Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

BitThief QoL

Bachelor Thesis

Markus Hauptner

markuhau@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georg Bachmeier, Michael König
Prof. Dr. Roger Wattenhofer

September 14, 2017

Acknowledgements

I want to express my gratitude towards my supervisors Georg Bachmeier and Michael König for providing ideas, important advice, and helpful feedback throughout the project.

Abstract

In this thesis, we improve the QoL (Quality of Life) functionality of BitThief, a free-riding BitTorrent client. The main goal is the creation of an easy-to-use web interface to enable users remote access to BitThief from different devices. To achieve this, the client had to be extended by web-server functionality and a web frontend had to be designed. Furthermore, the already existing code had to be altered to improve modularity and ease extensibility.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Motivation . . . 1
1.2 Related Work . . . 1
1.3 Goals . . . 1

2 Methods 2

2.1 BitTorrent . . . 2
2.2 BitThief . . . 2
2.3 Jetty . . . 2
2.4 Web Technologies . . . 3

3 Implementation 4

3.1 Architecture . . . 4
3.2 Communication . . . 4
3.3 Backend Implementation . . . 5
3.3.1 Predefined Handlers . . . 5
3.3.2 Newly Created Handlers . . . 6
3.4 Frontend Design and Implementation . . . 7
3.4.1 Design . . . 7
3.4.2 Implementation . . . 7
3.5 Code Structure . . . 7

4 Future Work 14

Bibliography 15

Chapter 1

Introduction

1.1 Motivation

The BitTorrent network is big. With an ever-growing number of users and available files in the network, having a Torrent client running in the background could be of good use, for example for automating downloads guided by machine learning algorithms or downloading on a separate device for maintaining a media server. The main motivation for this particular project was making the usage of BitThief more easy and attractive for potential users.

1.2 Related Work

As this thesis concerns the BitThief BitTorrent client, previous publications about it might be of interest to the reader. *Free Riding BitTorrent is Cheap* by Locher et al. [1] describes the basics of BitThief and explains the motivation for building such a free riding BitTorrent client.

1.3 Goals

The main goal of this thesis is the creation of a web-interface for BitThief. In order to achieve this, the code of the program will be altered to improve the separation of user-interface and functionality. The web-interface should support the general features of the old interface, like starting downloads via magnet link, changing settings, like the maximum download speed, and observing the status of current downloads.

Chapter 2

Methods

2.1 BitTorrent

BitTorrent is a peer-to-peer filesharing protocol that utilizes the download and upload capacities of each network member. If a user wants to download a file provided in the BitTorrent network, he can join the network and download parts of the file from any other network member that already holds parts of said file. Thus, unlike traditional client-server models, the performance of the network, if described by throughput, improves with a growing number of network members. More information about the BitTorrent protocol can be obtained in [2].

2.2 BitThief

BitThief is a so-called free riding BitTorrent client, which roughly means, that it can download data from the torrent network without uploading any. Detailed information can be read in [1]. Generally speaking, the original intention for developing such a selfish client was showing that the BitTorrent protocol can be abused by selfish clients. A possible solution for this problem was proposed in [3]. BitThief has been developed at ETH since about 2007, and was the subject of some previous bachelor and master thesis, thus being suitable for further development in this thesis.

2.3 Jetty

In order to extend BitThief with server functionality, the decision of which web server to use had to be made. Jetty seemed especially suitable as it is a lightweight webserver/servlet container and, like BitThief, Java-based. Furthermore, it is developed by the Eclipse Foundation and thus can be expected to stay maintained in the foreseeable future. Especially the convenient embedding process made Jetty stand out from its competition for this particular application.

2.4 Web Technologies

To make the web interface reactive, modern, and visually appealing, HTML, JavaScript and CSS have been used. HTML (HyperText Markup Language) is a markup language and the standard way to define the structure and hierarchy of web pages. CSS (Cascading Style Sheets) provides a way to describe the style of a web page and how the different HTML elements should be displayed. JavaScript is a scripting language that can be used to define the behavior of a web page, like reacting to the user pressing a button. Additionally, jQuery, a JavaScript library, has been used in order to simplify HTML document manipulation and AJAX requests (Asynchronous requests to the server). To simplify the creation of monotonous and complex elements of the user interface, another library, jQuery UI, has been used.

