Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Skirting ISP Traffic Shaping in P2P Systems, and Countermeasures

Bachelor's Thesis

Pascal Studerus
studi@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Raphael Eidenbenz
Prof. Dr. Roger Wattenhofer

October 16, 2011

Abstract

The BitTorrent (BT) protocol was already in 2005 a widley used peer-to-peer (P2P) file sharing protocol and produced a high amount of traffic. Some internet service providers (ISPs) began to slow down the BT protocol to provide users more bandwidth for other protocols. The BT community introduced a new protocol, called Message Stream Encryption (MSE) to bypass the limitations of the ISPs. This thesis discusses the structure of the MSE protocol. Furthermore, different shaping methods that could be used by ISPs to detect and slow down BT traffic, are compared with the possible methods for BT clients to hide from the shaping methods. The improvements of the possibilities of ISPs and the BT developers can be regarded as an arms race.

Contents

Chapter 1

Introduction

1.1 Introduction

The BitTorrent (BT) Protocol is a protocol for distributed file sharing and was designed in 2001 by Bram Cohen [21]. He developed it originally for the etree.org community for sharing linux files and presented the code at the hacker convention CodeCon in 2002 [4]. BT was soon used for other purposes, including illegal file sharing, i.e. distributing copyrighted files over local networks and/or the internet. The BitTorrent Protocol creates a peer-to-peer (P2P) network to exchange files. In contrary to the client-server paradigm each node acts as a client and server in the P2P model and hence there is no centralized server. Because there is no hierarchy in the system, each node has the same role and thus we call the nodes the peers.
In 2005, BitTorrent was responsible for more than a third of the total traffic in the internet [21]. This lead to a problem for internet service providers (ISPs), as they had to support more and more data flow through their networks. In order to keep up with the high amount of BitTorrent traffic, ISPs had two options: Either they could increase the capacity of their networks or they could throttle the traffic generated by BitTorrent and other popular P2P protocols. Some ISPs began to shape traffic, i.e. increase or decrease the speed for certain packets through their network. ISPs that shape traffic assign priorities to protocols and slow down or even block traffic with low priority. Due to the high amount of traffic generated by the BT protocol many ISPs that shaped traffic assigned a low priority to BT. As a response, the BitTorrent community used obfuscation and encryption to prevent traffic shaping [2]. The BitTorrent developers proposed a new encryption protocol called Message Stream Encryption (MSE) [6] in 2006. MSE is the standard encryption method for BitTorrent clients today.

Chapter 2

BitTorrent

2.1 BitTorrent Protocol

BitTorrent is a P2P file sharing protocol. A peer can publish information about the data of the file she wants to share by creating a so called torrent file.
In order to download a file with BT, a peer has to download the corresponding torrent or rather torrent metafile from a torrent discovery website, i.e. that is a website that publishes torrents. As a next step the peer has to connect to a P2P network to share her file. To achieve this the peer sends an HTTP get request to a tracker. Urls of trackers are provided in the torrent metafile. The response of the tracker is a BT specific encoded message that has stored IP addresses of other peers in the P2P network. The peer then sets up individual connections to a subset of the peers by sending a BT handshake to these addresses. The receiver checks the message and decides if she uploads and downloads from that peer. If she does, she sends her own BT handshake message. Once a peer has sent her own handshake message and received the other's handshake message, the handshake is completed. BT splits the original file into smaller pieces and provides a hash for each piece in the torrent metafile. Clients use these hashes to verify the integrity of received pieces. The BT protocol incorporates several types of messages, which are used to distribute the pieces of a file among peers. For more information on the protocol see the BitTorrent protocol specification [3].

2.2 Message Stream Encryption

Message Stream Encryption (MSE) was intended to provide payload and protocol obfuscation for data streams, thus making it much more difficult for passive eavesdroppers to identify the BitTorrent protocol. As a positive side effect, MSE also gives the user some confidentiality and limited peer authentication.

2.2.1 MSE protocol structure

The protocol structure consists of a handshake, where the communicating peers decide how to send the BT messages afterwards.
In the following we consider A as the initiator of the handshake and B as the responder. The handshake consists of five steps, whereas in each step the structure of the step itself is presented. The sender is placed on the left side of the arrow, the receiver on the right side of it and the payload is shown in the box.

