

# **“PEER TO PEER FILE SHARING SYSTEM WITH THE BITTORRENT PROTOCOL”**

*A minor project report,  
submitted in partial fulfillment of the requirement for the award of  
**B.Tech. degree in computer science and engineering***

*by*

**Anjan Srihari (2018BCS-006)**



विश्वजीविनामृतं ज्ञानम्

**ABV INDIAN INSTITUTE OF INFORMATION  
TECHNOLOGY AND MANAGEMENT  
GWALIOR-474 015**

**2020**

## CANDIDATES DECLARATION

I hereby certify that the work, which is being presented in the report, entitled **Peer to Peer File Sharing System using the BitTorrent Protocol** , in partial fulfillment of the requirement for the award of the Degree of **Bachelor of Technology** and submitted to the institution is an authentic record of our own work carried out during the period *June 2020* to *September 2020* under the supervision of **Prof. Anurag Srivastava**. We also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

Date:

Signatures of the Candidates

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date:

Signatures of the Research Supervisors

## ABSTRACT

Two major factors in the QOS of major Internet Connections are Data Rate (Download and Upload Rates in Bytes per Second) and Data Volume (Amount of data that can be downloaded within billing period).

For local-networks sharing an Internet connection via a common router, an efficient and potentially cost saving practice is to minimize the number of redundant downloads into the network.

In order to efficiently utilize this utility this project provides a P2P file sharing system.

P2PFS is a three-module software that sets up a BT compatible tracking server, a Web-interface with which BT specific metainformation can be shared and searched as well as a document database to store those metainfo files.

P2PFS is designed using a Functional Architecture, in using the Programming Language Haskell, for developing a simple, rapid and abstract description of the problem environment, by substituting programs tests with proofs of correctness, and guarantees no runtime errors as long as the system is well defined.

This paper discusses in detail the design, working and performance advantages of P2PFS.

*Keywords:* P2P, fuzzy text searching, BitTorrent Protocol, Document Databases, Functional Programming.

## ACKNOWLEDGEMENTS

I am highly indebted to **Prof. Anurag Srivastava** , for his esteemed mentorship, and allowing me to freely explore and experiment with various ideas in the course of making this project a reality. The leeway I was given went a long way towards helping cultivate an genuine thirst for knowledge and keeping up the motivation to achieve the best possible outcome.

I can genuinely say, that this minor project made me explore many areas of computer science that are new to me, and kindled an interest to further follow up on some of those areas. Moreover, the successful completion of this project has brought with it great satisfaction and more importantly, confidence in my ability to produce more high-quality non-trivial software applications that can make a difference.

As such, I would like to thank this institution for allowing me and my colleagues to pursue a minor project despite the difficult circumstances facing the whole world. I deeply appreciate the efforts of my Professors in mentoring and fairly evaluating our works remotely.

Finally, I would like to thank my family, whose support and understanding helped me confidently commit my time towards this project, and helped provide a laymans point of view that was very valuable.

(Anjan Srihari)

# TABLE OF CONTENTS

<b>ABSTRACT</b>	<b>ii</b>
<b>1 Introduction</b>	<b>2</b>
1.1 The Requirement for File Sharing Systems . . . . .	2
1.2 Potential Software Based Data Distribution Strategies . . . . .	2
1.2.1 Server-Client . . . . .	2
1.2.2 Peer to Peer . . . . .	3
1.3 Indexing and Publishing Data . . . . .	4
1.4 Problem Statement in Brief . . . . .	4
<b>2 Literature Survey</b>	<b>5</b>
2.1 P2P Network Applications . . . . .	5
2.2 BitTorrent Protocol . . . . .	6
2.3 Fuzzy Text Searching . . . . .	6
2.4 Functional Programming and Architecture . . . . .	7
2.5 Full Problem Statement . . . . .	7
2.5.1 Objective : . . . . .	7
<b>3 Requirements Analysis and Specification</b>	<b>9</b>
3.1 Torrent Tracker . . . . .	9
3.1.1 Announce Requirements . . . . .	9
3.1.2 Scrape Requirements . . . . .	10
3.2 Database and Search Module . . . . .	10
3.2.1 Database: . . . . .	10
3.2.2 Indexing: . . . . .	11
3.2.3 Searching: . . . . .	11
3.3 Web Based Frontend and Server . . . . .	11
3.3.1 Search Page: . . . . .	11
3.3.2 Results Page: . . . . .	11
3.3.3 Torrent Description Page: . . . . .	11
3.3.4 Upload Page: . . . . .	12

<i>TABLE OF CONTENTS</i>	1
3.3.5 All pages: . . . . .	12
<b>4 System Design and Methodology</b>	<b>13</b>
4.1 Torrent Tracker . . . . .	13
4.1.1 Methodology . . . . .	13
4.1.2 Design . . . . .	14
4.2 Database Access . . . . .	14
4.2.1 Methodology . . . . .	14
4.2.2 Design . . . . .	15
4.3 Search Module . . . . .	15
4.3.1 Methodology . . . . .	15
4.3.2 Design . . . . .	16
4.4 Web-Server and Frontend . . . . .	16
4.4.1 Methodology . . . . .	16
4.4.2 Design . . . . .	16
<b>5 Implementation Specifics</b>	<b>18</b>
5.1 Module Specific Details . . . . .	18
5.1.1 Torrent Tracker . . . . .	18
5.1.2 Database and Search Module . . . . .	18
5.1.3 Web Server . . . . .	19
5.2 The working project . . . . .	20
<b>6 Test Results and Screenshots of the Project</b>	<b>21</b>
6.1 Test Results . . . . .	31
<b>7 Conclusion and Future Scope</b>	<b>32</b>
7.1 Conclusion . . . . .	32
7.2 Future Scope . . . . .	32

## ABBREVIATIONS

BT	BitTorrent
DB	Database
QOS	Quality of Service
ISP	Internet Service Provider
P2P	Peer-to-Peer
P2PFS	Peer-to-Peer File Sharing System
metainfo	Meta-Information
RR	Round Robin
FSS	File Sharing System

# CHAPTER 1

## Introduction

### 1.1 The Requirement for File Sharing Systems

In the current era of internet and computing, the demand for data has skyrocketed, and as such ISPs charge for their services on the basis of **Data Limits** and **Transfer Speeds**. Typically upon exceeding the Data Limit, the Transfer speeds are throttled.

Logically, we can assert that, an Internet connection is best utilized by accessing the largest possible quantity of **desirable, distinct data**, i.e. try and minimize the amount of redundancy in the data that is fetched over the internet connection.

The common sense approach to resolving this is to utilize the proximity of the users, and distribute data internally thereby minimizing their need to access the internet. Considering we describe the size of a network in terms of the number of users it serves, we can utilize this proximity in different ways.

1. For Small Networks: Physically transfer data using Mass Storage Devices, or short range networking like bluetooth.
2. For Medium to Large Scale Networks: Use a software based data transfer protocol, to enable easy transfer over larger physical distances, using the existing Networking Infrastructure.

### 1.2 Potential Software Based Data Distribution Strategies

#### 1.2.1 Server-Client

This is a very general description of a system that would follow the above definition

1. All data is concentrated at a Central Server.

2. This Central Server is accessible via local network by other computers.
3. Users query the server for the data they require.
4. If it exists they get it from the server.
5. Otherwise, download the file from the internet, and save it in the Central Server for future queries from other users.
6. Users may delete the files, and then recollect them from server at any time.

From this description it is inferred that

1. Central Server must have a large amount of data storage capability, to the Order of 10s, or even 100s of Terabytes.
2. Central Server at worst-case will have a large number of users querying it at the same time.

This leads us to the conclusion that this approach is expensive, to setup, and has a singular point of failure that reduces its robustness and doesn't scale with increase in number of users.

### **1.2.2 Peer to Peer**

A peer can simply defined as a host that is simultaneously both a server as well as a client. This implies, that a peer not only waits to be accessed, but also sends requests to other peers. This is important because of the following

- Peers share their computational resources with each other.
- Due to distributed responsibility, the task being performed has more than one point of failure, therefore allowing the process to continue at large, even with multiple failures.
- Allows the ability to scale the performance and service with the number of peers.

The above are amongst the most ubiquitous properties of any P2P system. In the context of a Data Transfer System, this means that

- Users are Peers.
- Peers upload as well as download data.
- One Peer can upload data to multiple Peers at once.
- There maybe one or more Peers with the same data.



From the above, clearly, there is no requirement for a central data concentration. No server. The amount of data available is the sum of the data each peer offers to the system. Therefore, such a system would scale for free with an increase in number of contributing users.

### 1.3 Indexing and Publishing Data

The previous section discusses two approaches to conducting data transfer, provided that we have sources to communicate with and some description of the data that we require. For a file sharing system to be effective, it should also enable users to find those aforementioned sources.

Quite simply, It should let the user:

- Advertise that they have data to share, and describe the data. This data to describe data is nothing but MetaData.
- Search for data available via the system, made available by other users. The above search would try to match queries against metadata, which in turn uniquely identifies the required data.

To provide the above services, the system must be able to index the data using some criteria, most likely using some form of metadata.

### 1.4 Problem Statement in Brief

The problem statement for this project, with reference to the aforementioned knowledge is, to provide a file distribution system that operates in a P2P approach, along with a file sharing system that allows users to find and/or publish data to the system.

The critical advantages this must provide are

- The setup cost to organization is minimal
- The cost to scale the system in order to accommodate more users is minimal.
- Performance doesn't degrade for a large number of users.
- Existing network infrastructure is optimally utilized.