Chapter 3

Implementation

The BitThief source code had been modified in three distinct ways. Firstly, some of the existing code has been refactored in order to improve modularity. Secondly, the backend for the web server, offering the actual functionality of BitThief, has been implemented, and lastly, the frontend for the web server, which is responsible for the user interface, had to be created as well.

3.1 Architecture

The general setup of the architecture looks as follows: The functionality of BitThief is controlled by the *Environment* class. It basically provides the API of BitThief and can be used to, for example, start downloads. The *MainServer* class handles all things server related and is responsible for communicating with both the web frontend and the BitThief functionality (via the *Environment* class). It serves as a mediator, as it interprets requests from the web-frontend in order to call suitable methods of the *Environment* class. The web-frontend is a website that can be viewed and interacted with by the end user. Users can access it by querying the BitThief server with their preferred web browser. It displays human-readable information about the download (e.g. status, speed, etc.) and provides mechanisms to inform the server that certain changes are desired (e.g. pause download via a button).

3.2 Communication

In order to communicate with BitThief, the user has to query the web server using a web browser. The address reads as follows: *http://<ip-address>:<port>* where *<ip-address>* denotes the ip address of the machine that BitThief is running on and *<port>* denotes the configured port of the BitThief web server. The port number can be defined when starting BitThief by calling the *-port* starting parameter. If we assume that the starting script is called *BitThief.sh*, starting

the server on port 1234 would work by executing the following command (Linux): *./BitThief.sh -port 1234*

The communication between the frontend and the backend roughly works as follows: The frontend, which is the web browser running the JavaScript code and HTML, is shown to the user, who selects appropriate actions, like adding a magnet link. The JavaScript code converts these queries into POST and GET requests suitable for sending to the BitThief server (represented by the MainServer class). The BitThief server interprets said POST and GET requests using its handlers (described in detail in section 3.3), and takes appropriate actions (by using the *Environment* class described in section 3.5), like starting a download. Then, an answer (a JSON array or JSON string) is generated and sent back to the frontend. Thus, the actual download is happening on the device running the server, and not necessarily on the device running the frontend.

3.3 Backend Implementation

As mentioned before, the Jetty Webserver/Servlet-Container was used to implement the server functionality of BitThief. The component of Jetty that handles HTTP requests and responses is the *Handler* class, which executes the *handle*-method upon each new request. In order to respond to different kinds of requests, multiple handlers can be defined and executed sequentially. For this project, six handler classes were created, each for one specific purpose, and three predefined handlers were used.

The API (Application Programming Interface) of the server implementation for BitThief is the custom created *MainServer* class to handle all things that are related to its server functionality. *MainServer* wraps the *Server* class (responsible for actually running the web server) provided by Jetty and adds some initialization methods. When a message from the client arrives (a JSON object or JSON array), the handlers defined in the *MainServer* instance process it and execute appropriate methods of the *Environment* instance.

3.3.1 Predefined Handlers

In this section, all predefined handlers provided by *Jetty* will be described briefly.

The *ConstraintSecurityHandler* is used to prohibit unauthorized users from accessing the BitThief functionality and can block access to all files organized by the server. The actual authentication works using a username and password, which is matched against usernames and passwords in the *users.properties* file. This file is automatically generated in the same directory as BitThief when starting it for the first time and can be edited in order to add users or change passwords.

The *ResourceHandler* can serve static content like HTML, CSS and JavaScript documents. It forms a core part of the server, as the whole user interface is implemented in JavaScript.

The *DefaultHandler* is a simple handler implementation used to end the handler chain more gracefully. It produces a simple error page (also called 404 page) if the requested resource cannot be located.

3.3.2 Newly Created Handlers

In this section, all newly created handlers will be described briefly.

The *UpdateHandler* is responsible for answering requests about the current status of all downloads. On a suitable request, a JSON array is being created and returned with each JSON object encoding information about one specific download. Some of the included details are name, bytes downloaded, total number of bytes, bytes uploaded, download speed and the download hash.

The *EditHandler* answers requests intended to edit certain options about one specific download. Examples include pausing/unpausing a download, removing a download from the download list, and reading or writing settings for one particular download (not global settings). Figure 3.4 shows the dialog box of the frontend used to issue requests for editing those specific settings. The post request must encode parameters like the hash string of the desired download, a string encoding which action to take (like *pause* or *readSetting*) and, in case the request tries to write settings, information about which settings to change.

