es the MainNode that a le is being uploaded and also the torrent ID gets passed to it.
3. From step 2 there are two outgoing arrows. Thus the 3rd step is going to be split in two.
   (a) After the \Upload File" the WebTorrent API call of \ client.seed' will also check if the le already exists. The check of if the le exists, continues in the tracker. If YES, then we go to 4(b). If NO, then 4(a).
   (b) MainNode waits for the tracker to check if le Exists. If it does exist, nothing happens since the error gets thrown at the Client and the ow stops. Else, the ow continues after the seeding of client to step 6.
4. (a) If the le did not already exist, The WebTorrentAPI creates the torrent from the uploaded le. And then then client starts 5. seeding the torrent. Then the ow continues to 6. Where the MainNode knows the client seeded the torrent successfully and informs the LocalNodes to replicate the le.
   (b) If the le already existed send error message from WebTorrentAPI to the client, that le already exists.
5. Client starts seeding.
6. Inform LocalNodes to replicate le.
7. Replicate le on LocalNode.
8. Replicate le on LocalNode.

Figure 3.6: Upload File Sequence

Here is a diagram of the \Download Sequence" [See gure 3.7].

1. The client nds an existing torrent that he wants to download and gets its corresponding MagnetURI.
2. Then through the API client.add() the Client asks for downloading based on the MagnetURI.
3. Following from step 2 there are two actions that take place.
   (a) After the client tries to download a torrent, the number of peers of the downloaded torrent increases by one. As a side e ect the MainNode needs to check the updated number of peers of the torrent, since it has changed after the new download.
   (b) At the same time when the Client requests to Download a torrent, the API handles the interaction with the Tracker. Essentially the magnetURI is used to identify the torrent requested and the tracker makes a list with all the peers that seed it.
   (c) Send the peer list back to the API. And then the API handles the rest of the le chunks gathered to be downloaded in the Client.
   (d) File downloading from peers.

4. The MainNode node compares the number of seeders of the torrent, which has just been downloaded from the Client, to a set threshold.

5. (a) If the number of seeders is bigger by at least two in comparison to the threshold. In that case, the MainNode sends the torrentID (or infohash) to both LocalNodes and instructs them to stop seeding it.

   (b) Suppose the number of seeders is bigger by exactly one in comparison to the threshold, then the MainNode sends the torrentID to one of the LocalNodes (Which one is to be determined !!! Talk about the slight hierarchy) and instructs it to stop seeding it.

   (c) If the number of seeders is equal to the threshold, then there is no action to be taken.

6. (a) The LocalNode receives signal from the MainNode to stop seeding the speci ed le and removing it from the storage, using the API call \ remove".

   (b) Same as 6(a)

Figure 3.7: Download File Sequence

That concludes the design section. Now that we know how the system works, we first discuss a proof-of-concept implementation and then compare it with existing file systems: HDFS, IPFS and AcademicTorrent's file system. The criteria for comparison are cost, replication, resilience/persistence and user experience.

# Chapter 4

# Proof of Concept

In this section we will describe how we implemented a proof of concept for the WebTorrent File System.

## Website with client

We start with the main website that will allow access to WTFS. The HTML file that holds the web page is accompanied by a Node.js script that uses the WebTorrent API to create a Client associated with the current page. We achieve that by using browserify [4], a package that allows you to require Node.js modules in the browser. We can thus include the WebTorrent package and have access to the API through client-side JavaScript. The functionality is achieved by basic API calls, such as adding torrents and seeding.

Code in appendix: A.2.1.

## Tracker

The tracker is built using the bittorent-tracker package and specifications found on the WebTorrent repository [5]. We code our own implementation in Node.js and not just use the default tracker that the package provides, because the tracker needs to send metadata information to the MainNode. The metadata are sent everytime a new file is uploaded, or downloaded. That way the MainNode gets the torrent hash and the accompanied updated number of peers in order to decide if the LocalNodes should take action.

Code in appendix: A.2.4.

## MainNode

The MainNode is a WebSocket server that just waits and listens for any messages from the Tracker and also detects when clients and LocalNodes

connect to it. Upon receiving new information, we check the client ID. We have assigned special IDs for the LocalNodes and the tracker in order to identify them. The rest of the logic is based on the dynamic replication speci cation we described in our design section. A noteworthy implementation technique was that with the custom IDs we assigned to the WebSocket clients, we are able to store the connection and directly send information to the LocalNodes; to seed or delete a torrent, without iterating over every connection.