# CHAPTER 2

## Literature Survey

### 2.1 P2P Network Applications

Since the early 2000s where disruptive software platforms such as napster and gnutella made the case for real world P2P software applications, The domain of P2P Network Applications has been the subject of many studies. The first step towards understanding the concept is being able to properly define the term. The definition utilized in this project was given during the first international conference on P2P computing.[1]

**Definition 1** *P2P Network*

1. *A distributed network architecture may be called a P2P network, if the participants share a part of their own hardware resources. These shared resources are necessary to provide the Service, and content by the network. They are accessible by other peers directly without passing intermediary entities. The participants of such a network are thus resource providers as well as resource consumers.*
2. *Such a P2P architecture may be "pure" or "hybrid" as per the given characteristics*
  - (a) *If a system is P2P according to the previous definition, it is also **pure** if upon the removal of any **single arbitrary** terminal entity the system continues without any loss of service.*
  - (b) *If a system is P2P according to the previous definition, it is **hybrid** if a central entity is necessary to provide **parts of the offered services**.*

Having defined what criteria makes a system P2P, we turn our attention towards common concepts associated with P2P systems, in particular, P2P search and indexing. In P2P systems, the resources, must be identified and queried and searched somehow.

A common approach is to utilize a database that contains the required indexing information that will be queried[2]. Because this database is a centralized entity, the P2P system becomes a Hybrid one.

From the above we deduce, that the P2P file sharing system should ideally have

- A central database for searching and indexing data
- A pure P2P system for the distribution of the data.

Clearly the desired System will be a Hybrid P2P System.

## 2.2 BitTorrent Protocol

The BitTorrent Protocol, is a P2P Communication Protocol. It is a Hybrid P2P Protocol with the following features.[3]

- Peers are Anonymous
- Peers find other Peers using a "Tracker" (A central server)
- Peers communicate with other Peers using "Clients", via TCP
- Content available on System is uniquely identified using MetaData files known as "torrent" files.
- It is capable of sustaining multiple Peer failures.
- It doesn't unfairly snub[3], or overwork any Peer.

Since it's inception in 2001, by Bram Cohen, BitTorrent has gone on to become one of the most commonly used P2P protocols, and as such has a lot of work done on it.

The BT Protocol over the years has been highly standardised[4] and has been very well documented, via BitTorrent Enhancement Proposals or BEPs [5].

Using the BEPs, one can implement from scratch a P2P communication system, that has drop-in compatibility with existing BT-Related Software utilities.

## 2.3 Fuzzy Text Searching

As per the BT protocol, content is uniquely identified by its MetaData, and the MetaData is further uniquely identified by their SHA-1 Hashes (Also known as InfoHashes). Additionally, this MetaData is stored in files known as "torrent" files.[4]

Using InfoHash as Document IDs, and the text based fields of the torrent files as text indices, we can generate a search and indexing model for torrents using Standard Information Retrieval Methods, that have had extensive study and have been documented

into many standard pieces of literature, one of them being 'Introduction to Information Retrieval'[6].

Humans use search engines, using words in human languages. These words may be misspelt, inaccurate or incomplete, but the Human user must still get some results from the search engine, by predicting the words that may match the misspelt query given, i.e. fuzzy partial matches.

One of the methods used to match natural languages is using N-Grams, which represents words using the patterns that can be found in them. These patterns can be used for lookups in search engines, as Sekine did in his Ngram Search Engine[7]

## 2.4 Functional Programming and Architecture

Functional programming is a paradigm where programs are constructed by applying and composing functions. Here, function definitions are trees of expressions that each return values, as opposed to a sequence of steps that will change some internal state. Additionally, the functions are treated as first class citizens, and allow partial application[8].

Functional Programming most importantly supports the definition of abstract, and algebraic data-types, using which modelling entities is very easy and highly abstracted. Due to the roots of Functional Programming being in Category Theory and Lambda Calculus, programs rather than being tested, can be rigorously proven to be accurate and consistent, thereby avoiding time spent handling run-time errors, as stated in the seminal article on the topic by John Hughes, entitled "Why Functional Programming Matters". [9]

## 2.5 Full Problem Statement

With all of the aforementioned context, here comes the exact no holds barred problem statement.

### 2.5.1 Objective :

To create a P2P file sharing system, using the BitTorrent Protocol for File distribution and a Fuzzy Text Search Engine and a Document Database containing Torrent MetaData for File Search and Indexing, via a Web-Based Frontend, using a Functional Reactive Architecture.

The File Sharing System consists of the following modules

- A Torrent Tracker, compatible with all existing third party torrent clients. (For File distribution)

- A Document Database with a Fuzzy Search Module that contains Torrent Meta-Data. (For file searching and indexing)
- A Web-Based frontend to search for and upload for required BT Protocol specific MetaData.

# CHAPTER 3

## Requirements Analysis and Specification

We can divide the requirements from the system as per the three particular modules that are present in this project.

### 3.1 Torrent Tracker

This module has the following requirements, as per the BT Protocol Specification[4]. Upon adhering with these requirements we additionally fulfill the non-functional requirement that the file sharing system must be compatible with any generic third-party Torrent Client, such as BitTorrent, UTorrent, deluge etc.

#### 3.1.1 Announce Requirements

The Tracker Must Accept GET Requests as shown in 3.1, and return a PlainText BEncoded Response with the parameters as shown in 3.2

info_hash	SHA-1 Hash of Info Key in MetaInfo
peer_id	urlencoded 20 Byte String as ID for client
port	Port Number of Client
uploaded	Total Amount uploaded by Client
downloaded	Total Amount downloaded by Client
left	Number of Bytes pending to download
event	(optional) one of (started, stopped, finished)
ip	(optional) rfc3513 IP address of Client
numwant	(optional) Number of Peers Client wants

Table 3.1: Announce Request Parameters

failure_reason	(optional, if present no other fields) error message
warning message	self explanatory
interval	number of seconds to wait till next request
min interval	(optional) minimum number of seconds client should wait between requests
tracker id	string that client returns on next announce
complete	no of seeders
incomplete	no of non-seeders
peers	list of dictionaries containing (PeerID, IP Address of peer, Port of peer)

Table 3.2: Announce Response Fields

### 3.1.2 Scrape Requirements

Tracker respond to GET Requests containing one or more "info hash" fields with Plain-Text BEncoded response, which is a dictionary mapping each "info hash" to the scrape data associated with it. Scrape Data has the structure as shown in 3.1.2

complete	number of peers with entire file
downloaded	number of times tracker has recieved "complete" signal
incomplete	number of non-seeders
name	(optional) name of torrent as specified in metainfo

Table 3.3: Scrape Response Dictionary

## 3.2 Database and Search Module

This module stores and indexes the **torrent metadata file**. This is the smallest unit that must be distributed first in order to conduct larger file transfer. metadata files may be of a few KB as opposed to the actual data that may be any size upto several GB. It has the following requirements.

### 3.2.1 Database:

The database must be an unstructured Document database, to accomodate the optional nature of the fields in a torrent file.

### **3.2.2 Indexing:**

All the documents in the database must be indexed using their unique infohashes and the concatenation of all of their text based fields.

### **3.2.3 Searching:**

Given a textual query, database must return a list of documents that may satisfy that query, sorted in decreasing order of relevance. The text query contains words that may be present in the text index of the document. The words in the query may be misspelt and not match the exact equivalent words found in the documents.

## **3.3 Web Based Frontend and Server**

This module provides a user accessible, web-based frontend to the user, that can be viewed in a web-browser. It has the following requirements.

### **3.3.1 Search Page:**

Search Page must contain a text-box that accepts text queries and redirects to the Results Page.

### **3.3.2 Results Page:**

Must contain several result boxes, that may match the query given in the Search Page. Result Boxes must contain the following:

- Name of the torrent. This must be clickable and open full torrent description page.
- Name of uploader, if not specified Anonymous is shown.
- Size of the torrent.
- Excerpt of the Description of the torrent

### **3.3.3 Torrent Description Page:**

Must contain human readable meta information about the data, which is

- Name of the Torrent
- Name of the Uploader



- Full description of the torrent (Written by original creator)
- List of files that the torrent contains and their sizes. (Directory Structure must be displayed as well.)
- Clickable download button for getting the torrent file, that is automatically be opened by the torrent client.

#### **3.3.4 Upload Page:**

Must contain a quick guide on how to use the P2P file sharing system to upload data and a file selector to select a torrent file and upload it.

#### **3.3.5 All pages:**

Must contain shortcuts and links to navigate the entire site without requiring to type any URLs

# CHAPTER 4

## System Design and Methodology

This project utilises a Functional Reactive Programming, that is implemented using a Pure Functional Programming language which is Haskell.[10] [11]

**Definition 2** *Functional Reactive Programming[12] Functional Reactive Programming seeks to model software on the basis of Events and Behaviours. It is designed asynchronously, to react to events, using some defined behaviour. This approach is highly declarative, and highly abstract, allowing us to easily model operational environments.*

**Definition 3** *Pure Functional Programming In "pure" functional programming, there are two major characteristics*

- *Functions are first class entities, that can be operated upon and created dynamically. These are known in theory as Higher-Order Functions.*
- *Functions are **Black Boxes**, whose sole purpose is to take inputs and produce outputs. They must not perform any side-effects. Side-effects are computations outside of the scope of the function.*

Note that all the diagrams below represent a highly simplified view of the system to abstract away all the nitty-gritty details.