The *MagnetHandler* is responsible for starting a download from a magnet URI. The only parameter it needs is the magnet URI itself.

The *UploadHandler* is called when uploading a torrent file to the device running the server. Though the de-facto standard of most popular torrent trackers is the magnet-link, some trackers still use torrent files.

The *SettingsHandler* is a little different to the other handlers, as it responds to two different targets. If the specified target is *readSettings*, it will respond with a JSON object encoding the global settings of BitThief, if it is *writeSettings*, it will modify the global settings of BitThief according to the parameters encoded in the request.

The *DirectMagnetHandler* is the only handler not called by a request from the web frontend. It was added in order to ease the direct encoding of magnet links and server address in a single string. This means that tools accessing the BitThief functionality can now start a download more easily. If we assume the server was running on a machine with ip address *10.0.0.1* and listening on port *8080*, calling *10.0.0.1:8080/direct?<magnetLink>* would start a download for the magnet link *<magnetLink>*

3.4 Frontend Design and Implementation

The web frontend was implemented using *HTML*, *CSS* and *JavaScript*.

3.4.1 Design

To keep the design simple, the main interface only displays three buttons. One for starting a download using a magnet link, one for uploading a torrent file, and one for accessing the global settings. These buttons are situated prominently on the very top of the page. When a download is being started, either using a magnet link or a torrent file, a new box appears below the main buttons. Said box contains all necessary information about the download, like speed and status, and adds some control buttons. Additionally, a progress bar is being added to visually represent the percentage of downloaded data. Some screenshots of the interface can be seen in figures 3.1, 3.2, 3.3 and 3.4.

3.4.2 Implementation

The core component of the frontend implementation is a JavaScript file. It is responsible for the whole functional behavior of the web page. Additionally it tries to update the user interface periodically to reflect the changing state of the download. To accomplish this task, a GET request is sent to the server (handled by the UpdateHandler, as described in section 3.3.2) each second. In order to simplify some commonly used or sophisticated tasks, the script relies on the JavaScript libraries *jQuery* and *jQuery UI* (described in section 2.4).

The HTML file defines the hierarchical structure of the buttons and dialog boxes, while a CSS file is being used to define the styling of all HTML components.

3.5 Code Structure

In order to improve modularity and the separation of functionality and user-interface, some of the previously existing source code has been moved or altered. Instead of discussing details, a short description about the current structure of the functionality part will be provided. A graphical representation can be seen in figure 3.5.

The core class for all functionality matters is the *Environment* class. An instance of this class represents an instance of the BitThief functionality itself and contains all other important classes used to control it. If a program (in this case, a server) wants to use any of BitThiefs functionality, it can instantiate