Code in appendix: A.2.3.

## LocalNodes

The LocalNodes are built using the WebTorrent-Hybrid package. That means that we can access the API without the browser. We are also connecting with WebSockets to the MainNode and we wait to receive instructions from it. Once a WebSocket message is received, we parse it and read the instruction and the accompanying torrent hash. Then, according to the instruction, we either delete or seed the le using the WebTorrent API. In the case of seeding, we are also supposed to download it and the speci ed path can be passed as a parameter to the API calls.

Code in appendix: A.2.2.

## Conclusion of POC section

Above we gave general insight to how we structured our code to achieve a proof of concept for WTFS. We are able to upload and download les freely, while also replicating the content in the LocalNodes. If you want to see more of the code and test the functionalities yourself, the whole code base can be found in: A.3.1. There are also accompanying instructions in the README le to help with the local setup. A quick preview of the uploading and downloading without the LocalNodes can be found in Appendix A.3.2.

# Chapter 5

# Case Study

In this case study we will compare our file system in a usage of an archive website. Our comparison will be made with the Academic Torrents website which uses BitTorrent architecture to make their data available. We will also compare it with how HDFS and IPFS would handle such data. To start of we scraped the Academic Torrents website and gathered all the torrents under the paper section, as well as their corresponding metadata, such as the name, downloads, seeders, leechers and date added (see A.1.1 for details).

## 5.1 Cost-Total Storage

In total we found 2198 entries with a total of 79403.424GB or approximately 79.4 TB of data. In order to calculate how much it would cost to store that amount of data we can take the pricing of Google Cloud, which is 10 dollars per 1TB/month. In the case of HDFS we would have a replication factor of 3 and thus we would need 3x the storage (Assumptions - we do not include the hardware for any of the solutions. Thus the cost of it could be considered equal through all the File Systems { for the sake of comparison) 79.4 * 3 = 238.2 TB. Google only offers 20TB and then 30TB of storage options with 200 and 300 dollars cost, so we assume that the cost remains the same for more extensive scaling. Thus for 238.2TB we need 2382 dollars per month. IPFS needs a node to cache the data and achieve some sort of resilience. Thus we will treat it as a system with a replication factor of 1. For the consistency of price comparison we are going to assume that the node is using as storage a Google Cloud solution. Thus IPFS would need 1*79.4 = 79.4TB of storage and hence 794 dollars per month.

With Academic torrents and the use of their BitTorrent implementation, they essentially have 0 cost, other than a typical server that keeps track of all the files uploaded to the system and displays them on the website. Though, as we know it, some of the torrents used are also in Web Servers and thus we could say that they also need to pay 794 dollars for

79.4TB of storage, per month, with the 1 replication being used.

Our WTFS , with a con guration of 2 LocalNodes would need 79.4*2 = 158.8 TB. If the LocalNodes work with a cloud solution to have the le locally, it would cost around 1588 dollars per month. As we can see, our implementation is cheaper than HDFS, but more expensive than Academic Torrents and IPFS. In order to understand what our design improves upon both the other two le systems, we need to factor in the data resilience regarding the cost.

## 5.2 Replication/Data resilience

Even though Academic Torrents claims to have webservers backing up the torrents while no one is seeding them, it is not the case for all the les. After the scraping, we run a small analysis on the data and saw that 215 out of 2198 les are not available. We also know that the problem is not that they are very new les that have not been on the webserver yet, because they date back to the year 2014. Our implementation of Webtorrent File System keeps track of all the les that are in the swarm and immediately replicates them to LocalNodes, and thus at minimum with 0 seeders we have a replication factor of 2. The HDFS on the other hand is a bit more expensive but always provides a replication factor of 3. As for IPFS , the price is identical to that of Academic Torrents, but the minimum replication factor is 1. Though, with IPFS, we could add more nodes and achieve a higher replication factor that would actually increase the cost.

In WTFS we compromise the 3rd replication for very unpopular papers that no-one seeds, that is why the compromise is minimal and we save 1/3 of the cost. For popular torrents it will almost always be the case that we will have a replication factor greater than 3 with no upper limit.

Another bene t we have over the Academic Torrents approach is the user friendly nature and integration of our le system with a website. WebTorrent is meant to be run on the browser and thus makes the interaction with the browser seamless.