### 4.1 Torrent Tracker

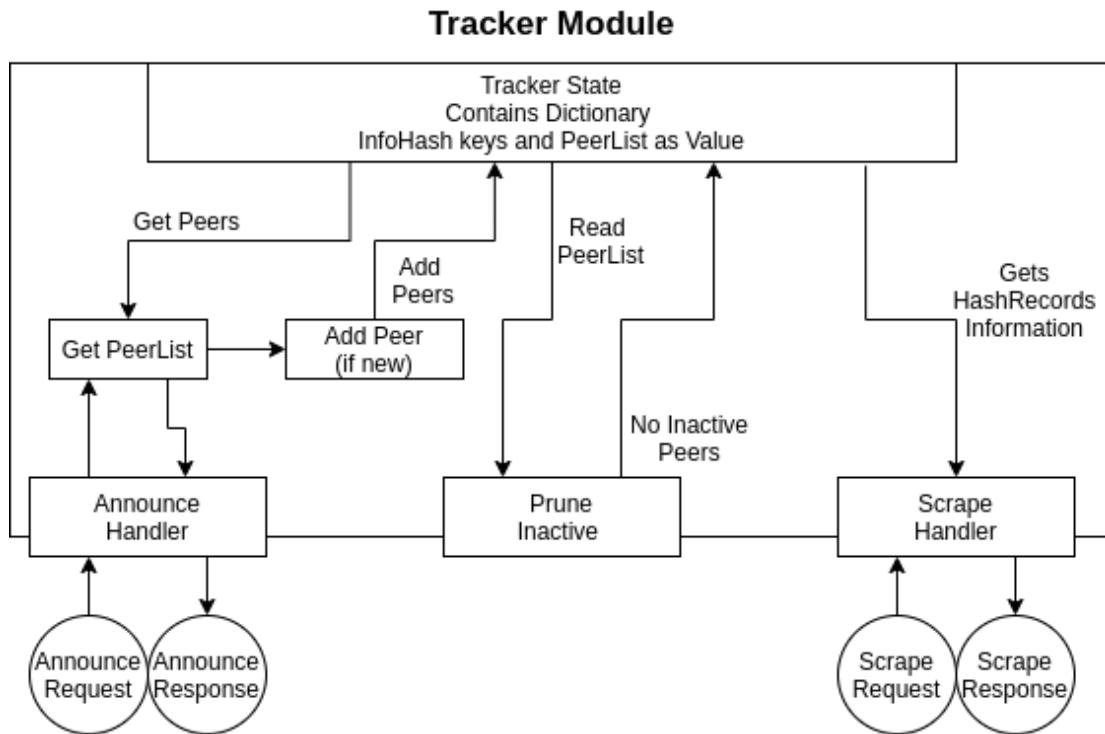
#### 4.1.1 Methodology

A server by nature is to be accessed asynchronously and concurrently by several clients at once. In particular, the torrent tracker will be queried asynchronously by torrent clients, who will issue GET requests and expect a response.

From the reception of a request upto the generation of a response, the computations can be represented as a simple sequence of functions that processed the data and produced outputs, that became inputs for other functions, further behind in line.

By utilizing a pure functional programming language, we are able to easily handle concurrent accesses, as the lack of side effects greatly minimizes the possibility of race conditions and inconsistent states.

### 4.1.2 Design



## 4.2 Database Access

### 4.2.1 Methodology

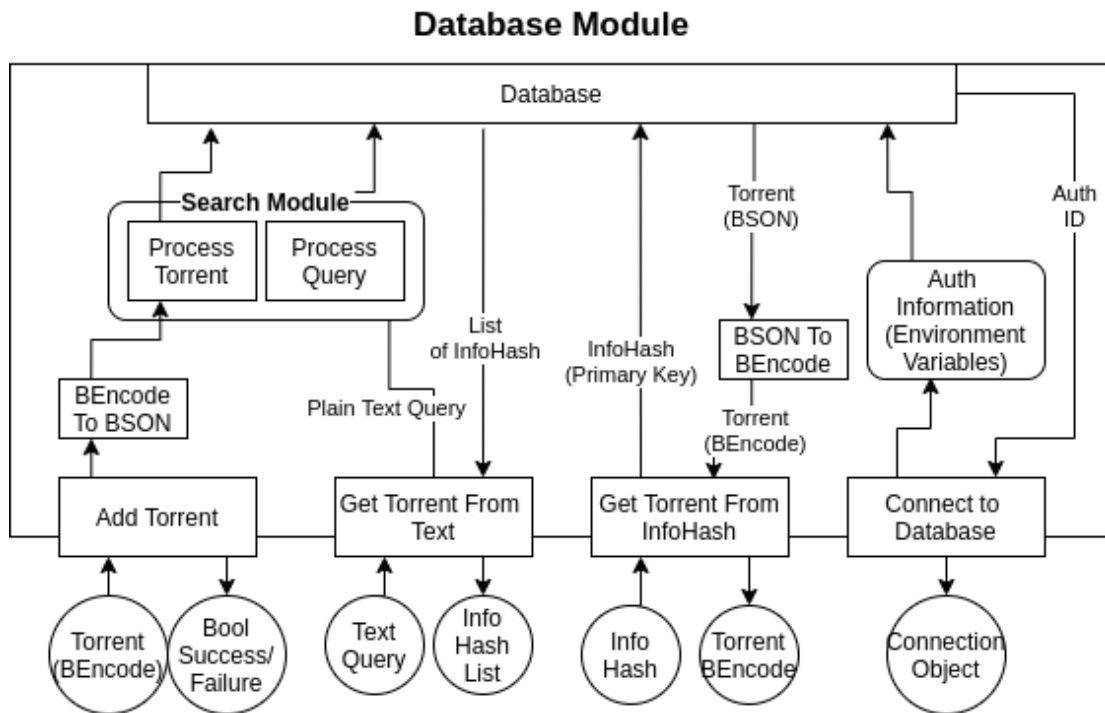
As per requirements, the database should have to efficiently store torrent metadata, which may not always have the same fields.

Therefore, it was clearly necessary to utilize an Unstructured Document Database. Furthermore, due to the highly specific usecase of this database module, we are able to abstract the full functioning and implementation details of uploading and fetching values behind wrapper functions.

In this project, due to the specifications of the BT Protocol, the encoding of the documents being uploaded into the database isn't recognizable by the database, and vice-versa, therefore, before the database access modules, there exists a translation sub-module, which converts queries to the database from BT specific to database specific encoding, and converts results from the database from database specific to BT specific encoding.

This module is nothing but a library. It presents an interface of functions, while obscuring implementation details to the Web-Server module, which utilizes them.

## 4.2.2 Design



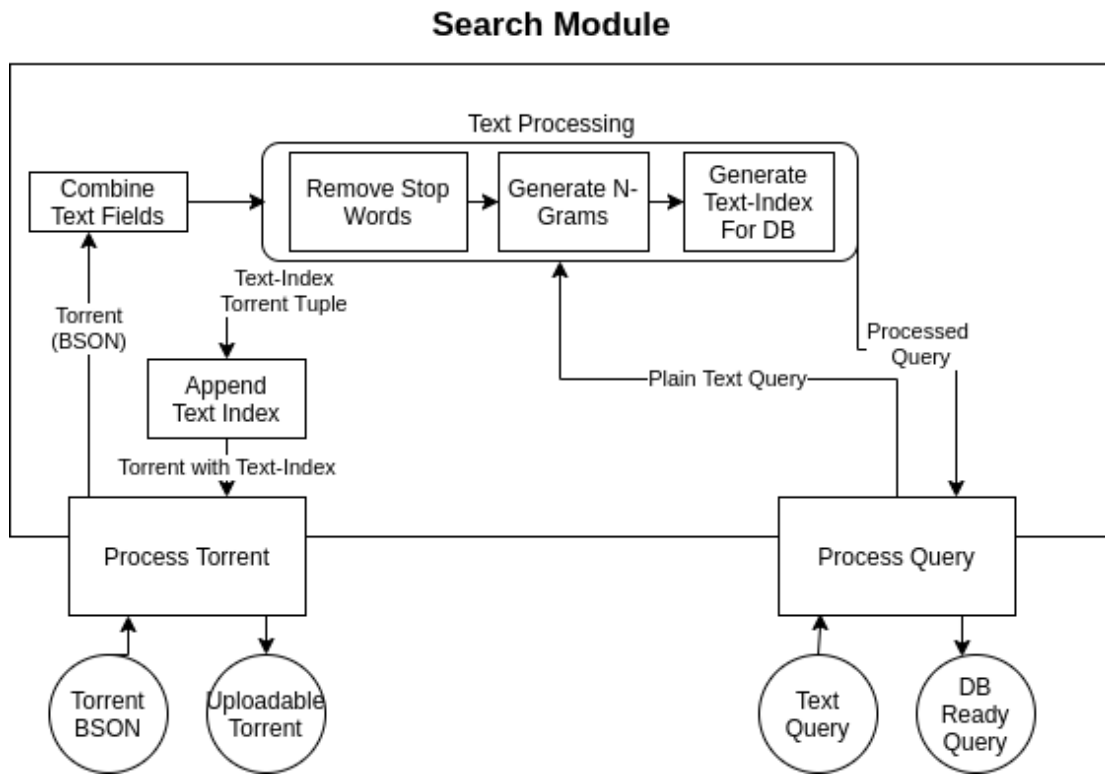
## 4.3 Search Module

### 4.3.1 Methodology

This module logically lies between the Web-server and the Database module in the sequence of execution. In particular, for torrent file uploads, it processes the torrent file, and appends searchable text-indices that are utilized in order to search for torrent files using text.

For searches, it receives text queries from the server module, which it then processes and cleans up, and constructs a query for the database module, that will return a list of results which is passed along as is to the server module.

This module, in effect builds onto the database the ability to perform fuzzy text searches.



