ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing Group
Prof. R. Wattenhofer
Summersemester 2008

Semester Thesis

Extending the Functionality of BitThief

Student:
Ronny Milani

Advisor:
Thomas Locher

Abstract

The BitTorrent client *BitThief* was implemented as a proof of concept showing that free riding in BitTorrent is indeed possible. As such it only needed to contain the basic features of the protocol and did not deliver much user comfort or additional functionalities.
This thesis describes the new features added to *BitThief*. Their main aim is to improve the client's efficiency and to render it more user-friendly. Additionally, some suggestions for further improvements are outlined.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, we look deep into the interiors of *BitThief*, a free riding BitTorrent client. The aim is the elimination of bugs and the improvement of the client's overall performance. Especially the startup phase will get some attention as a lot of time is spent there.
Also some additional features will be added to expand the functionality and help *BitThief* to become a full BitTorrent client.

1.1 Motivation

Since BitTorrent became one of the most popular network protocols for file sharing, also a large number of clients have been developed, most of them adhering to the guidelines of the original proposed BitTorrent protocol by Bram Cohen [1]. Because this describes a fair file sharing protocol, their users are expected to contribute to get an acceptable download rate in return.
Especially for users with a small upload bandwidth this may result in a bad download performance. For those people a client that does not force its users to upload would be better suited.
Another reason for not contributing is the legal aspect. In Switzerland downloading copyrighted material is legal — except for software — while uploading would result in a copyright infringement. In countries with this legal position, being able to set the upload rate to zero would help to avoid legal problems.

BitThief delivers exactly the ability to free ride in BitTorrent networks. We now try to enhance *BitThief's* performance and also its functionality to get a fully functional client well suited for everyday use.

1.2 Contents

In the following, we take a short look at the BitTorrent protocol and outline especially the mechanisms used in *BitThief* to make free riding possible.
Further, we describe the extensions made during this thesis to *BitThief* to improve its performance and expand its functionality.

Chapter 2

BitTorrent and BitThief

The BitTorrent Protocol as well as the reference implementation known as the mainline client, were devised in 2001 by Bram Cohen. The intention was to distribute network traffic among all those peers actually downloading or having downloaded the offered content. This is achieved by using downloaders at the same time to upload already received data to other peers.

The BitThief client, was originally developed by Patrik Moor [2]. It was a proof of concept showing that free riding in BitTorrent, contrary to popular beliefe, is possible.

2.1 BitTorrent Protocol

2.1.1 Distributing and Accessing Content

Sharing files over BitTorrent requires three different components. A torrent *metafile*, a server/*tracker* and clients sharing the file. All these together form a so called torrent *swarm*.

The torrent metafile contains all necessary information to contact the swarm and details about the files being shared:

- names/folders
- size
- tracker url
- number and size of pieces (the files are split into smaller pieces)
- hash value of all pieces

Metafiles can be stored and distributed in every desirable way.

The tracker mentioned in the metafile is responsible for storing the IP addresses of all clients together with a hash of the metafile (*info hash*). The tracker then forwards this list on request to other peers.

In order to start downloading a file a client has to obtain the metafile and announce itself to the tracker. The clients IP will be stored at the tracker and in return a list of active peers interested in the same file is sent back. This precedure has to be done regularly to always keep a current picture of the swarm. The client is now able to contact other peers in the swarm and start downloading from them.

2.1.2 Peer-to-Peer Communication

Before downloading from a peer a handshake is performed to establish the connection. Afterwards the actual download progress is communicated. This information is held up to date by sending a "have" message when a new piece has been received.

Peers interested in a piece not yet downloaded will send an "interested" message. If a remote peer has no pieces of interest anymore this can be indicated by sending a "not-interested" message. Initially, peers are always not interested in each other. This mechanism helps to efficiently choke and unchoke the right peers, as we now know which peer will request pieces.

Unchoking/choking decides whether we allow a peer to download or not. By sending an "unchocke" message we signal that requests will be answered while a "chocke" message signals that requests will be ignored. This decision is reconsidered from time to time and is based on the amount of data the remote peer uploaded to us.

