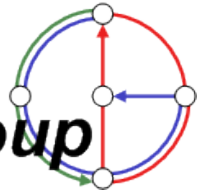


ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

**Distributed
Computing Group**



Semester Thesis

To Be or Not to Be a BitThief

Winter 2009/2010

Mathias Karlsson
mathiask@ethz.ch

Distributed Computing Group

Prof. Dr. Roger Wattenhofer

Supervisor: Raphael Eidenbenz

Thomas Locher

Abstract

The BitTorrent protocol does not strictly enforce sharing which was proven with the BitTorrent client BitThief that only downloads, but does not upload. As an improvement to standard BitTorrent clients BitThief implements a secondary protocol called *TitForTat (T4T)* that enforces uploading. To switch to the T4T protocol two BitThief clients that meet in a BitTorrent swarm first have to be able to identify each other.

This thesis describes how BitThief clients can secretly identify each other without evoking suspicion among non-BitThief clients.

Contents

- 1 Introduction 4
 - 1.1 Motivation 4
 - 1.2 Contents 4
- 2 The Extension protocol (LTEP) 5
 - 2.1 Handshake 5
- 3 The Peer Exchange (PEX) 6
 - 3.1 Peer Exchange Message (ut_pex)..... 6
 - 3.1.1 PEX Message Parts..... 7
 - 3.1.2 Added Flags 7
 - 3.1.3 A PEX example 8
- 4 BitThief Handshake over PEX..... 9
 - 4.1 Introduction..... 9
 - 4.2 Creating PEX Messages..... 10
 - 4.3 BitThief Handshake..... 10
 - 4.3.1 IP Creation and Injection Into PEX Message 10
 - 4.3.2 BitThief Detection..... 11
- 5 Future Improvements 12
 - 5.1 Bitfield 12
 - 5.2 Encryption 12
 - 5.3 NAT traversal 12
 - 5.4 Improve LTEP Handshake 13
 - 5.5 Improve IP Creation for BitThief Detection..... 13
- 6 Conclusions 14
- Appendix 16
 - LTEP Handshakes..... 16

Chapter 1

Introduction

In this thesis, we discuss the possibilities to hide a handshake in the BitTorrent protocol. The goal is that two BitThief clients are able to identify each other without other clients being able to detect the handshake.

In addition this report documents the findings in the usage of the LibTorrent Extension Protocol and Peer Exchange. Both BitTorrent Enhancement Proposals did not reach the ‘accepted’ status yet, but are already heavily used.

1.1 Motivation

The BitThief client supports the possibility to share content through a secondary protocol that enforces uploading. However, before this can happen two BitThief clients have to be able to identify each other. This is not a trivial task since the BitThiefs cloak themselves as other clients in order not to be detected and banned. On one side the BitThief client does not want to be identified as BitThief, on the other hand it should be recognized as BitThief by other BitThief clients.

Up until now this hidden handshake was implemented by setting an unassigned Bit in the reserved bytes within the primary handshake. This worked well, as other clients are in general very tolerant to minor protocol violations, because many clients implement the BitTorrent protocol slightly different. It is trivial for any BitTorrent client to detect a BitThief; it only has to check for the ‘BitThief bit’ in the handshake and then it can just drop the connection.

This is not unlikely to happen in the future, because more and more selfish clients appear, trying to impersonate the μ Torrent client, or other popular clients to keep a low profile. This has led to the result that some trackers, which do not want to support such ‘evil’ clients, also reject μ Torrent clients. The client developers are likely to add further mechanisms to verify or identify other clients to make it harder to impersonate clients.

Hence, it is important for the BitThief client to have a secret handshake that cannot be identified as such to maintain its advantage in the future.

1.2 Contents

First we have a brief look at the Extension Protocol and the Peer Exchange message, as we use them to hide the secret BitThief handshake. Second, we describe how the new BitThief handshake for the T4T protocol exploits the peer exchange mechanism.

For more information about BitTorrent or BitThief in general, refer to the work of Patrik Moor [1], Dorian Kind [2], and Ronny Milani [3] or check out the BitTorrent Wiki [4].