### 4.3.2 Design

## 4.4 Web-Server and Frontend

### 4.4.1 Methodology

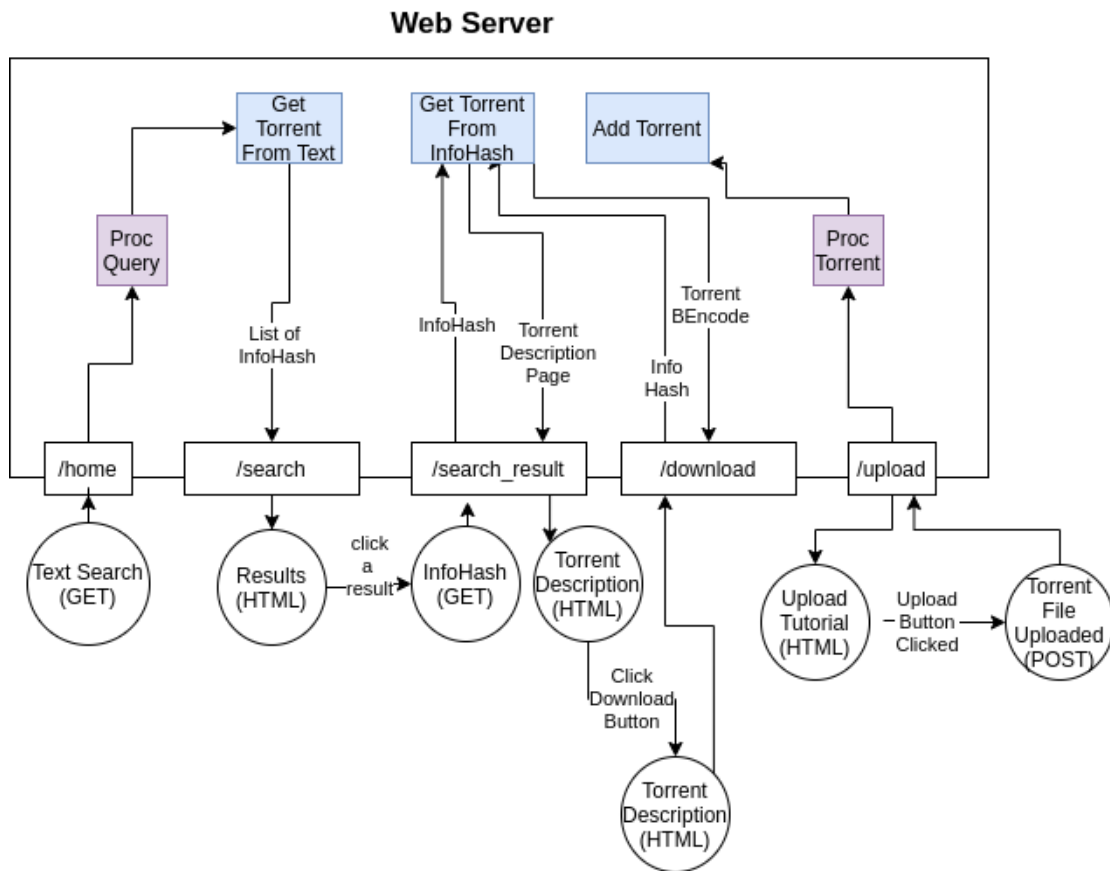
The web-server has been implemented as a REST API, with multiple routes, and a handler for each route. Each handler uses functions exposed by the database and the search modules in order to generate responses.

In between receiving the requests and sending them on to their handlers, there are functions present to deserialize them into internal representation, as well as functions to serialize the internal representation into network compatible data.

Here, all the webpages are dynamically generated from templating engines using the return values from the backend functions, which are from the database and search modules.

### 4.4.2 Design

Blue boxes are functions from Database module, Purple boxes are functions from Search Module



# CHAPTER 5

## Implementation Specifics

### 5.1 Module Specific Details

#### 5.1.1 Torrent Tracker

For the Torrent Tracker, HTTP as well as UDP servers were implemented, using the Haskell network library, along with Servant, in order to easily define the API.

The state information for the torrent tracker is all being stored in memory. Due to this, upon failure of the system due to external factors, Upon restarting, the tracker will be very briefly be, non-functional, however with subsequent announces from the clients, the state is very quickly reconstructed.

The advantage of this is that the system becomes fault tolerant, and remains fast as the state is in memory. The memory consumption is not as much as one would expect, as the state information for each torrent is at most a few kilobytes, with even 4GB of ram, the number of torrents that can be handled by the torrent is in the ten thousands.

#### 5.1.2 Database and Search Module

For the database, we utilized MongoDB[13], for the following two reasons

- It is unstructured and document based by default.
- It supports text indexed search, by default, with the caveat that it cannot provide fuzziness, or the ability to make partial matches.

In order to build on the support for Fuzzy and Partial Seaching onto MongoDB, we utilized N-Grams. An N-Gram is a contiguous chunk of the string of length N, for example, the 3-grams of "hello" are ["hel", "ell", "llo"].

With the above context, the exact steps taken in Pre-Processing of any document are

- First all text fields are broken down into words, then all these lists of words are concatenated with each other to produce a single list of words that contains all the words that can be found in the document.
- This list of words is passed through a stop word filter, to remove very common words that don't describe the document such as "this", "that", "furthermore" etc. We will be left with a list of keywords.
- Now, we generate the 3-Grams, 4-Grams and 5-Grams for each word, and concatenate them with each other. This produces for each word a list of N-Grams.
- Now that we have a list of lists, which is a list of the N-Grams Lists, we must flatten this into a single list that contains all the N-Grams contained in the whole document.

The above pre-processing is applied to torrents before uploading, to generate a list of n-grams that acts as a text-index for it, as well as for queries for searching.

MongoDB by default for text searching takes a specified text-index, and analyses all documents that are uploaded to that collection by running a simple TF-IDF algorithm on all of them to be able to search for documents using exact keywords. However, since we are specifying the list of N-grams as the text index, and then further searching using n-grams of the user supplied text query, MongoDB will infact end up searching and indexing for us, all the patterns that can be found in the text. And because the same pattern may be found in several documents, or rather in several words, multiple number of times.

The reason this method works to build in fuzziness, is because even for misspelt words, in order to identify them as misspelt, there must be some patterns of the correct word that must be present in it, for example lets consider "Google" and "Gogle".

N-Grams of Google : ["goo", "oog", "ogl", "gle", "goog", "oogl", "ogle", "googl", "oogle"]

N-Grams of Gogle : ["gog", "ogl", "gle", "gogl", "ogle", "gogle"]

Clearly between the two list of N-grams, we are able to see many matches, with which using similarity indices, we can predict that Gogle must have meant google.

### 5.1.3 Web Server

The web server was built using Servant[14], in order to easily model the API in a typesafe manner that will only allow by default correctly formatted requests and prevent run-time errors.

This is the front-facing module. It calls and references the database module, in order to present its functions.



Here, all the web-pages are dynamically generated from Blaze HTML [15] templates parametrized by the return values from the internal database and search functions.

This Servant web application is being served by the Warp web-server [16], as it has a very high performance, and scales well with many concurrent accesses.

## 5.2 The working project

The final working project can be found on <https://github.com/beaszt-nix/Minor-Project>. It contains the pre-requisites as well as the steps required to deploy this project, along with some additional implementation specific details about the project.

It contains three executables

- `torrdb-init`: This sets up the existing MongoDB connection to work with this project
- `torrent-db`: This establishes the connection to the database, and also activates the web-based front-end, accessible on `localhost:8080`.
- `hask-tracker`: This starts the torrent tracker with which the actual file distribution mechanism is facilitated.

This project is compiled, therefore it provides additional security, as we only host the compiled executable binaries on the server.

The ideal approach to maintaining and updating this project would be to compile off server, and simply upload and deploy the executable binary onto the server.

## CHAPTER 6

# Test Results and Screenshots of the Project

In order to show how the project works, let's work our way through sharing a file, and then downloading it on another device. Along the way, we shall discuss the numbers and performance achieved.