## 5.3 UI/UX comparison

Now let us describe a problem of Academic Torrents that our le system solves. As mentioned by the authors of AT [15], they acknowledge that their platform is not the most accessible one and that is due to limitations of classic BitTorrent. We overcome these hurdles with the use of WebTorrent, which on upload, it turns les into torrents and on download, it can display them immediately in the browser (if the le is supported on the browser { pdf, jpg, png, txt, mp4, but not xml, xls, docx). Below we show a walk

through of the Academic Torrents uploading of  le and downloading and then compare it with a front end implementation of using WebTorrent.

## Academic Torrent Upload and Download File

For the screenshots of the procedure look at the Appendix A.4.

### Upload

To upload a  le to Academic Torrents, you  rst need to create a torrent from it. We use the qBitTorrent's tracker creator tool. We select a  le and then specify the number of pieces we want our  le to be split in. After pressing \Create Torrent" the torrent  le is saved on the system and can be uploaded to Academic Torrents. The upload page is restricted only to people with an academic email, or the ones that have requested access privately. We have to use the torrent  le we created from the original pdf  le. Even after uploading the  le, it is not even seeded. That is because the seeding does not happen through Academic Torrents themselves. Yet again we need a dedicated BitTorrent client to seed the  le. So, now, we have to go back to qbittorent and seed the torrent we created by specifying the path that the original pdf is located. And after all this procedure we  nally have a torrent  le that is also seeded by us.

### Download

First step to download a  le in AT is by copying the magnet Link or download the torrent  le. We then open the torrent using a local torrent client (ex. qBitTorrent). We download the  le speci ed by the torrent  le we just uploaded and just like that, we downloaded the paper. Then the pdf  le can be opened with a pdf reader (Note: In our test the Magnet Link approach was broken in Academic Torrents).

## WTFS Upload and Download File

For the screenshots of the procedure look at the Appendix A.5.

### Upload

Now we see how the same paper could be accessed by the WTFS front-end, from the Proof of Concept. We can press the \choose Files" button and select a  le from our system. We choose the \Approximations for Binary Gaussian Process Classi cation that we have downloaded from Academic Torrents". And just like that we have uploaded the  le and are already seeding it. We can also see the corresponding magnet link, access the info hash or download the .torrent  le.

Download

We take the magnet link from above and paste it in the download field. Pressing \Download" we get the info hash of the torrent and also seeing it displayed right there in the browser. We can Download the file if we just want it in our system available offline, just with a click of a button. This is extremely straight forward and easy to use. Even better, a website with the use of WTFS could also list all the available papers in a list (just like AT) and by clicking the title, the magnet link could immediately be used without being copied and just displaying the file online. That way you can preview files online, before deciding whether to download them locally.

## 5.4 Final Comparison Table

To conclude the comparison, we create a table showcasing what features and properties each file systems has.

| Comparison of File Systems | | | | |
|---|---|---|---|---|
| Features | WTFS | IPFS | HDFS | AT |
| Data Resilience | Yes | No | Yes | No |
| Dynamic Replication | Yes | No | No | No |
| User Friendliness | Yes | No | No | No |
| Cost | Medium | Low | High | Low |
| Limited Bandwidth | No | No | Yes | No |
| Dependency solely on clients | No | Yes | No | Yes |

# Chapter 6

# Future Work and Discussion

With WTFS being an Open Source project, we would like for the community to further develop research around some of the ideas proposed on this paper. A signi cant  rst step would be the creation of an actual implementation of WTFS. That way extensive testing could be carried out, ranging from bandwidth test, to availability of content, fault tolerance, resilience depending on the setup and more.

The WTFS design itself could be transferred to a few di erent technologies. We chose WebTorrent due to its integration with the web and its lack of connection with the blockchain, but others do not have to choose that. Ultimately, the notions of dynamic replication and the idea of Localnodes could also be implemented on systems like IPFS, where the LocalNodes would not be clients in the BitTorrent Protocol, but some sort of super Node in the IPFS network.

An experiment of great interest would be a user test between Academic Torrents and WebTorrent File System. Even with the current proof of concept, the users could be asked to use both WTFS and AT in a hypothetical scenario, where they would have to upload and download a  le in both systems. Later, an evaluation form would be handed out, that would ask users about their opinion on the two systems through targeted questions on the designs. From the results of the feedback form, we could assess if the WTFS approach is indeed more user friendly.

# Chapter 7

# Conclusion

In conclusion, we presented the design and POC implementation of the WebTorrent File System: a decentralized system with superpeers that use dynamic replication, to provide data resilience and simultaneously alleviate strain on the machines. Additionally, the system is able to integrate with the browser to make web solutions more user friendly and less complex to build.