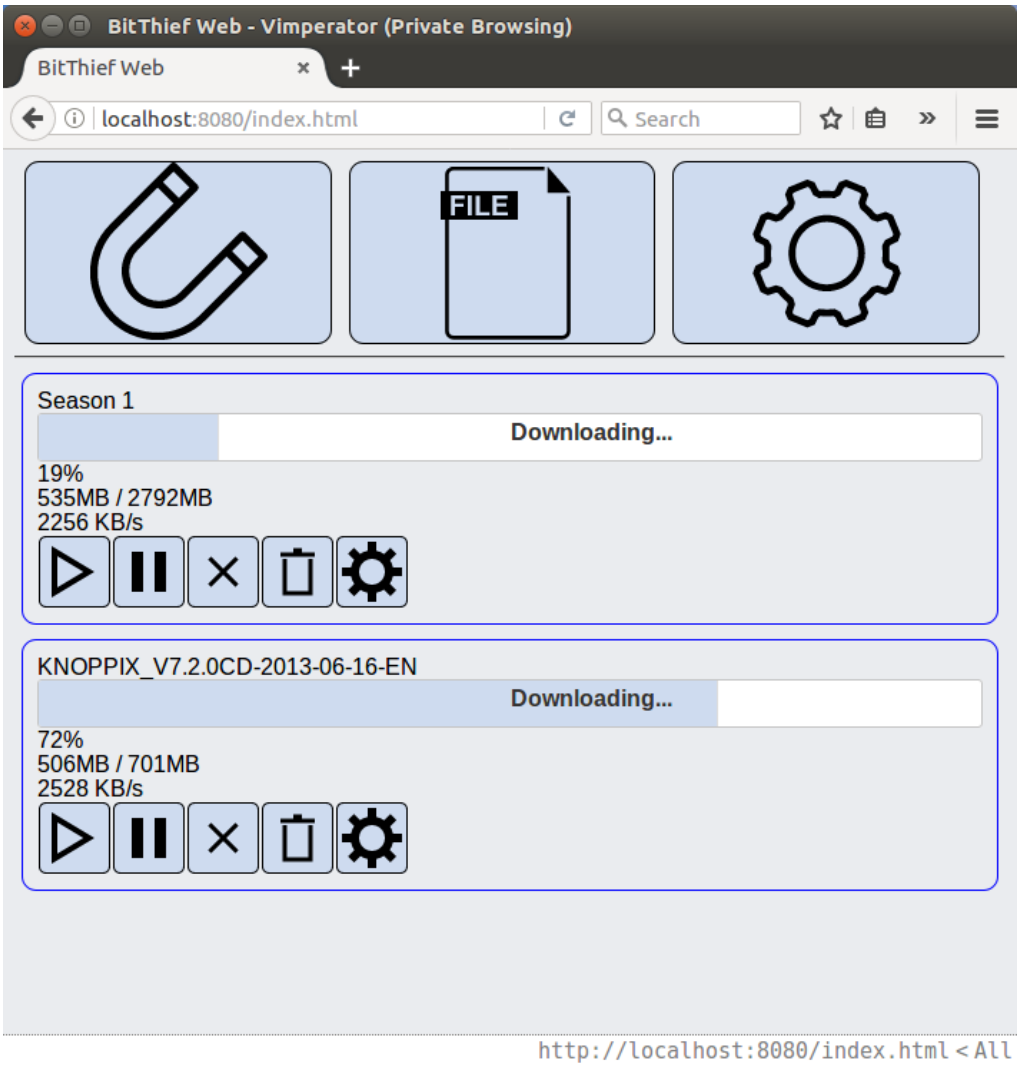

Figure 3.1: The main interface while running two downloads simultaneously.