1) A $\rightarrow$ B : | Y_A | Pad_A |

   A sends its Diffie-Hellman (D-H) [13] public key (Y_A) and a random padding (Pad_A) with a length of 0-512 bytes to B. The group taken for D-H is $\mathbb{Z}_p^*$ where $g = 2$ is the generator, and p is a 768 bit prime. X_a, X_b are the D-H private keys of A and B with recommended length of 160 bits. Thus, the secret D-H key is $S = (g^{Xa})^{Xb} = (g^{Xb})^{Xa}$.
   The padding Pad_A is varied to prevent a detection of the handshake based on particular message lengths.

2) B $\rightarrow$ A : | Y_b | Pad_B |

   As soon as B receives the first parts of A's message, B sends her own D-H public key (Y_b) with an appended random padding (Pad_B) of length 0-512 bytes to A.

3) A $\rightarrow$ B : | hash part | data part |

   The message consists of two parts: a hash and a data part.

   hash part: | H("REQ1"‖ S) | H("REQ2"‖ SKEY) XOR H("REQ3"‖ S) |

   The hash part consists of three hashes. SHA-1 is used to compute the hash for H with an output length of 20 bytes. "REQ1", "REQ2", "REQ3" are predefined strings used by the hash function H. The common shared key (S) and the common shared weak secret (SKEY) are taken as additional inputs for the hashes. SKEY is the torrent info hash of the torrent to be downloaded and can be found in the torrent metafile. The torrent info hash is a secret as it is only provided in the torrent metafile. It is a weak secret, as everyone that has access to the torrent metafile knows it.
   B uses the first hash to synchronize, since the length of Pad_A is unknown.

   data part: | E(VC‖ CryptoProvide‖ L(Pad_C)‖ Pad_C‖ L(IA)) | E(IA) |

   The entire data part is encrypted using RC4 [17] as a stream cipher. RC4 is a binary additive stream cipher that generates a pseudo-random keystream.

This keystream is combined with the plaintext using XOR to create the ciphertext. Therefore it is important to decrypt the ciphertext in the right order and it holds that $E(a\|b) = E(a)\|E(b)$. Hence it's possible to decrypt one part after another of the cipher text. When A sends data, it uses the 20 bytes hash H("keyA"‖ S‖ SKEY) as key for RC4. When A receives data, it uses H("keyB"‖ S‖ SKEY) as key for RC4. Analogously, B uses H("keyB"‖ S‖ SKEY) when sending and H("keyA"‖ S‖ SKEY) when receiving data. The first 1024 bytes of the keystream are discarded to protect against a Fluhrer, Mantin and Shamir attack [18].
VC is a verification constant that is an array of 8 bytes, where each byte is set to 0x00.
The 4 byte field CryptoProvide stores the information which cryptographic methods A supports. For now, the first 3 bytes are all 0x00 and reserved for later use. The last byte is either 0x01 for plain or 0x02 for RC4.
L(x) is a function that maps 2 big-endian encoded bytes to an integer that indicates the length of x.
Pad_C is a zero-valued padding of length 0-512 bytes.
IA is the so-called initial data out. It can be the typically BT handshake message or the empty string.

4) B → A : | E(data part) | E2(Payload Stream) |

The message consists of two parts: a data and payload stream part.

data part: | E(VC‖ CryptoSelect‖ L(Pad_D)‖ Pad_D) |

The entire data part is encrypted using RC4 as stream cipher. The keys for encryption and decryption are the same as described in Step 3.
A uses the verification constant (VC) to resynchronize, as the length of Pad_B is unknown.
The 4 byte field CryptoSelect contains the information which cryptographic method B chooses for the encryption of the message stream after the handshake. Like with CryptoProvide, the first 3 bytes are set to 0x00. The information about the encryption method is contained in the last byte, 0x01 for plain, and 0x02 for RC4.
Pad_D is an additional zero-valued padding of length 0-512 bytes.

E2() is the encryption method on which A and B have agreed in Step 4, plain or RC4. With plain "encryption", messages are sent in plain text without any obfuscation or encryption. With RC4, messages are encrypted, using the same RC4 stream cipher as in Step 3.
Payload stream can be any original message of the BT protocol that B wants to send to A.

5) A → B : E2(Payload Stream)

A uses the encryption method selected by B in Step 4 to encrypt future BT messages.