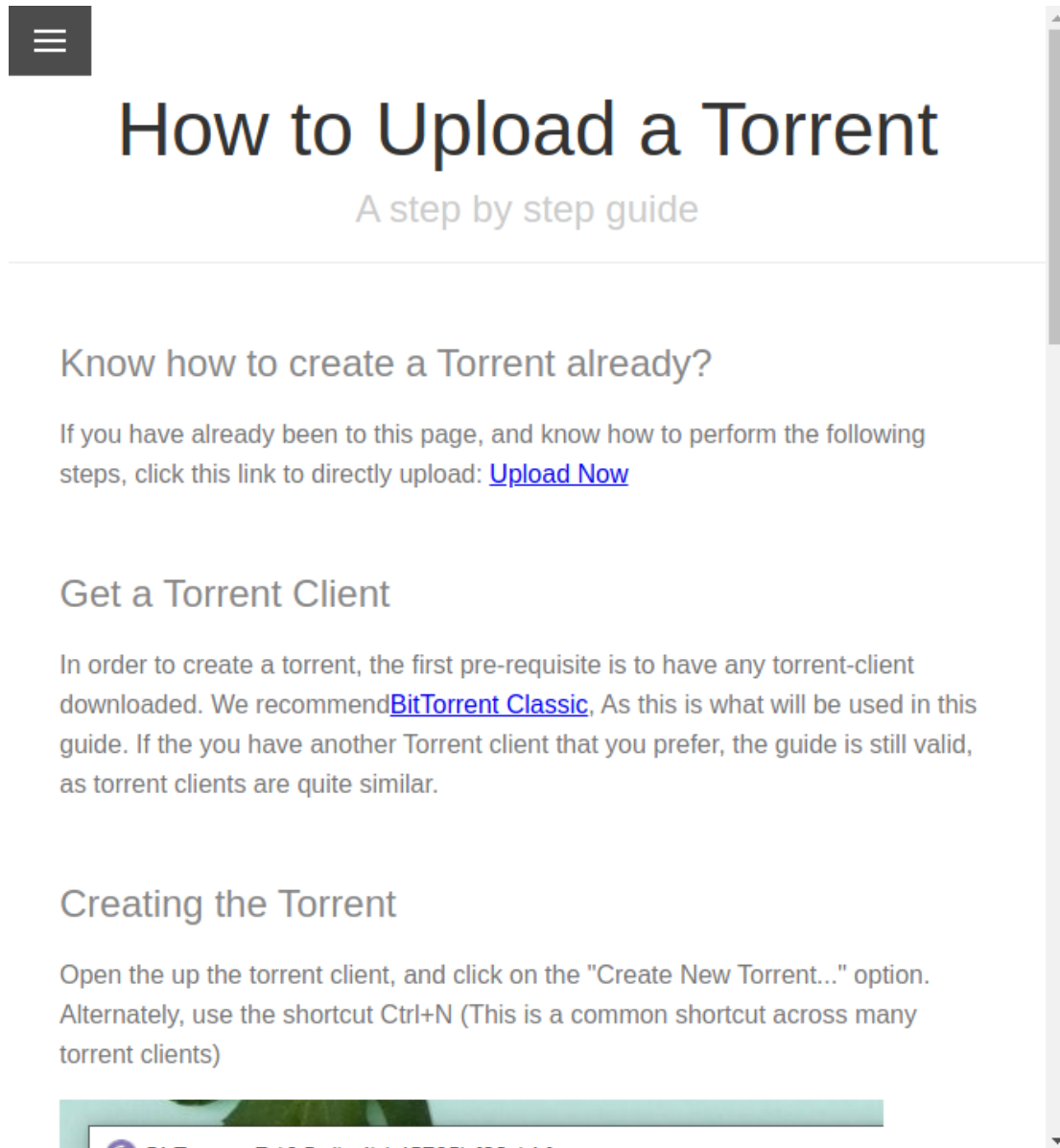
First launching the modules via the binaries.

A terminal window with a dark background and a faint image of a lighthouse. The prompt is [anjan@anjan-latitude3400 ~]. The first command is hask-tracker, which outputs "Starting HTTP Tracker", "Started UDP Tracker", "Binding to 0.0.0.0:6969", and "Binding to :::6970". The second command is torrent-db, which is entered but has no output yet.

```
[anjan@anjan-latitude3400 ~]$ hask-tracker
"Starting HTTP Tracker"
"Started UDP Tracker"
Binding to 0.0.0.0:6969
Binding to :::6970

[anjan@anjan-latitude3400 ~]$ torrent-db
```

Open the upload page, located at "server\_addr/uploadTut" and follow the tutorial given to upload a torrent



The screenshot shows a web page with a dark sidebar on the left containing a hamburger menu icon. The main content area has a large title "How to Upload a Torrent" in a dark serif font, followed by a subtitle "A step by step guide" in a smaller, lighter font. Below this is a section titled "Know how to create a Torrent already?" with a paragraph of text and a blue link "Upload Now". The next section is "Get a Torrent Client", followed by a paragraph of text. The final section is "Creating the Torrent", with a paragraph of text. At the bottom, there is a partial view of a screenshot showing a BitTorrent interface.

# How to Upload a Torrent

## A step by step guide

### Know how to create a Torrent already?

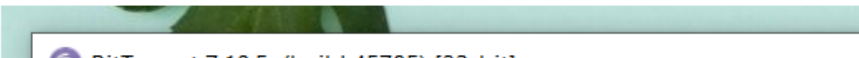
If you have already been to this page, and know how to perform the following steps, click this link to directly upload: [Upload Now](#)

### Get a Torrent Client

In order to create a torrent, the first pre-requisite is to have any torrent-client downloaded. We recommend [BitTorrent Classic](#), As this is what will be used in this guide. If the you have another Torrent client that you prefer, the guide is still valid, as torrent clients are quite similar.

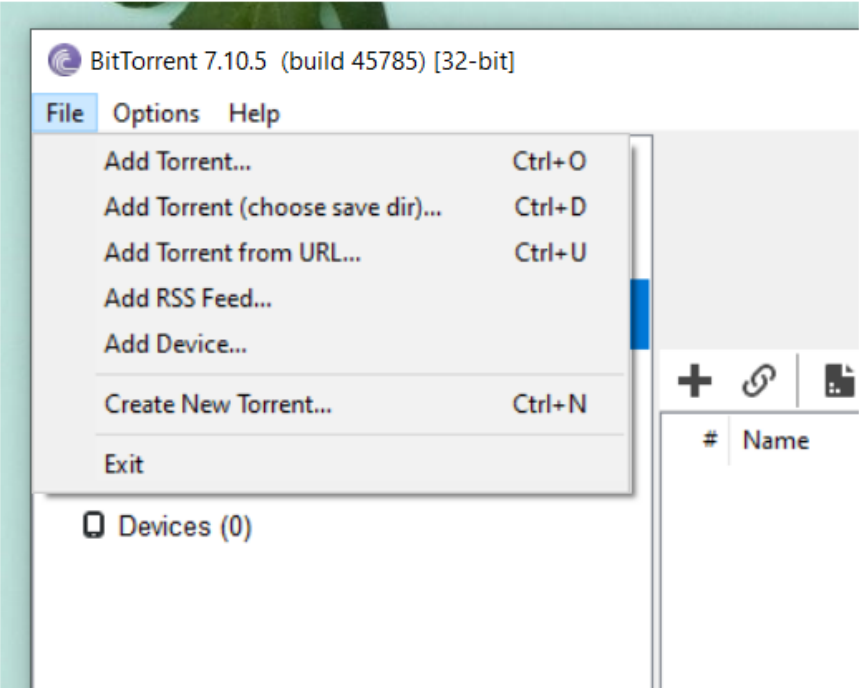
### Creating the Torrent

Open the up the torrent client, and click on the "Create New Torrent..." option. Alternately, use the shortcut Ctrl+N (This is a common shortcut across many torrent clients)

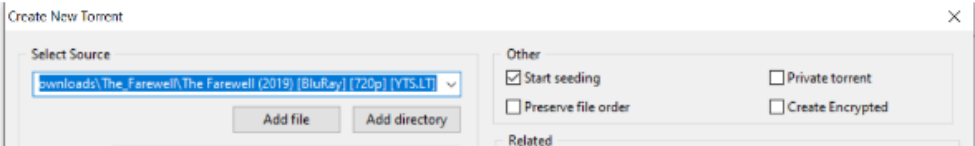


## Creating the Torrent

Open the up the torrent client, and click on the "Create New Torrent..." option. Alternately, use the shortcut Ctrl+N (This is a common shortcut across many torrent clients)

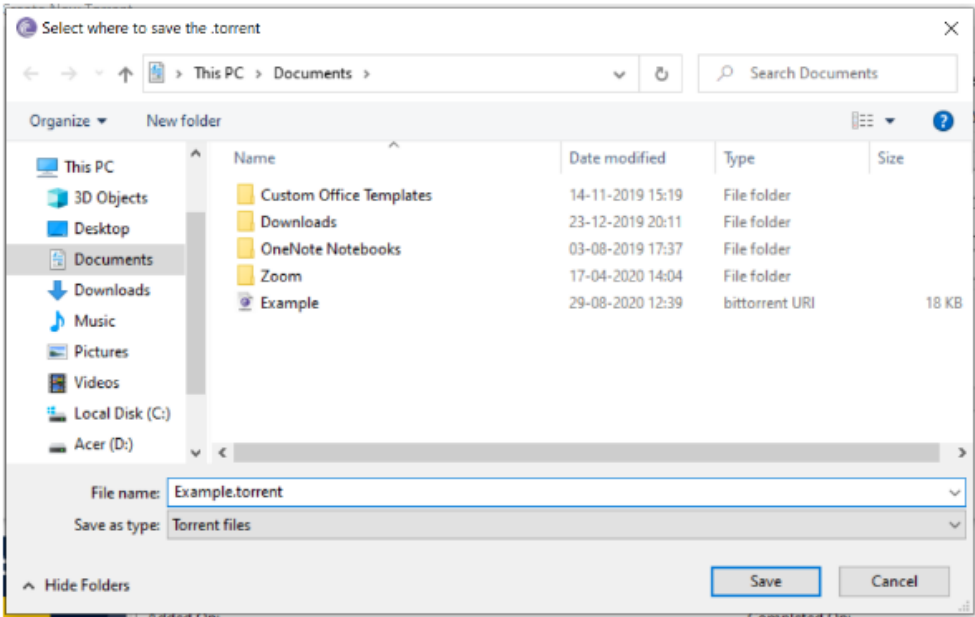


Once this is done, the following window pops up



Select the torrent file and click upload.

become the title for its listing in the P2P file sharing system, therefore, ensure that the name is relevant to the data and contains clear keywords. This makes it easier for another user to find this torrent.