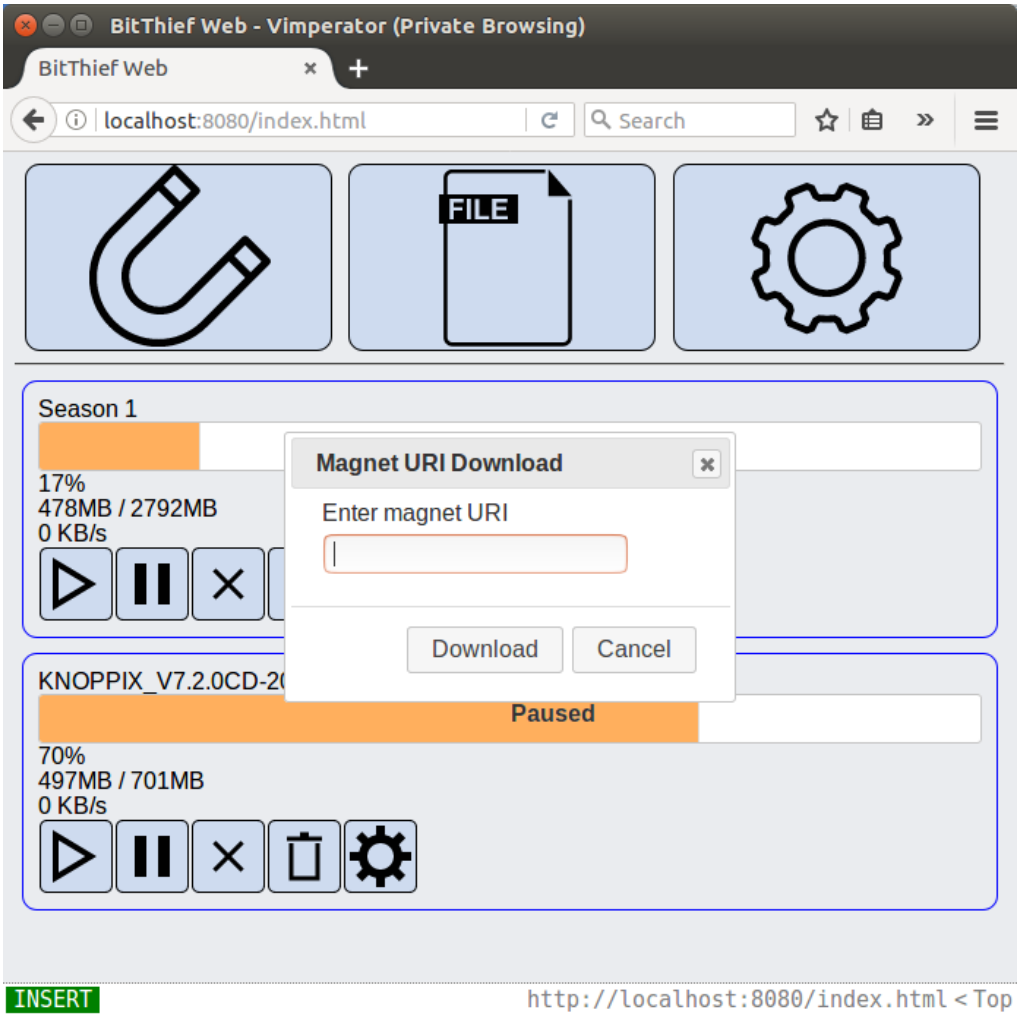

Figure 3.2: The dialog for starting a download via magnet link.