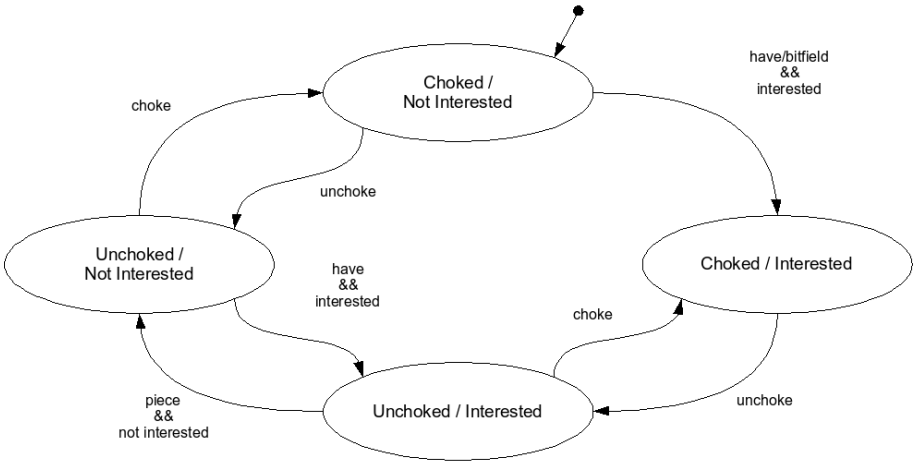

Figure 2.1: State Diagram

Hence, a peer can be in one of four states depicted by the state diagram in Figure 2.1. A peer being interested and unchoked is allowed and also willing to send a "request". Requests already answered by other peers can be aborted by sending a "cancel" message.

2.2 BitThief

Now we will have a look at the mechanisms used by *BitThief* to exploit the weaknesses of the BitTorrent protocol.

2.2.1 Connection Opening

To speed-up the bootstrapping process, *BitThief* opens as many connections as possible as fast as possible. Since a single announce to the tracker usually only delivers a list of up to 50 peers, *BitThief* contacts the tracker more often than the announced interval, to get more peers in a short time. Surely this should not be exaggerated to avoid being banned from the tracker.

2.2.2 Exploiting Seeders

The probably weakest point in the BitTorrent protocol is the *seeders* behavior. The term *seeders* refers to clients being in posession of the whole file. Clients missing any pieces are referred to as *leechers*.
As seeders do not download from other peers, they have no possibility to know which peers are uploading to the swarm and which are not. Their decision which peer to choke and which to unchoke is therefore not based on the level of contribution. They simply upload using a round-robin algorithm. Hence, also peers not uploading anything are served.

2.2.3 Uploading Garbage

To convince leechers that a free riding peer actively participates in the swarm, there is the possibility to upload fake data instead of valid pieces. As complete pieces will be checked against the hash value in the metafile, uploading a full piece would reveal the free rider's bad intentions.
The fact that pieces are additionally broken into blocks gives us the possibility to upload only a percentage of all blocks and therefore prevent a hash check or at least keep the downloader in the dark about which peer sent garbage if it received blocks from different peers. Unfortunately modern clients try to download full pieces from a single peer. Just not answering requests for certain blocks would end in the opposite side stalling our requests. Therefore this attack is not really useful.

Chapter 3

Extensions for BitThief

This chapter presents the new functionalities added to *BitThief*.

3.1 Upload

As *BitThief* was up to now only used to send garbage over the network, seeding own content was not yet possible. But as a user might want to do exactly this or maybe someone really wants to contribute in a swarm by uploading real data, the uploading functionality was added to enable this. But although the possibility to upload exists now, *BitThief* remains a free riding client if not told explicitly to upload.

3.2 Enhanced Resume Functionality

A user starting a client or resuming a paused download is highly interested in achieving the client's maximum performance again as fast as possible. Also if a client only runs for short time intervals one nontheless expects some progress to be made.

BitThief had one major shortcoming preventing him to achieve a quick startup period. Resuming a download resulted in checking the hole file for already downloaded pieces by first reading them and afterwards verifying them against their hash value. Especially for large files this operation took a long time.
Another drawback of this procedure lies in the impossibility to recover partially downloaded pieces. As the hash value of the stored piece will not be equal to the one given in the metafile, they will be automatically identified as not yet downloaded. Since *BitThief* tries to get as many pieces in parallel as possible, there might be a lot of pieces in progress at any time resulting in many lost blocks if the download is stopped in such an unfortunate moment.

We solved both these problems by saving the actual download progress to a separate file. As the structure of this file may be changed or extended in later versions of *BitThief*, XML was chosen for this task.
Every download gets its own progress file which is created or updated whenever the download or BitThief itself is stopped. By removing the download from the list the progress file will automatically be removed. To guarantee no conflicts with filenames they are named after the 'file hash'.

The structure of the XML file is as follows:

```
<torrent_download>
  <name>download name</name>
  <valid_pieces>
    <piece>x</piece>
    ...
    <piece>y</piece>
  </valid_pieces>
  <downloaded_blocks>
    <block>
      <index>index</index>
      <length>length</length>
      <offset>offset</offset>
    <block>
    ...
  </downloaded_blocks>
</torrent_download>
```

3.3 Multi-Tracker

A single point of failure in a BitTorrent swarm is the tracker. In the original protocol a crashed tracker resulted inevitably in a dying swarm because peers could not get any new peers from the tracker.

One method to diminish the consequences of a server breakdown is based on the use of multiple trackers. For this purpose, in addition to the single tracker contained in every metafile, a list of supplementary trackers is given. In fact a list of lists/tiers is contained in the metafile.