### Upload The Torrent

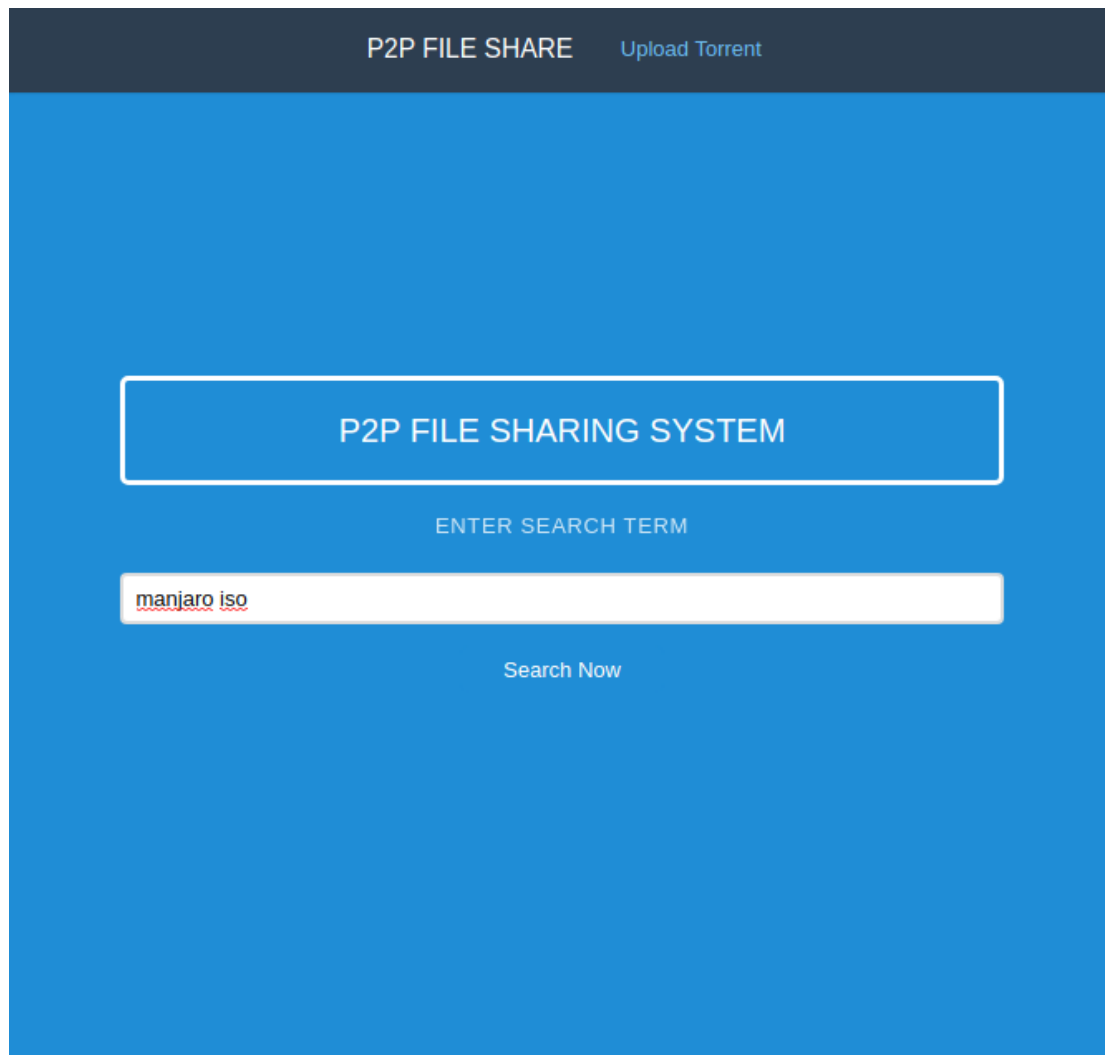
Select File

Choose file

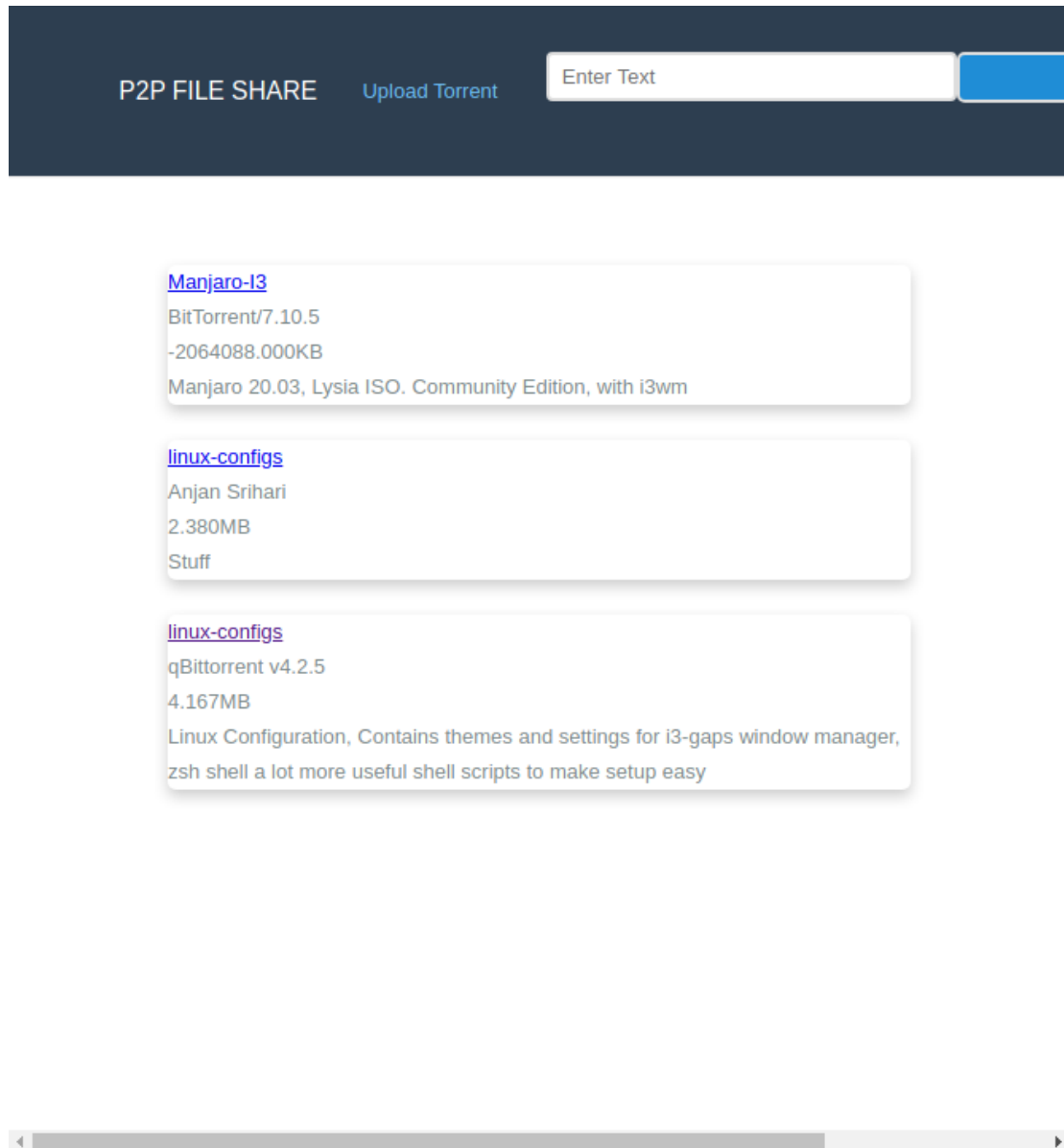
No file chosen

Upload

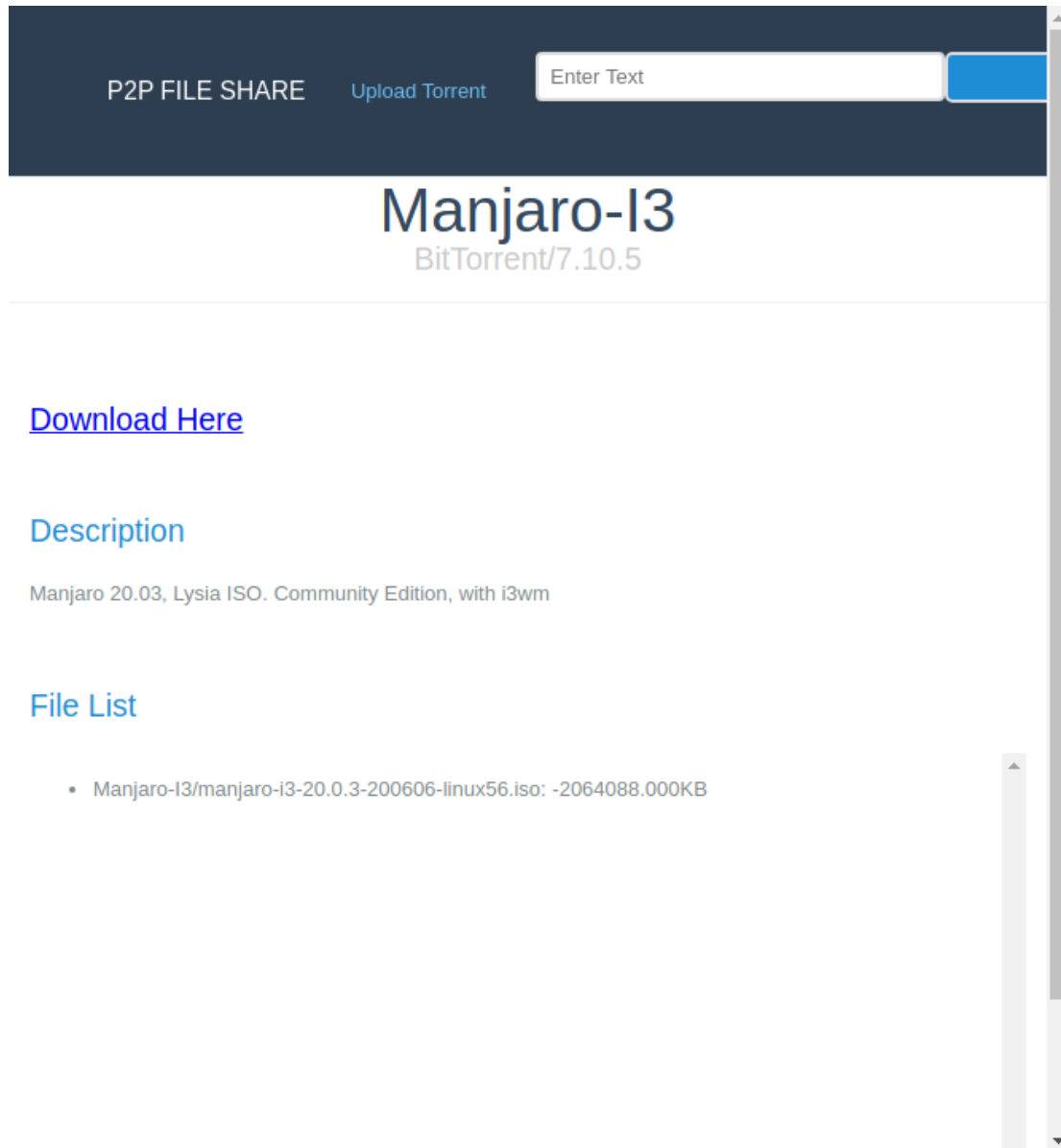
On a different system on the same network, visit the search page, (located at "server\_addr/home"), and enter the search term