2.2.2 Early termination conditions

The MSE protocol description provides additional options that if any of the following conditions holds, the handshake can be regarded as failed and the connection is closed. It is not mandatory to check the conditions and therefore it is up to the implementation if the conditions are checked, or ignored. The conditions should be checked before the mentioned step of the MSE protocol has started.

1. Step 2: B terminates the connection if
   - A sent less than 96 bytes within 30 seconds, or
   - A sent more than 608 bytes within 30 seconds.
2. Step 3: A terminates the connection if
   - B sent less than 96 bytes within 30 seconds, or
   - B sent more than 608 bytes within 30 seconds.
3. Step 4: B terminates the connection if
   - the correct S hash can not be found within the first 628 bytes, and thus, B has not found the synchronization point,
   - it received a wrong SKEY hash after the S hash,
   - VC can not be decoded correctly after the SKEY hash,
   - none of the options provided in CryptoProvide are supported, or each byte of CryptoProvide is set to 0x00.
4. Step 5: A terminates the connection if
   - VC can not be decoded correctly within the first 616 bytes, i.e. A has not found the synchronization point, or
   - the option selected in CryptoSelect was not provided in Step 3.

It is up to the next protocol layer, the BT protocol, to terminate the connection if none of the conditions are met.

2.2.3 Tracker peer obfuscation

The tracker peer obfuscation extension [9] is intended to prevent a passive eavesdropper to read the replied peer list from a tracker. The extension is the response to an attack on the BT protocol that reads the peer list to get IP-port pairs. Corresponding to BT users such IP-port pairs could be used to identify peers willing to share files among each other, and the connection between them could be blocked.
In the torrent metafile, a special list with trackers that support the tracker peer obfuscation extension is added. A peer adds new parameters to the standard URL get request indicating what kind of connection it prefers when requesting a peer list from such a tracker. One parameter is whether she supports or requires encrypted connections to other peers. The second parameter is the port on which she accepts BT connections. In such a request, a SHA-1 hash of the info hash is sent rather than the info hash in plaintext. The tracker itself obfuscates the peer list and any other part containing information about peers with RC4. The first 768 bytes of the keystream of RC4 are dropped to prevent a known attack [18]. According to the BitTorrent Enhancement Proposal (BEP) Nr. 0008 [9] the tracker peer obfuscation is deferred. BEPs are used to discuss possible extensions to the BT protocol and standardize them. As it is not an official accepted proposal, only a very small amount of trackers support the extension. Tracker peer obfuscation leads to more overhead for trackers and clients, which could be a reason for it to be only an unofficial extension.

2.2.4 Handle MSE connections

BT clients that support MSE have three possible options how to deal with connections [6]. These options are necessary for such clients, because they can not communicate the desired way with all other clients as not all of them support MSE.

- Allow incoming MSE encrypted and usual BT connections. Only usual outgoing BT connections.
- Allow incoming MSE encrypted and usual BT connections. Outgoing connections are MSE encrypted, usual outgoing BT connections only if a MSE encryption connection attempt failed.
- Only MSE encrypted incoming and outgoing connections.

A fourth option which uses only usual BT connections is omitted as it is the same as to disable MSE. Such an option is regarded as there is no support for MSE.

Chapter 3

Discussion

3.1 When to use MSE

The main reason for a BT user to use encryption is to optimize its speed if the user is traffic shaped by the ISP. While the ISP does not use sophisticated methods to detect BT traffic, using MSE could lead to the same speed as if the user was not traffic shaped.
Using a stream cipher like MSE to encrypt and decrypt any data sent increases the load on a system but provides obfuscation. RC4 was chosen over more secure stream ciphers, because RC4 is less hardware consuming. An additional option using plain instead of RC4 can be selected to reduce the load compared to RC4. But except for the handshake any BT message is sent in plaintext instead of being encrypted, and thus the overhead is in most cases insignificant.
As long as a user has an ISP that does not throttle BT traffic, there is no reason to use encrypted connections to other peers that are not traffic shaped either. This is simply because in this scenario the cryptographic handshake leads to a higher load on the system and a possible drop in the download speed without any advantage. MSE can not be used to prevent any passive eavesdropper to read what is shared between users as it does not provide authenticity nor confidentiality. The MSE protocol is not entirely secure as there exist several weaknesses in it that can be abused to obtain information about the message stream [12].