Chapter 2

The Extension protocol (LTEP)

The *LibTorrent Extension Protocol (LTEP)* is the draft *BitTorrent Enhancement Proposal (BEP) 10* [5] created in January 2008, last edited end of February 2008, and it did not change the status ever since. Even though it is only a draft, most of the clients support it, as it allows adding features without a change of the basic protocol or using bits of the reserved bytes in the primary handshake. It also allows to dynamically changing the support for features without having to restart the client.

There exists also another extension protocol called *Azureus Extended Messaging (AZMP)* created by the Azureus/Vuze developers, which is only supported by the Azureus/Vuze and Transmission client.

2.1 Handshake

In the handshake, the client can advertise in a dictionary which messages it supports, and which extension header ID should be used for that feature. Other information that might be important to the other client can be sent that e.g. the client version.

Commonly advertised message support in the dictionary:

upload_only	partial seed (BEP 21) [6]
ut_holepunch	µTorrent hole punching
ut_metadata	metadata message (BEP 9) [7]
ut_pex	µTorrent Peer Exchange (BEP 11)

Commonly used handshake parameters:

p	sender TCP listening port
v	client name and version
yourip	receiver IP
ipv4	sender IPv4 public interface, byte encoded, 6 byte
ipv6	sender IPv6 public interface, byte encoded, 18 byte
reqq	number of outstanding requests supported
metadata_size	number of bytes of the metadata (BEP 9)

A detailed example can be found in the appendix.

Chapter 3

The Peer Exchange (PEX)

Peer Exchange (PEX) messages are exchanged between peers to gain faster and better knowledge about the swarm to make it more resilient against separation. Since tracker operators are being more and more targeted by law enforcement tracker independent technologies, like peer exchange, are getting more attention.

There exist two Peer Exchange Protocols, one developed by the μ Torrent developers called *ut_pex* and the other one by the Azureus/Vuze developers called *az_pex*. The *az_pex* message uses the Azureus Extended Messaging protocol (AZMP), and is only supported by Azureus/Vuze and Transmission. The *ut_pex* message uses the LibTorrent Extension Protocol (LTEP) and is widely supported, and also implemented in BitThief.

There exists an unofficial agreement [8] between the Azureus and μ Torrent developers to limit the amount of transmitted peers, but it sometimes is violated as can be seen in PEX crawls [10].

3.1 Peer Exchange Message (*ut_pex*)

The *ut_pex* (BEP 11) message is a LTEP message consisting of six lists containing information about peers with which the client either has a connection, or to which it recently lost the connection. The message can be separated into two main parts, a list of connectable peers called 'added' and a list of not connectable peers called 'dropped'. For each 'added' peer, there exists a flag byte indicating some of the features supported by that peer. The IPv4 section of the 'added' part is the most used section, the IPv6 section is probably uncommonly used due to the lacking of IPv6 support. The 'dropped' section is at the moment often empty.

3.1.1 PEX Message Parts

The information of each peer is encoded into a byte string. An IPv4 peer needs four bytes for the IP and, two for the port. An IPv6 peer needs 16 plus two bytes respectively. Peers of the same type are appended to the same list and the according flag is appended to the flag list at the same position.

List label definition:

added	IPv4 peer, 6 bytes per peer (4 IP + 2 port)
added.f	IPv4 peer flag, 1 byte per peer
added6	IPv6 peer, 18 bytes per peer (16IP + 2 port)
added6.f	IPv6 peer flag, 1 byte per peer
dropped	IPv4 peer, 6 bytes per peer (4 IP + 2 port)
dropped6	IPv6 peer, 18 bytes per peer (16IP + 2 port)

There are no flag lists for the 'dropped' lists, since this information is obsolete.

3.1.2 Added Flags

A flag byte indicates the features supported by the corresponding peer in the added list. As the protocol is not approved yet the flag bits are not officially assigned. Out of the maximal 8 bits the bits 0 to 4 seem to indicate the following features.

Flag definitions (bit # supported feature):

- 0 encryption
- 1 seed or partial seed
- 2 μ TP support, μ Transport protocol support
- 3 hole punching support
- 4 is outgoing connection, is connectable from the internet

3.1.3 A PEX example

On this example the different message parts will be explained. The received byte encoded message would look like as follows.