The user is now presented with a number of results

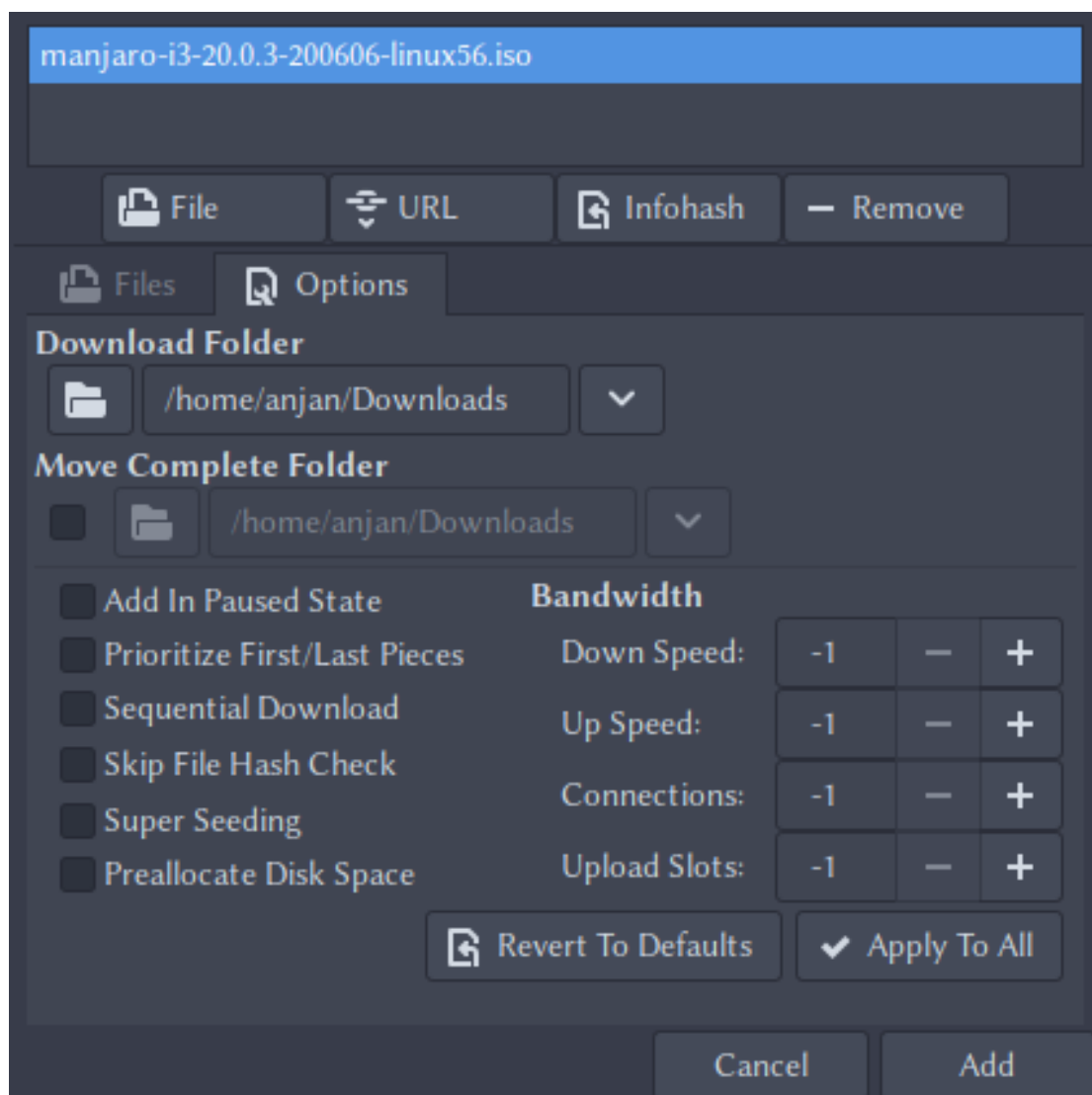
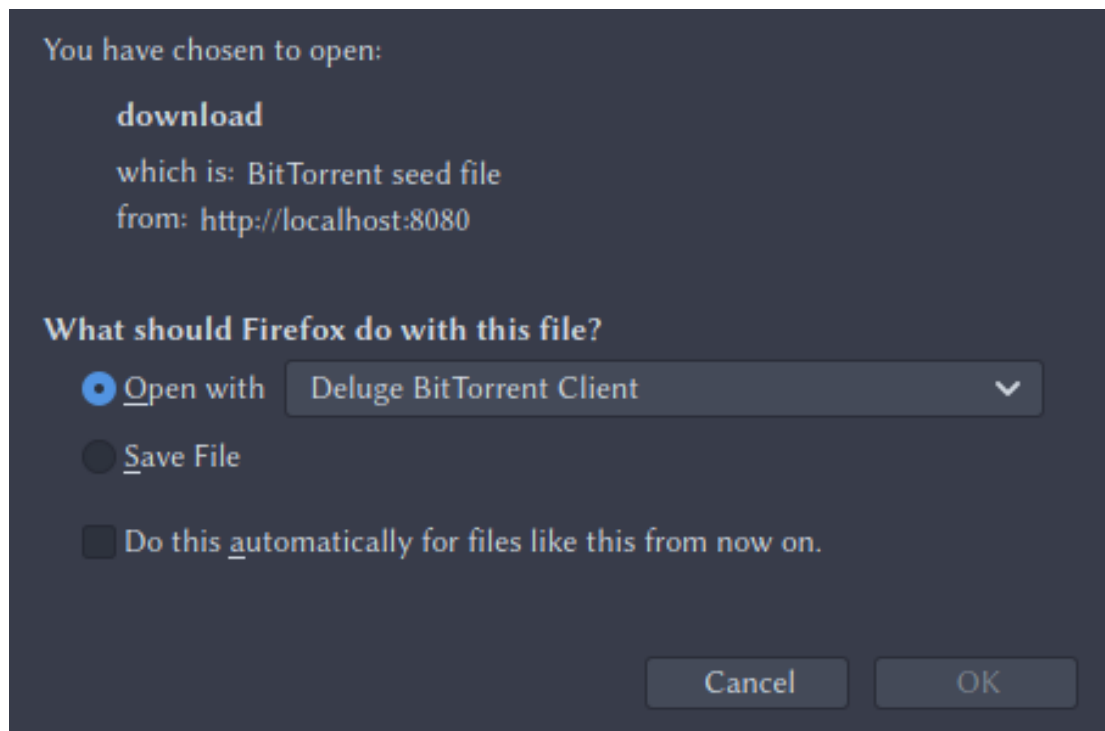