3.2 Traffic shaping methods

There exist several traffic shaping methods, which are discussed in this section.

Packet inspection

A method to detect BT messages is to check for the BT handshake messages by inspecting sent packets [19, 20]. If such a handshake message is found, an

ISP knows which parties are invloved. The ISP also knows on what ports they communicate and can slow down or block any packets matching that IP-port pair. This shaping method can be avoided by a BT client using MSE. If the RC4 option is used for MSE it is not possible to detect any BT specific byte pattern in messages as all the messages are encrypted. An ISP that still wants to detect MSE using RC4 can not try to break possible MSE connections as it would be too power consuming to break several single connections [12].
Using packet inspection can lead to privacy issues for an ISP, because the ISP has to read packets sent by users, which is not allowed on some network locations.

Flow-level heuristics

A technology for ISPs to detect also encrypted BT traffic are flow based methods using statistical approaches [16, 14, 22]. These methods seem to be able to detect BT traffic with a high efficiency and have a good enough accuracy. Flow-level heuristics are more robust against protocol changes as they detect protocols by its characteristics that do not often change, i.e. how long average messages are and how often messages are sent between two peers. A negative point is that it takes longer to execute the detection algorithm, meaning more messages must be analysed to detect a protocol. A possibility for BT clients to hide from this method is to use flow obfuscation or to hide inside a well known protocol [14].

Intercept peer-to-tracker communication

Trackers are the entry points for any peer to a BT P2P network. The message from a tracker to a peer contains a list of IP/port pairs of peers that are already in the P2P network that the peer wants to join. The ISP can store the IP/port pairs and as soon as the peer sends a message to a pair in the list, the ISP knows that it is a BT connection and can close it. To achieve this an ISP may according to the policy close BT connections between two such peers by sending TCP resets. A company providing hardware tools to block BT and other P2P traffic using this idea is Sandvine [8]. Tracker-peer obfuscation described in 2.2.3 may help to avoid this traffic shaping method.

Bandwith limiting methods

According to the Vuze wiki [11] there are ISPs that slow down traffic if a client has a too high traffic volume or limit the possible traffic of any client at peek times during a day. Against such methods can not even encryption increase the speed of the BT client, and there exist not any other known possibility to bypass bandwith limiting methods yet.

Prioritize traffic

Some ISPs prioritize recognizable traffic and therefore it can happen that hiding BT traffic by obfuscation methods does not increase the speed as the unrecognizable BT traffic also receives a lower priority. An example of a company that prioritizes traffic is pipex [7]. ISPs like pipex give traffic like HTTP a higher priority whereas P2P traffic receives a lower priority. Such ISPs regard HTTP and the like as sensitive traffic, because you immediately recognize if the speed is reduced in contrast to P2P traffic. These ISPs state that at peek times, where the used bandwith is close to its maximum, sensitive traffic should be prioritized over unsensitive traffic. It also should not hurt P2P systems that much if P2P systems can use their full speed during times where the bandwith of an ISP is not fully used. Due to the higher priority of HTTP traffic, some developers try to hide BT traffic as HTTP traffic. But even this method seems to be detectable [5].

3.3 Conclusion

MSE was designed to provide peer obfuscation but not confidentiality or authentication. Therefore it should not be used to hide any secret data from an adversary, as several attacks exist to break the protocol. But these attacks are power consuming, hence these attacks can not be used by ISPs to shape BT traffic.
How useful MSE is in practice can not be clearly stated. It seems like statistical approaches can detect MSE well enough to be used by ISPs to shape MSE encrypted traffic. The best approach for an ISP is to combine packet inspection with statistical approaches [16]. Packet inspection is faster, but works only against non-encrypted traffic, whereas statistical approaches can be used if packet inspection failed. A problem that remains is false positives, i.e. detecting non-BT traffic as BT traffic by mistake. ISPs have to decide what their priority is. Either they detect less undesired traffic with a lower amount of false positives, or they detect more undesired traffic with a higher amount of false positives.
A user having an ISP shaping BT traffic to lower his BT speed has to try several methods provided by the used BT client. This is required because she does not know which shaping techniques are used by her ISP. If the user's BT client supports any method to hide BT traffic that the ISP does not have a detection for, the user can increase her speed. A user has to try all possibilities, as there are many ways to hide BT traffic, similarly many ways to identify BT traffic. It is like an arms race [15] where each side (ISP versus P2P) tries to improve its methods to come out ahead. According to the seen traffic shaping methods, it seems like if an ISP really wants to block BT traffic using any available shaping method, the ISP can do it with a good accuracy.