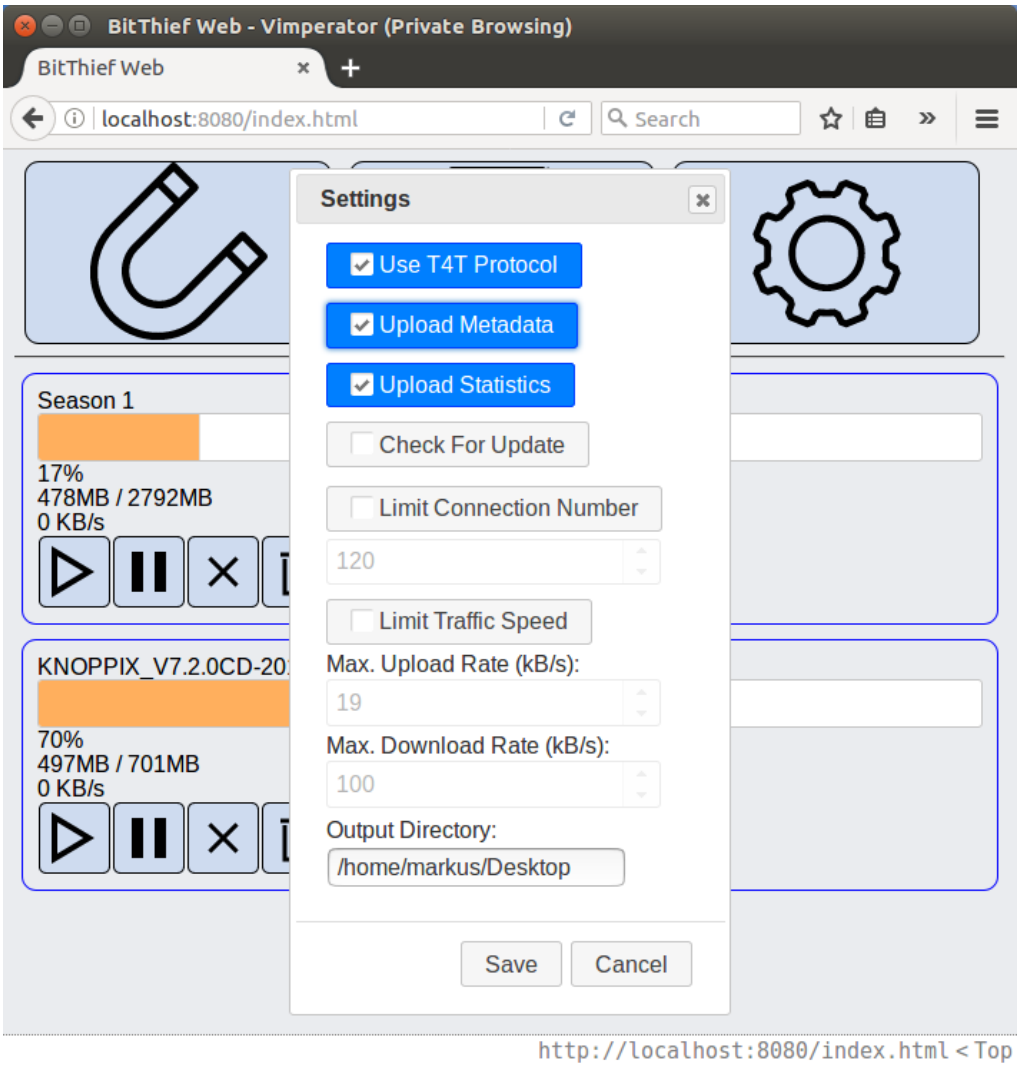

Figure 3.3: The settings dialog for changing global settings. Most of the settings options are self-explanatory. *Check For Update* lets BitThief check for available updates automatically. *OutputDirectory* defines the directory in which all files should be downloaded to. Keep in mind that it refers to the directory of the machine running the BitThief server and not necessarily the one running the web frontend.

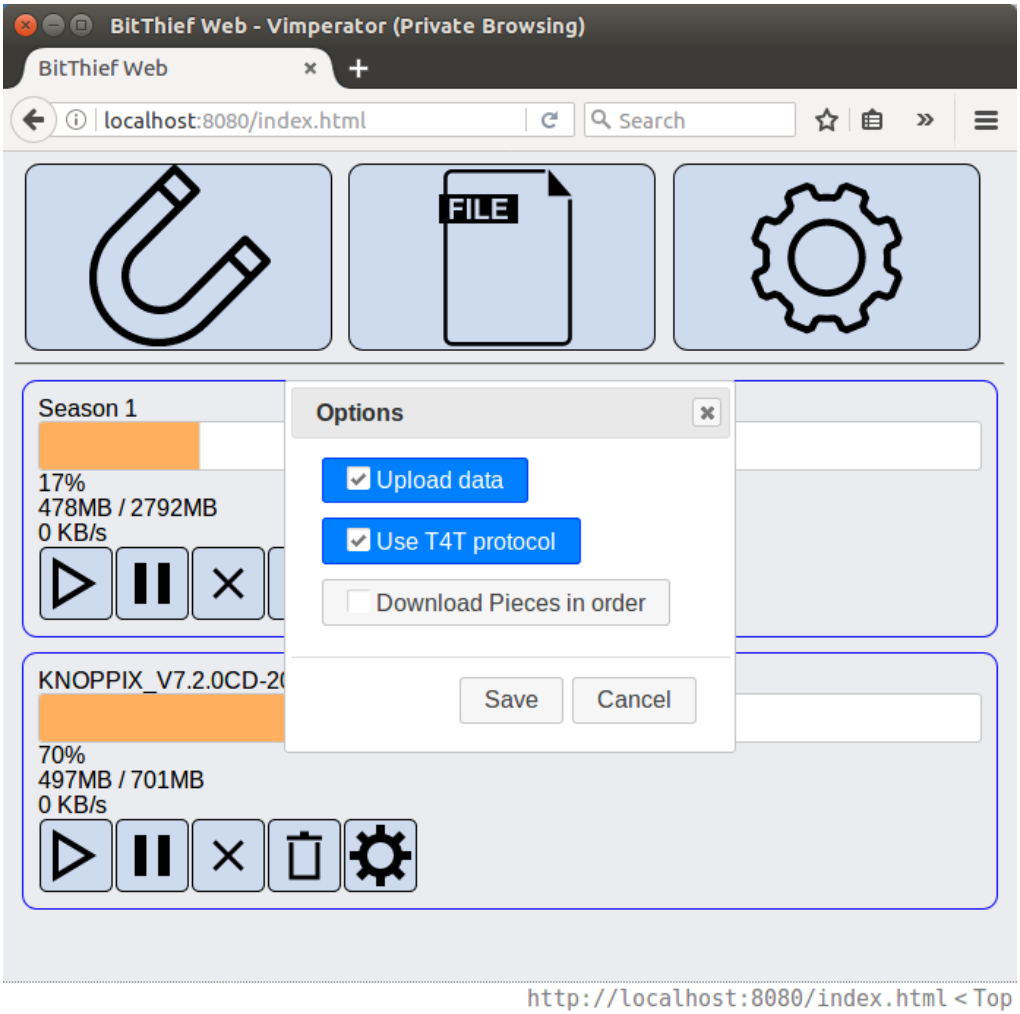

Figure 3.4: The settings dialog for changing download-specific settings. *Upload data* defines whether any data (except meta data) should be uploaded to the BitTorrent network. *Use T4T protocol* defines whether the T4T protocol should be used instead of the BitTorrent protocol. *Download Pieces in order* defines whether the pieces that a download consists of, should be downloaded in order or randomly