[tracker-list] = [[tier0][tier1]..[tierN]]

The usage of these tiers is supposed to follow some simple rules.
First of all, tiers should be accessed sequentially one after the other. Every additional tier serves as a list of backup trackers. Moving to the next tier should only

be done if no tracker in the former tier replied to our announce.

[tracker-list] = [[tier0][backup0]..[backupN]]

When accessing a tier for the first time the contained urls have to be shuffled to guarantee load balancing among the different trackers. Afterwards, the trackers are as well contacted sequentially one after the other. The first one replying to our request will then be moved to the top of the list. Hence, in later announces it will be used first from this tier.

Clearly, announcing to all trackers would lead to a shorter startup period as we can get more peers in the same time. Therefore, such an algorithm seems very interesting for us. Unfortunately trackers in one tier — and maybe also in different tiers — are suggested to communicate their peer lists. So using such an aggressive strategy could easily be counteracted in any desirable way by the trackers.

Nonetheless, we implemented both strategies for *BitThief*, the suggested and the aggressive announce to all. It seemed as if actual trackers do not yet support this peer list exchange, since no deficit could be detected. We were still served normally by all contacted trackers.

3.4 Connection Limit

Due to *BitThief's* aggressive policy concerning the opening of connections to other peers, a lot of memory is consumed. Also some reports of routers crashing due to the massive number of tcp connections, require the possibility to limit the number of concurrent connections.
Surely, just restricting to a maximum value would be no difficult task, but also not a very desirable solution.
The main problem of this naive approach are bad connections. If the connection limit is already attained but the bigger part of all connected peers influences the download performance in an unfavorable way, we are stuck in this unfortunate situation.

Again the solution is fairly obvious. If new peers want to connect to a peer although its limit is already reached, the bad connections are simply closed. But this leads to the next question — how to characterize a bad connection?

We classify connections after different criteria. First, all connections being of no interest, hence not having any required pieces, are considered bad. If no peers fulfill this characteristic, those peers refusing to cooperate are closed. Should again no peers have been selected yet, we disconnect from the peers with the lowest rate.

3.5 Speed Limit

Although we want our downloads to be as fast as possible, there are several occasions where we would like to throttle our network speed.
In an environment where several persons use the same link to the Internet, it may not be desirable to use up the whole bandwith only for downloading.
Maybe also the cpu load gets to high to allow normal usage of the computer.
Or a slower downloading torrent may starve due to a faster torrent using up all the available link capacity.

To limit download speeds we regulate the period between two successive "request" packets. This way we can easily calculate a basic time interval as starting point which we can slightly adapt to get even closer to the desired limit. In practice it turned out to be sufficiently accurate to simply calculate the period (3.1).

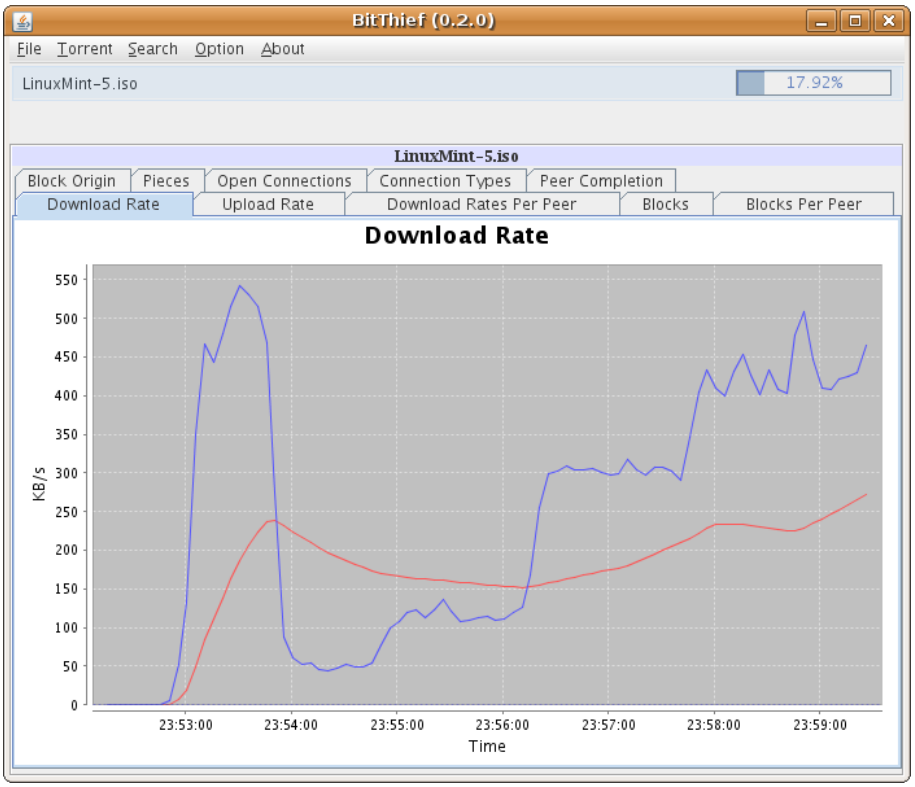

Figure 3.1: Speed Limitation in Action - No Limit -> 50kB/s -> 125kB/s -> 300kB/S -> No Limit

$$t = blocksize/speedlimit \tag{3.1}$$

By the way, the same concept is used to adjust the upload speed. But as we do not have to regard any delays or churn here, by using the same equation as for the download we can calculate an exact value for this period.