BitTorrent Header Byte	LTEP Header Byte	PEX Message
------------------------	------------------	-------------

1:d5:added12:aaaaabbbbb7:added.f2:☹◀¹6:added60:8:added6.f0:7dropped6:cccccc:8:dropped60:e

The BitTorrent header byte was already removed since it does not matter for this example. The first byte, in this case it is a “1”, is the LTEP header byte. The value was defined in the LTEP handshake in the parameter “ut_pex=1”. The rest of the message is a byte encoded dictionary of the actual PEX message. In a decoded form called dictionary it would look as follows.

```
{added='aaaaabbbbb', added.f='☹◀', added6='', added6f='', dropped='cccccc', dropped6=''}
```

In this message both IPv6 lists were left empty, but the IPv4 contain the following information.

```
added peer 'aaaaaa' supporting the features '☹'
added peer 'bbbbbb' supporting features '◀'
dropped peer 'cccccc'
```

The peer address can be extracted by interpreting the first four bytes separately as part of an IPv4 address, and the fifth and sixth byte together as TCP port. The features the peers should support can be found by checking for the set bits in the corresponding flag for each peer.

The given addresses will decode into the following peers.

```
added peer [aaaaaa, ☹] → [97.97.97.97:24929, 0x02] → [97.97.97.97:24939, seed]
added peer [bbbbbb, ◀] → [98.98.98.98:25086, 0x11] → [98.98.98.98:25086, encryption, outgoing]
dropped peer [cccccc] → [99.99.99.99:25443]
```

¹ These symbols are used in this example to represent non printable ASCII values since the values are quite small. In the end of the example they will be replaced by their hex value.

Chapter 4

BitThief Handshake over PEX

4.1 Introduction

Before this semester thesis, the BitThief handshake was implemented in than an unassigned bit (*evil bit*) in the reserved bytes of the initial handshake was set. Unfortunately, this is rather trivial to detect by a non-BitThief client, if it only would check if any unassigned bits are set. This is in general not done yet, because such a restrictive policy would lead to problems. All clients would be required to have an up to date list of assigned bits. Otherwise an older client would refuse a connection to a more modern client since it does not know about the recently assigned bit. Also developers might want to try out new features that require the usage of a reserved bit in the initial handshake. Either the experimental clients would be refused, because they violate the protocol, or the developers have to wait until their feature gets approved. This would slow down innovation significantly. But the setting of such an evil bit might not only lead to detection, but, moreover the used bit might be assigned to an official feature. From that moment on a BitThief identifies all clients supporting this new feature as BitThiefs, and the other clients might disconnect the BitThief as it does not support the advertised feature. The initial handshake is clearly not the way to go. However the initial handshake is not the only message exchanged before data exchanged, and it is therefore worth to have a look at these messages.

The initial handshake is normally followed by the Bitfield message. The Bitfield informs about the pieces the peer already has, and is willing to share. Traditionally BitThief clients advertise to have no file blocks. Instead of having an empty Bitfield it would be possible to encode information into the Bitfield message by advertising a special pattern that is recognizable by another BitThief. But this could lead to malicious behaviour if the other peer is a non-BitThief client since the BitThief might be forced to advertise pieces it does not have or is willing to upload to create this special pattern. However, exactly such a piece could be of interest for the non-BitThief client requesting it of the BitThief. As the BitThief is not uploading the requested piece the non-BitThief client could interpret this as malicious behaviour, and also cease uploading to the BitThief, or even drop the connection. Therefore it is not a good idea to hide information in this message.

In case the peer advertised support for the LTEP protocol in the initial handshake it is required to exchange the LTEP handshake. One could introduce a parameter for this handshake advertising that the client is a BitThief. In comparison with setting the evil bit in the initial handshake this would at least prevent the collision with other features. Nevertheless it would be entirely trivial to detect since one could simply check if the BitThief parameter is set.