Outside of the design of the system, we have given a comparison with some of the popular le systems in use, that showcases the bene ts of using our system. WTFS might still be in a rather preliminary, but with the proof of concept we show that there is potential to be used in practice.

# Bibliography


[1] Bep protocol 3. http://www.bittorrent.org/beps/bep_0003.html .

[2] Bep protocol 9. http://bittorrent.org/beps/bep_0009.html .

[3] Bittorrent tracker package. https://github.com/webtorrent/bittorrent-tracker .

[4] Browserify node module. https://browserify.org/ .

[5] Webtorrent default tracker - wss://tracker.webtorrent.io. https://github.com/webtorrent/webtorrent/issues/218 .

[6] Feross Aboukhadijeh. Webtorrent. https://webtorrent.io/ , accessed 21/06/2022.

[7] Feross Aboukhadijeh. Webtorrent-hybrid. https://github.com/webtorrent/webtorrent-hybrid .

[8] P. Baran. On distributed communications networks. IEEE Transactions on Communications Systems, 12(1):1{9, 1964.

[9] Juan Benet. Ipfs - content addressed, versioned, p2p le system.

[10] B. Beverly Yang and H. Garcia-Molina. Designing a super-peer network. In Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405), pages 49{60, 2003.

[11] John D. Blischak, Emily R. Davenport, and Greg Wilson. A quick introduction to version control with git and GitHub. PLOS Computational Biology, 12(1):e1004668, January 2016.

[12] J Blomer. A survey on distributed le system technology. Journal of Physics: Conference Series, 608:012039, may 2015.

[13] Zoran Cekerevac, Zdenek Dvorak, Ludmila Prigoda, and Petar Cekerevac. Big vs small data in micro and small companies. Communications - Scienti c Letters of the University of Zilina , 18(3):34{40, 2016.

[14] Bram Cohen. Incentives build robustness in bittorrent. 2003.

[15] Joseph Paul Cohen and Henry Z. Lo. Academic torrents: A community-maintained distributed repository. In Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment, XSEDE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[16] Amit Kumar Dutta and Ragib Hasan. How much does storage really cost? towards a full cost accounting model for data storage. In Jorn Altmann, Kurt Vanmechelen, and Omer F. Rana, editors, Economics of Grids, Clouds, Systems, and Services, pages 29{43, Cham, 2013. Springer International Publishing.

[17] L. Garces-Erice, E. W. Biersack, P. A. Felber, K. W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In Harald Kosch, Laszlo Boszormenyi, and Hermann Hellwagner, editors, Euro-Par 2003 Parallel Processing, pages 1230{1239, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[18] Konstantin Laufer George K. Thiruvathukal. https://ds.cs.luc.edu/bit_torrent/bit_torrent.html .

[19] Sanjay Ghemawat, Howard Gobio , and Shun-Tak Leung. The Google File System. SIGOPS Oper. Syst. Rev., 37(5):29{43, oct 2003.

[20] Huadong Guo, Lizhe Wang, Fang Chen, and Dong Liang. Scienti c big data and digital earth. Chinese Science Bulletin (Chinese Version), 59:1047, 12 2014.

[21] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. A performance study of bittorrent-like peer-to-peer systems. IEEE Journal on Selected Areas in Communications, 25(1):155{169, 2007.

[22] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A survey of peer-to-peer storage techniques for distributed le systems. In International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II , volume 2, pages 205{213 Vol. 2, 2005.

[23] Florin Isaila. An Overview of File System Architectures, pages 273{289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[24] Sebj rn S ther Birkeland Jahn Arne Johnsen, Lars Erik Karlsen. Peer-to-peer networking with bittorrent. 2005.

[25] Henry Z. Lo and Joseph Paul Cohen. Academic torrents: Scalable data distribution, 2016.

[26] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, Peer-to-Peer Systems, pages 53–65, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[27] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system. Concurrency and Computation: Practice and Experience, 20(2):127–138, 2008.

[28] Johan Pouwelse, Pawel Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In Miguel Castro and Robbert van Renesse, editors, Peer-to-Peer Systems IV, pages 205–216, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[29] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pages 1–10, 2010.