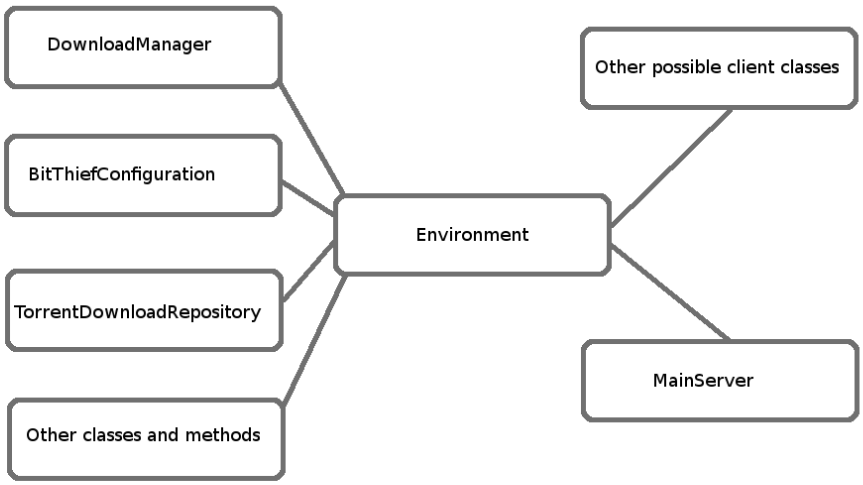

Figure 3.5: The general layout of BitThief. *Other classes and methods* represent more classes used by *Environment* to control the functionality. *Other possible client classes* represent other classes that want to use the functionality of BitThief, for example GUIs. A description of *Environment* and the classes on the left can be found in section 3.5. For more information about *MainServer*, refer to section 3.3

the Environment class or reference an already instantiated one. This way, the BitThief functionality could even be offered in form of a library.

An instance of *BitThiefConfiguration* is used to store and edit global settings for a particular Environment instance. It contains methods to read and modify global settings, like the maximum download rate, or whether the T4T protocol should be used. Figure 3.3 shows the dialog box provided by the web frontend to change those settings.

The *DownloadManager* class is used to coordinate downloads. Downloads can be added, paused, and removed using either the *TorrentDownload* object itself (described below) or a hash string encoding the download.

The *TorrentDownloadRepository* class contains, as the name implies, information about all the downloads currently processed by BitThief. Each of those downloads is being represented by a *TorrentDownload* object.

A single *TorrentDownload* instance represents an actual download. It contains all download-specific information, like the current status (downloading, paused, etc.), filename, the download speed, a hash string that uniquely identifies it, and much more. The hash string can be used to get the *TorrentDownload* object it represents by calling a specific method (*getDownload(hashstring)*) on *DownloadManager* or *DownloadRepository*.

The *BTPreferences* class is intended to store preference information that are more specific to the user interface.

Chapter 4

Future Work

As BitThief can now be run in server mode only, some interesting new utilizations are possible. As the previous thesis about BitThief [4] has already implemented a system to retrieve information about available torrents from indexes (Web pages that list available magnet links), a certain automation of downloading might now be desirable.

A possible useful extension would be a system that tracks downloading habits, and automatically downloads files that the user might be interested in. This could be useful if the user wants to obtain periodically appearing tv shows or shows of the same genre.

Another advancement could utilize the newly implemented function of starting a download via a string that encodes the address of the server and some magnet link. An example might be a browser plug-in that rewrites magnet links to links directly starting a download on BitThief. Once configured with the correct server address and port, any magnet link could then be clickable by the user and directly start the download.

Bibliography

[1] Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free riding in bittorrent is cheap. In: Hot Topics in Networks. (2006)

[2] Cohen, B.: Incentives build robustness in bittorrent. (2003)

[3] Locher, T., Schmid, S., Wattenhofer, R.: Rescuing tit-for-tat with source coding. In: Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing. (2007)

[4] Vasavada, V.: Bitthief new ux. Bachelor's thesis, ETH Zurich (2016)