If in the LTEP handshake the client advertised support for the PEX feature it should send a PEX message. Unlike in previous exchanged messages it does not make sense to assume possible malicious behaviour if some information cannot be verified, meaning a peer in the PEX message turns out to be not connectable. One cannot assume that all the peers are connectable due to several reasons like a connection loss of the peer or one-way connectivity. Experiments [9] showed that depending on the time and swarm only about 10-25% of the peers in a PEX message are connectable i.e. it is not unusual that a peer is not reachable. It is therefore not unusual that even peers in the 'added' part of the message are not connectable. Obviously peers in the 'dropped' part are very unlikely to be connectable. It seems to be a feasible approach to hide a message within the PEX message by which two BitThief clients can identify each other and for other peers it is just another not connectable peer. This seems to be by far the best place to hide our new BitThief handshake and the following sections will explain the details of this new handshake.

4.2 Creating PEX Messages

The BitThief client so far did not create and send PEX messages but only received PEX messages and added new candidates as this was sufficient for a selfish client. However, this asymmetric behaviour could be classified malicious and punished by other clients. For Example the empty PEX message of the KTorrent client is treated in such a way by some clients. But also for the new BitThief handshake the BitThief client requires a possibility to create a PEX message.

4.3 BitThief Handshake

The BitThief handshake takes place after two newly connected peers advertised in the LTEP handshake that they support PEX. The client generates a forged IP socket, injects it into the prepared PEX message and sends to the other client. Upon receiving a PEX message the client generates the same IP socket as the sending client would have. Each IP socket in the PEX message is compared to the generated IP socket, and if a match is found the T4T connection started.

4.3.1 IP Creation and Injection Into PEX Message

To forge such an IP socket uniquely for each connection and torrent the client takes the unique peer ID of both clients, the torrent hash, as well as a secret key and creates a hash of it. Of this hash the client tries to generate a valid IP address and port.

The generated information is injected at a randomly chosen position of the stored PEX message. If the information is injected into the 'added' section of the message the flag for the peer will be randomly chosen.

To reduce the risk of detection this identifier is only sent in the first PEX message. Additionally, if the initializing peer is a BitThief it waits until it receives a PEX message checks it and only adds the forged entry if the other peer is identified as BitThief. However, in the case where a BitThief wants connects to another BitThief none would send the first PEX message as both would wait for the other one to send it. To resolve this problem it was decided that the peer that gets contacted sends the first PEX message that always contains the forged entry. This is the better solution since the BitThief client does not support hole punching and is therefore mostly not connectable.

4.3.2 BitThief Detection

Upon receiving a PEX message, the client creates the forged IP socket the same way as a sending BitThief would have generated by switching the peer IP of the remote and the local peer. This information is then checked against all peers in the message and if a match is found the client assumes that the other peer is a BitThief. There is the possibility of a false positive, i.e., that the forged entry matches to a genuine peer entry. However, the probability of such an accidental match is very low as all the 48 bits of the IPv4 socket would have to match, and in case of an IPv6 entry even less probable.

Chapter 5

Future Improvements

5.1 Bitfield

With the Bitfield, the client advertises which pieces it already possesses, and offers them for downloading. Instead of sending an empty Bitfield, BitThief could wait and reply with a sub selection of the received Bitfield. This way the BitThief client would not always advertise an empty Bitfield since this could raise suspicion after downloading several blocks of a peer and still advertising to have no pieces at all. Nevertheless the other peer would not be interested since it already has all the pieces the BitThief offers.

5.2 Encryption

With the increasing popularity of BitTorrent, the traffic load for ISPs increased. As this is not in their interest the operators started to shape or even block the BitTorrent traffic. As a reaction many BitTorrent clients started encrypting the traffic. Users may decide to enforce an encrypted connection, which means that the BitThief client fails to build up a connection to these peers. Without encryption support the BitThief client would only be able to communicate with that part of the swarm that has encryption deactivated. To not lose the option to download from peers that demand encryption it might be a good idea to extend the client with the encryption scheme.

5.3 NAT traversal

At the moment, a BitThief client that is behind a NAT is cannot receive any connections from the internet. If two BitThief clients should find each other it is important that at least one of them is connectable to initiate a connection. A mechanism for NAT hole punching could remove this problem and might also improve the performance as also other peers might want connect to the BitThief behind the NAT.

5.4 Improve LTEP Handshake

Different clients advertise different support of features in the handshake, but also different additional information. Some of them are fairly commonly used such as the connectable port and the version of the sender and the IP of the receiver the sender sees.