[30] Klaus Wehrle, Stefan Götz, and Simon Rieche. 7. Distributed Hash Tables, pages 79–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

# Appendix A

# Appendix

## A.1 Academic Torrents Scraper

### A.1.1 Academic Torrents Scraper Code

## A.2 Proof Of Concept Code

### A.2.1 Client

### A.2.2 LocalNodes

### A.2.3 MainNode

### A.2.4 Tracker

### A.2.5 README le and Instructions on running the Proof of Concept

## A.3 WTFS Repository and Website

### A.3.1 The whole code repository

### A.3.2 Deployed website of WTFS

## A.4 Download and Upload Screenshots of Academic Torrents

### A.4.1 Upload

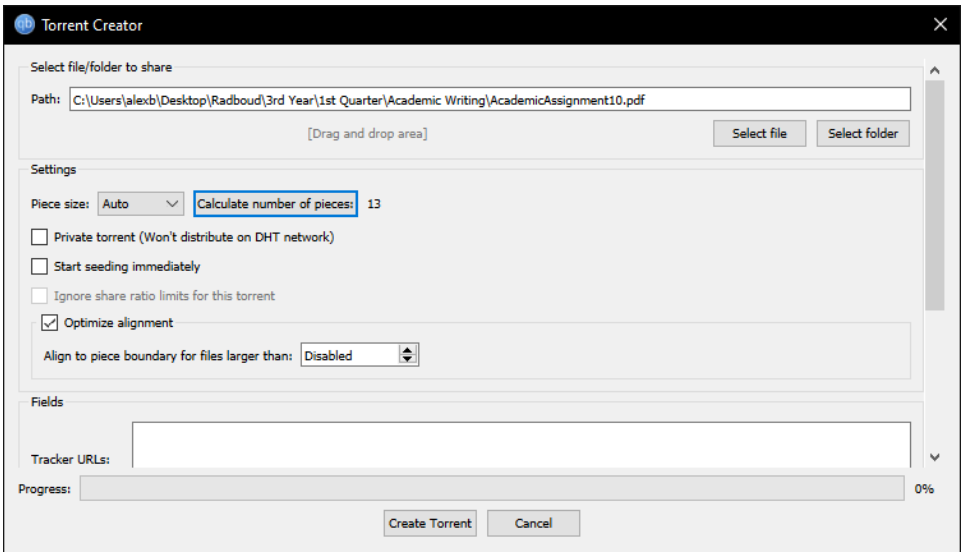


Figure A.1: Create torrent le from speci ed le path and #pieces selection

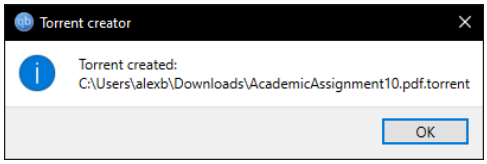


Figure A.2: Torrent created

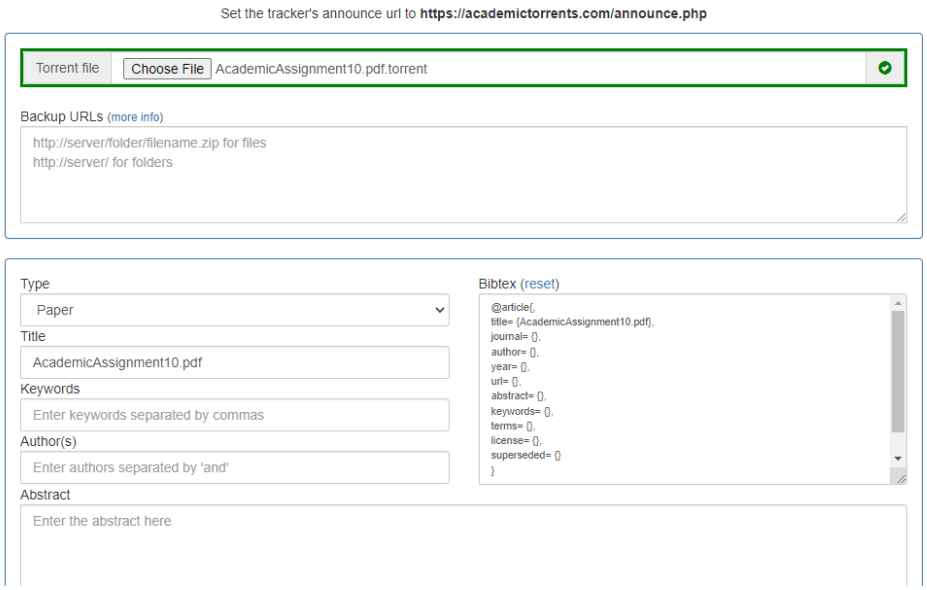


Figure A.3: Choose torrent file from system to upload to AT

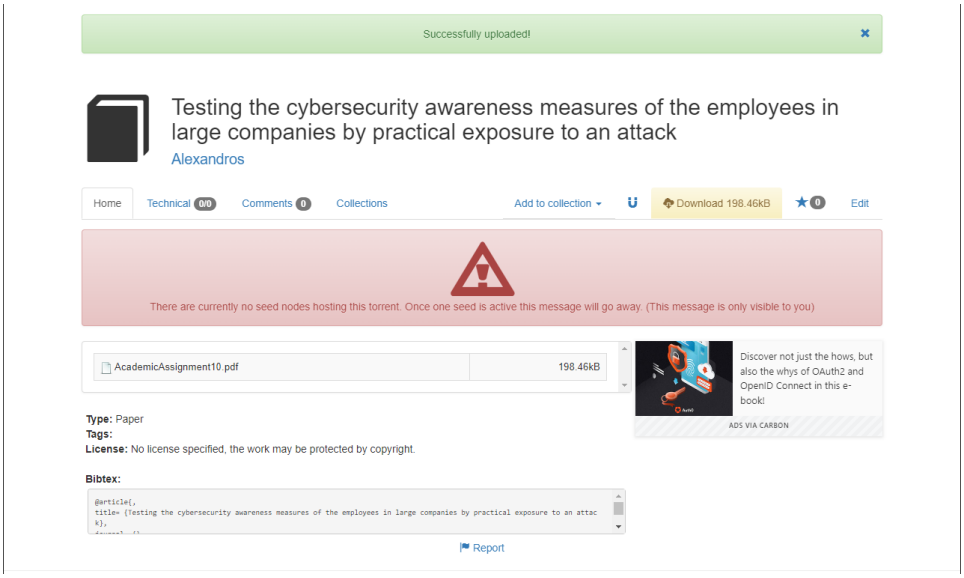


Figure A.4: Successfully uploaded torrent to AT, but no seeding