Chapter 4

Further Extensions

This chapter describes some ideas on how to improve *BitThief*.

4.1 Improve Memory Usage

The most obvious problem turned out to be the memory management. It seems that memory allocated by a download is never really released again. When a torrent is removed the former utilised memory space is still occupied.
Also big torrents proved to be a problem. This could be solved by writing partially available pieces to the storage device instead of holding them in memory.

4.2 Peer Exchange (PEX)

Peer Exchange is intended to reduce the load on trackers. This extension allows peers to gossip. Meaning that peer lists can also be obtained from other peers.
As this feature is not part of the official specification, several implementations have been developed in parallel, the two most popular being the Azureus Message Protocol (AZMP) and the LibTorrent Extension Protocol (LTEP). Also a negotiation procedure has been proposed which allows to agree on one of these two protocols.

Nevertheless, peer exchange cannot substitute the periodic announces to a tracker. A short example shows how the reliance only on PEX could separate a swarm in two individual partitions which would never join again until they contact a tracker.

Those being interested in the details of these protocols will find more information under [3]. Unfortunately those specifications are not very accurate. A look into an actual implementation would be unavoidable to successfully implement these protocols.

The benefit of such a protocol would be minor for us. As we are not interested in reducing the network load on the trackers, we only profit by getting more peers

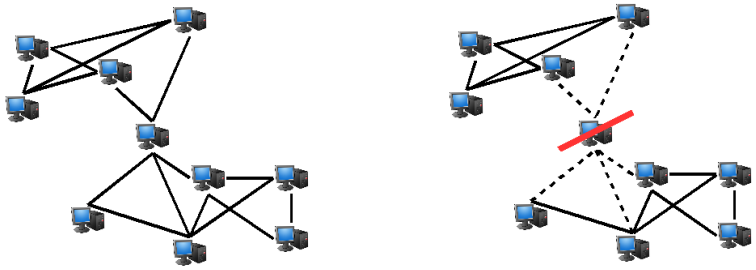

Figure 4.1: BitTorrent swarms relying only on PEX can easily get partitioned.

in a shorter time.

4.3 Fake ID

BitThief indentifies itself as the *Mainline* client version 4.4.0. This is one reason why blocking our client is not very easy.
But instead of just using the same ID for all connections one could gather and store the IDs of connecting peers and reuse those for our own purpose. Blocking our selfish client would become even more complicated.
Also our identity would reflect the current distribution of clients on the net and always have an up-to-date version number.

Chapter 5

Conclusion

The achieved results are definitely of great value for the *BitThief* client. Especially the startup phase could be shortened drastically. Also the numerous smaller changes such as resolved error messages are valuable improvements to *BitThief's* stability. Additionally the useability could be elevated observably. Mainly the interaction with the user is faster than in previous versions. Deadlocks where the GUI does not reply anymore to user interactions are banned. Though the program can still get quite slow when a lot of CPU consuming tasks are running in parallel.

Although I was able to implement a few useful extensions, I would have liked to go further. Unfortunately, it turned out to be much more time consuming to code a new feature than I initially believed. Maybe this resulted also from my lack of experience with such large projects. Also the fact that every new functionality brought some new errors from previous changes to the surface, slowed down the progress.

Nonetheless, it was a very interesting work from which I could benefit a lot. I was able to deepen my knowledge of Java and its debugging methods and was able to get familiar with subversioning.

Recapitulatory I am pleased with the added functionality and their quality. However I am not satisfied with the progress I made. I would have liked to have at least enough time to resolve the memory problem as well.

Bibliography

[1] BitTorrent wiki,
wiki.theory.org/BitTorrentSpecification.

[2] P.Moor. Free Riding in BitTorrent and Countermeasures,
Master Thesis Summer 2006.

[3] Peer Exchange,
wiki.theory.org/BitTorrentPeerExchangeConventions.

[4] T.Oetiker. The Not So Short Introduction to LaTeX, February 2000.

[5] Wikipedia,
en.wikipedia.org.

[6] XML Pull Parsing,
http://www.xmlpull.org/.

[7] Java API,
java.sun.com/j2se/1.5.0/docs/api.