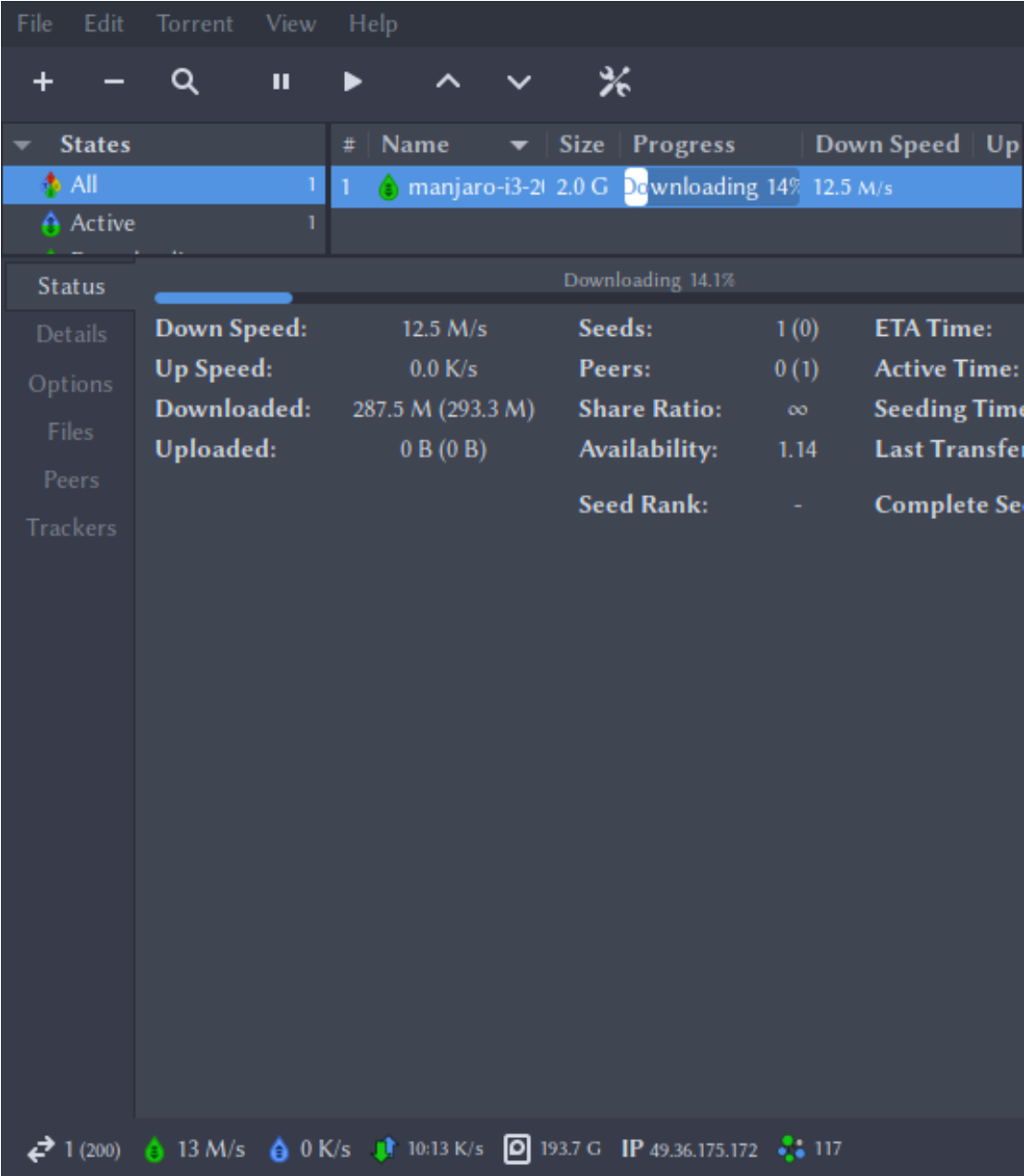
Upon clicking on the desired result, the torrent description page is opened, and after seeing the details, click the download button, which automatically asks us if torrent client must be launched.



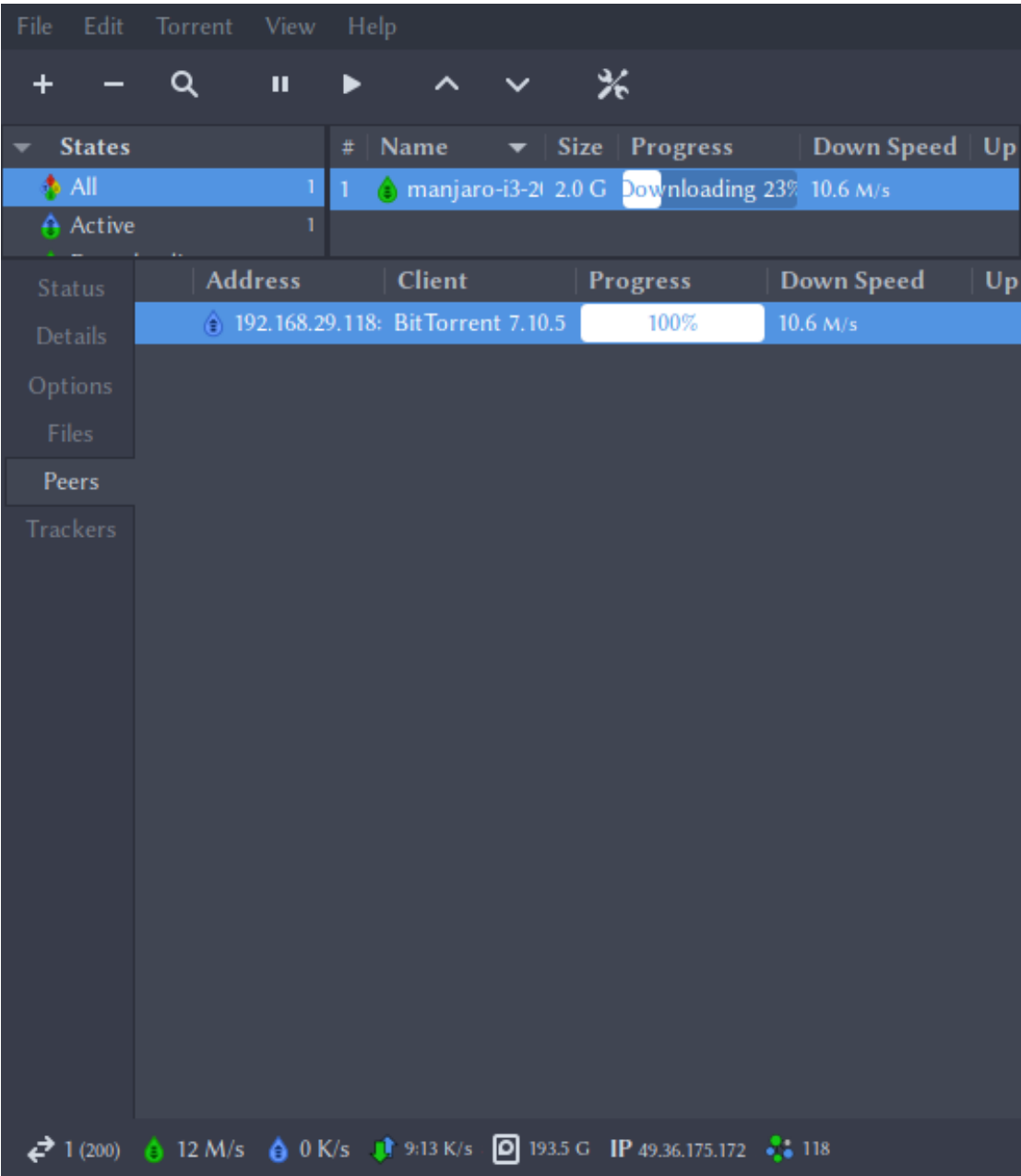
In the torrent-client, we are able to choose where to save the file, and clearly the torrent has started.







We can see here, that the Peer from whom the file is being downloaded is on the local network, by looking at the IP. (Note that the tracker IP in this case is 192.168.29.217)



## 6.1 Test Results

In the above set of pictures, we transferred a 2.3GB file, an ISO of the linux distribution Manjaro. It was uploaded from one device, and downloaded at another, both sharing a common wifi network.

Clearly, as we can see in the images, this system fully saturates the capacity of the local network infrastructure, coming in at 2.3GB transferred in a time period of 4 minutes and 16 seconds, giving an average transfer rate of approximately 9 MB/s.

# **CHAPTER 7**

## **Conclusion and Future Scope**

### **7.1 Conclusion**

In conclusion, from start to finish over a period of approximately 2 Months, this project implements from start to finish, a working file sharing system that functions over the BT Protocol, and allows us to share data over it.

The objective of high-speed data transfer within a local network has been achieved, and a simple ready to use web-based frontend has also been created. Additionally, and rather more importantly, the objective of easily deploying this system on any local network has also been achieved.

This project utilized theory and technology from various disciplines of computer science, and bringing it to fruition was an enlightening experience.

### **7.2 Future Scope**

As Mentioned before, in the Literature Survey of this report, the BT protocol has had several extensions made to it via the BEPs [5].

This project so far has implemented the most commonly used extensions and opted out of implementing others.

In the future, it is possible to implement more extensions as specified in the BEPs to achieve features like P2P multi-casting, real-time streaming etc.

# REFERENCES

- [1] R. Schollmeier, “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications,” 09 2001, pp. 101 – 102.
- [2] J. Risson and T. Moors, “Survey of research towards robust peer-to-peer networks: Search methods,” *Computer Networks*, vol. 50, pp. 3485–3521, 12 2006.
- [3] B. Cohen, “Incentives build robustness in bittorrent,” *Workshop on Economics of Peer to Peer systems*, vol. 6, 06 2003.
- [4] —, *The BitTorrent Protocol Specification: BEP\_0003*, [https://www.bittorrent.org/beps/bep\\_0003.html](https://www.bittorrent.org/beps/bep_0003.html), Jan 2008.
- [5] D. Harrison, *BitTorrent Enhancement Proposals: BEP\_0001*, 01 2008, [https://www.bittorrent.org/beps/bep\\_0001.html](https://www.bittorrent.org/beps/bep_0001.html).
- [6] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 01 2008, vol. 13.
- [7] S. Sekine, “Ngram search engine,” 10 2009.
- [8] R. Bird and P. Wadler, *Introduction to Functional Programming*, 01 1988.
- [9] J. Hughes, “Why functional programming matters,” *The Computer Journal*, vol. 32, pp. 98–107, 02 1989.
- [10] D. Harrison, *Haskell Programming Language*, <https://www.haskell.org/>.
- [11] A. Veen, “Dataflow machine architecture,” *ACM Comput. Surv.*, vol. 18, pp. 365–396, 12 1986.
- [12] Z. Wan and P. Hudak, “Functional reactive programming from first principles,” *ACM SIGPLAN Notices*, vol. 35, 02 2001.
- [13] *MongoDB*, <https://www.mongodb.com>.
- [14] *servant-server: hackage*, <https://hackage.haskell.org/package/servant-server>.
- [15] *warp server: hackage*, <https://hackage.haskell.org/package/blaze-html>.

- [16] *warp server*: *hackage*, <https://hackage.haskell.org/package/warp>.