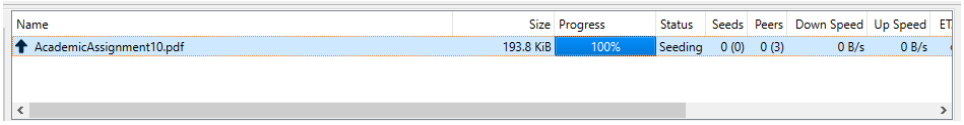


Figure A.5: Upload actual le to dedicated torrent client to seed le

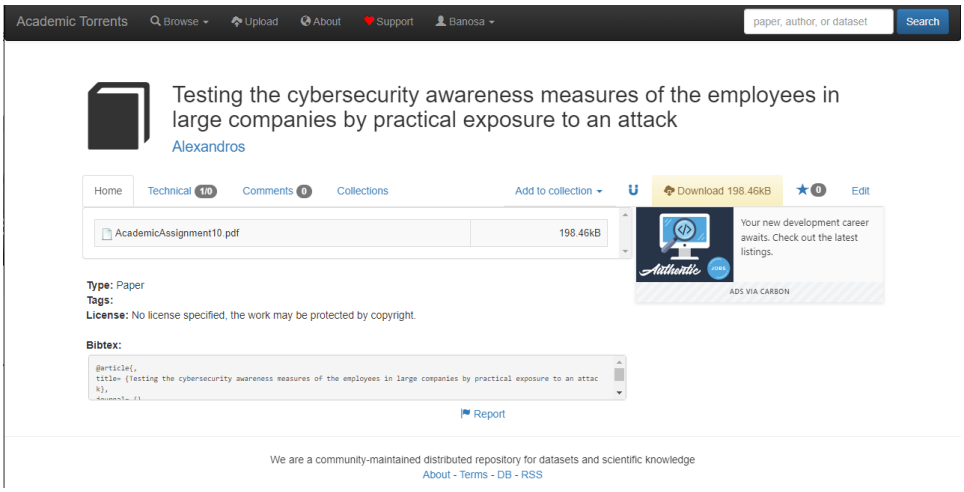


Figure A.6: Torrent available for download by AT, while we are seeding

### A.4.2 Download


Figure A.7: Magnet Link or Torrent le download from AT

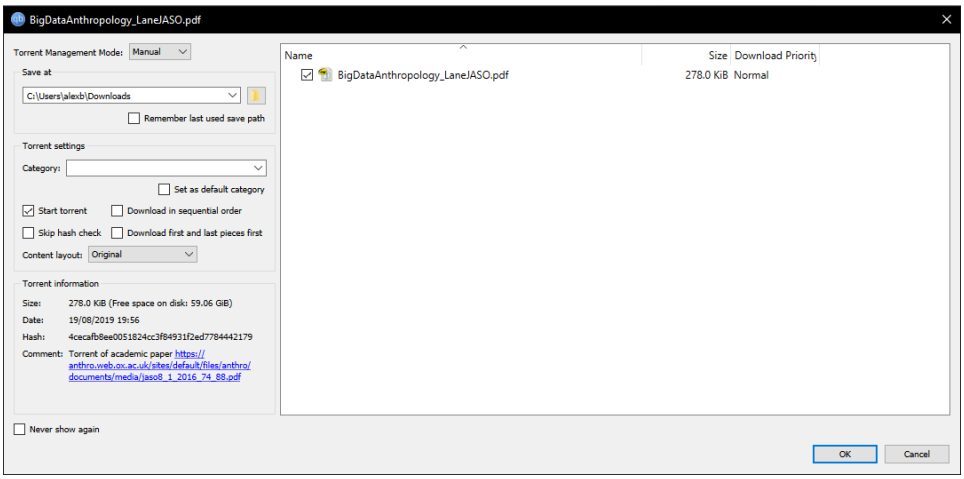


Figure A.8: Use a dedicated torrent client to download actual le from torrent

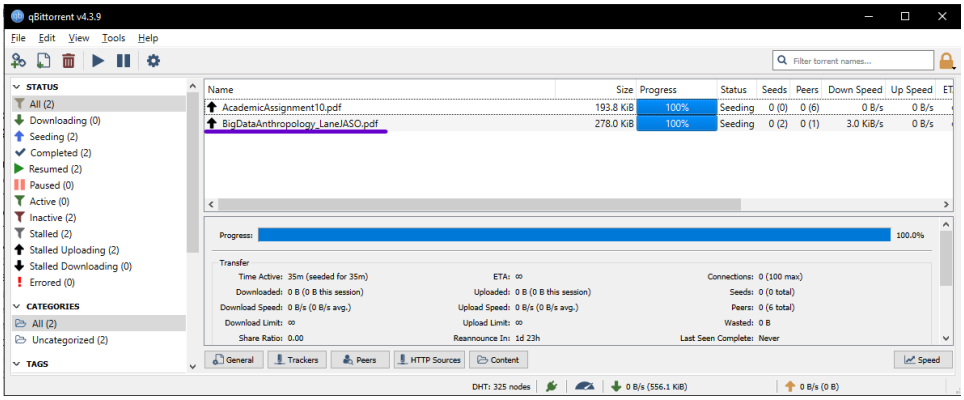


Figure A.9: Successful local download