Right now it is quite easy to detect a BitThief that cloaks itself as a μ Torrent because it only announces itself as μ Torrent in the primary handshake, but does not approve it in the version parameter in the LTEP handshake. Since also other clients beside BitThief implement a 'cloak as μ Torrent' option the real μ Torrent client tries to verify if the announced μ Torrent client is really a μ Torrent client. In the case the verification fails the μ Torrent client displays for the peer as client version 'fake μ Torrent'. The difference in the LTEP handshake can easily be seen in the example in the appendix.

5.5 Improve IP Creation for BitThief Detection

Unfortunately, in the current version the algorithm forging the IP socket is excluding obvious wrong IP addresses. It could still happen that it would generate an IP address that is not assigned, or, i.e., the address of the Google DNS server. However, the avoidance and the detection of such invalid addresses face the same problem of maintaining a list of such invalid IPs.

Chapter 6

Conclusions

The replacement of the BitThief's relatively obvious handshake with a hidden handshake contributes substantially to the goal of keeping BitThief's profile low, also in the future. Before this thesis it was enough to ban all clients violating the protocol, and BitThief would have been banned as well. Now, the BitThief client is not violating the protocol anymore, and to build a 'BitThief check' it would be necessary to analyze the code of the BitThief client.

The solution is not perfect and partially only works due to security by obscurity. The main problem is the distribution of the shared key. If the key is stored on a publicly accessible server then the key can be regarded also public since anyone could request it of the server, and is therefore useless to improve the security of the algorithm. If additionally the BitThief's source code would be publicly available it would be trivial to implement such a 'BitThief check'. In case the key is hardcoded into the source code, and only the executable is made available it is still possible to find the key in the byte code, and given enough time it should be also possible to reverse engineer the algorithm. This might be not the best solution, but could be only improved with significantly more complex designs, i.e., Trusted Computing. Our solution is hence not completely safe, the effort needed to detect our hidden handshake, however is substantially larger.

Analyzing the protocol, how different clients are working, and finding a way to hide the handshake within was a very interesting task. But the implementation and debugging were far more time consuming than I expected, and did not leave me much time to improve some other wrinkles I have found such as the version in the LTEP handshake. This is a real pity.

Nevertheless, it was a very interesting project, and I was able to profit a lot, especially in handling and working with such a large project, and getting deeper knowledge of Java.

Bibliography

- [1] P. Moor. Free Riding in BitTorrent and Countermeasures, Master Thesis, ETH Zürich, Summer 2006.
- [2] D. Kind. Honour among Thieves, Master Thesis, ETH Zürich, Summer 2008.
- [3] R. Milani. Extending the Functionality of BitThief, Semester Thesis, ETH Zürich, Summer 2008.
- [4] BitTorrent Wiki
<http://wiki.theory.org/BitTorrentSpecification>
- [5] Extension Protocol Proposal
http://bittorrent.org/beps/bep_0010.html
- [6] Partial Seed BEP
http://bittorrent.org/beps/bep_0021.html
- [7] Sending Metadata
http://bittorrent.org/beps/bep_0009.html
- [8] BitTorrent PEX conventions
<http://wiki.theory.org/BitTorrentPeerExchangeConventions>
- [9] Tribbler Distributed Tracker, PEX crawl
<http://www.tribler.org/trac/wiki/PEXCrawl>
- [10] Tribbler Distributed Tracker
<http://www.tribler.org/trac/wiki/DistributedTracker>

Appendix

LTEP Handshakes

Typical decoded Handshake (split for readability):

```
{
  ipv4='xC#a',
  m={upload_only=3, ut_holepunch=4, ut_metadata=2, ut_pex=1},
  metadata_size=27695,
  p=51727,
  reqq=255,
  v='µTorrent 2.0.1',
  yourip='6ì\2'
}
```

This client advertises support for several features, in the m dictionary, with the corresponding header values that should be used for the package. It also sends quit some additional information about itself and also the client getting the package.

Handshake sent by BitThief:

```
{
  m={ut_pex=1}
}
```

The BitThief client on the other hand only advertises the support for ut_pex.