## A.5 Download and Upload Screenshots of WebTorrent File System

### A.5.1 Upload

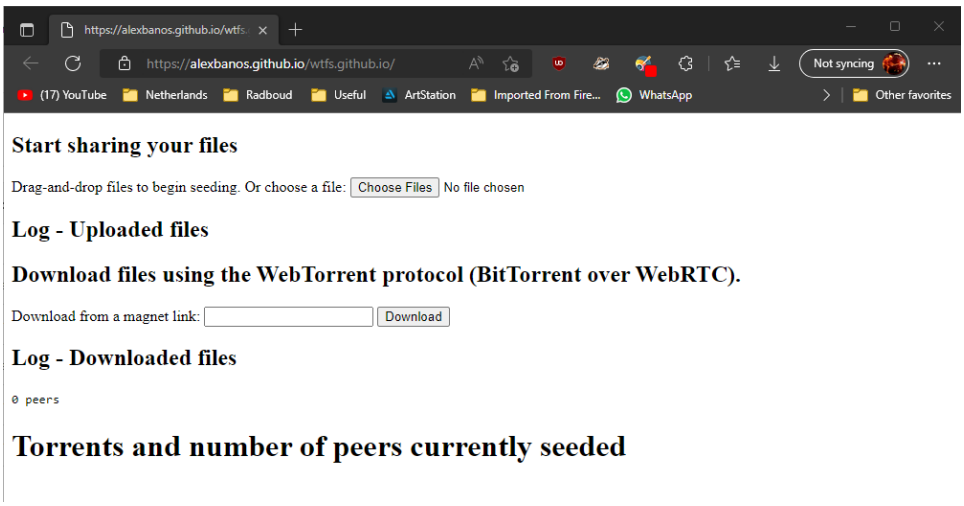


Figure A.10: Interface of WTFS. Press \Choose  les" button to upload

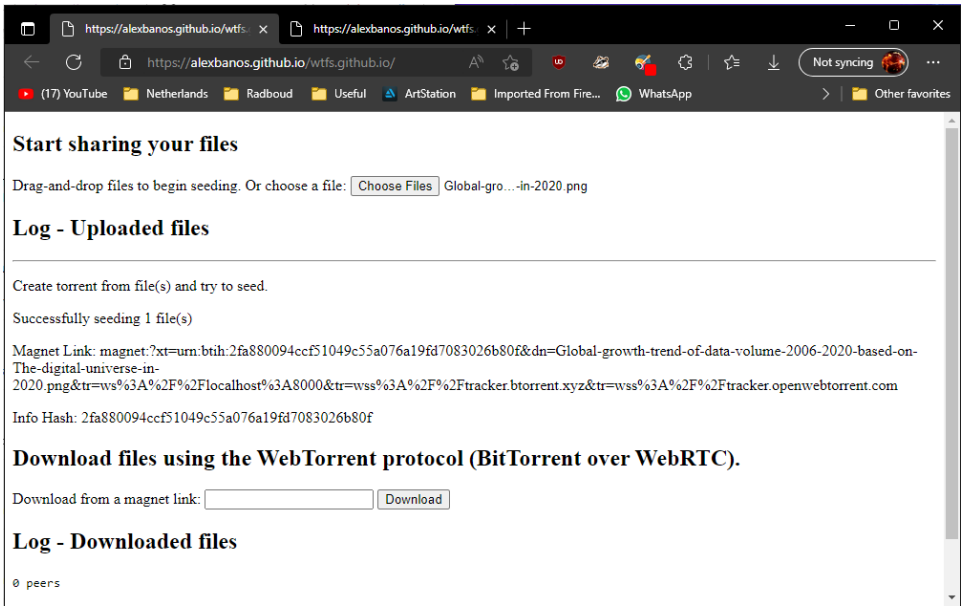


Figure A.11: Select  le, uploaded and seeding

### A.5.2 Download

We download on a di erent window/client to download the le we just uploaded on the other window.

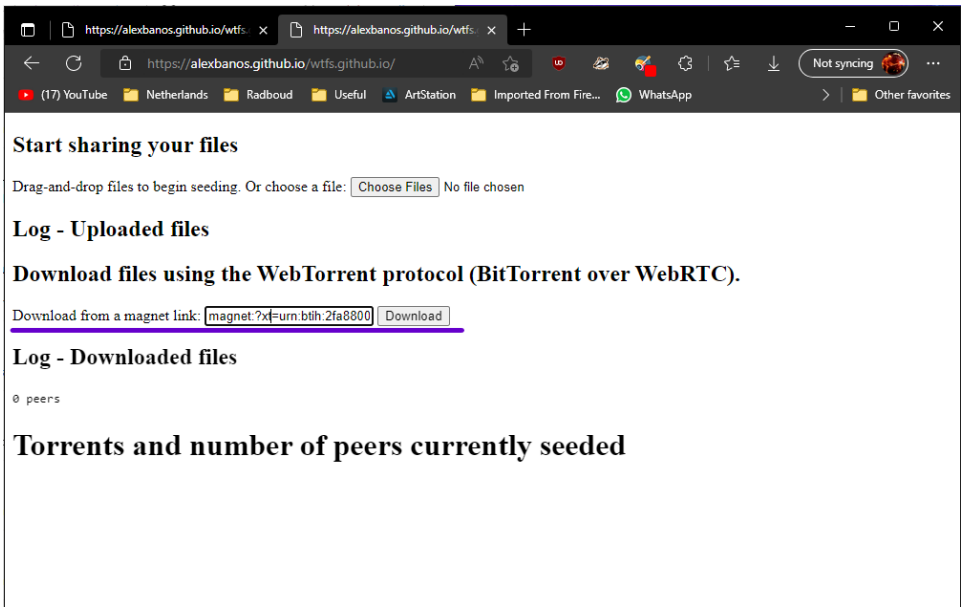


Figure A.12: Paste Magnet Link of torrent we uploaded on the other window