Chapter 4

Implementation

4.1 BitThief

During my Bachelor thesis I implemented Message Stream Encryption support for the ETH BitTorrent client BitThief [1] without the tracker peer obfuscation. There are four options concerning encrypted connections:

- No encrypted connections: Normal BT handshake is used for outgoing and incoming connections.
- Encrypted connections if needed: Outgoing connections are tried to establish with the normal BT handshake. If such a connection attempt failed MSE is used to try to establish a connection to that peer. Normal BT and MSE incoming connections are accepted.
- Encrypted connections if possible: Outgoing connections are tried to establish with MSE. If such a connection attempt failed a normal BT connection is tried to establish to that peer. Normal BT and MSE incoming connections are accepted.
- Encrypted connections only: MSE is used for outgoing and incoming connections.

4.1.1 Notes

I read and analyzed the MSE implementation of the open source BitTorrent client Vuze [10]. As the Vuze implementation mechanics differ a lot from BitThief, only a few classes could be taken from Vuze. The rest of the code was written specific to BitThief.
BitThief supports only RC4 as encryption method for the payload stream after the cryptographic handshake is completed. As described in 2.2.1 the RC4 algorithm depends on already encoded or decoded code. Following the correct order

of encoding and decoding must be met to be compatible to other BT clients. The first 1024 bytes of the keystream of the RC4 stream cipher are discarded to protect against a possible attack [18], as discussed earlier.

Bibliography

[1] *BitThief, A Free Riding BitTorrent Client.* bitthief.ethz.ch (retrieved on Oct. 9th, 2011)

[2] *BitTorrent protocol encryption, Wikipedia.* http://en.wikipedia.org/wiki/BitTorrent_protocol_encryption (retrieved on Oct. 9th, 2011)

[3] *BitTorrent protocol specifications.* http://wiki.theory.org/BitTorrentSpecification (retrieved on Oct. 9th, 2011)

[4] *CodeCon, Wikipedia.* http://en.wikipedia.org/wiki/CodeCon (retrieved on Oct. 9th, 2011)

[5] *Detect BT traffic encrypted as web traffic.* http://www.studlife.com/archives/News/2006/09/27/ResTechsolvesnetworkissues/ (retrieved on Oct. 9th, 2011)

[6] *Message Stream Encryption.* http://wiki.vuze.com/w/Message_Stream_Encryption (retrieved on Oct. 9th, 2011)

[7] *Pipexuk.* http://www.pipexuk.com/terms/broadband_terms_2.html (retrieved on Oct. 9th, 2011)

[8] *Sandvine homepage.* www.sandvine.com (retrieved on Oct. 9th, 2011)

[9] *Tracker-peer obfuscation BEP.* http://bittorrent.org/beps/bep_0008.html (retrieved on Oct. 9th, 2011)

[10] *Vuze, open-source BitTorrent client.* http://www.vuze.com (retrieved on Oct. 9th, 2011)

[11] *Vuze wiki.* wiki.vuze.com (retrieved on Oct. 9th, 2011)

[12] J. Valkonen and B. Brumley. Attacks on Message Stream Encryption, 2009.

[13] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory, vol. IT-22, no. 6, pp. 644-654*, Nov. 1976.

[14] W. John and E. Hjelmvik. Breaking and Improving Protocol Obfuscation, 2010.

[15] D. Sicker, K. Bauer and D. Grunwald. The Arms Race in P2P, 2009.

[16] D. Peleg, L. Roditty, R. Bar-Yanai and M. Langberg. Realtime Classification for Encrypted Traffic, 2010.

[17] R. Rivest. *The RC4 Encryption Algorithm.* RSA Data Sec, Inc., 1992.

[18] A. Shamir, S. Fluhrer and I. Mantin. Weaknesses in the Key Scheduling Algorithm of RC4, 2001.

[19] D. Wang, S. Sen and O. Spatscheck. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures.

[20] A. Broido, T. Karagiannis and M. Faloutsos. Filesharing in the Internet: A characterization of P2P traffic in the backbone. Technical report, University of California, Riverside Department of Computer Science, November 2003.

[21] Clive Thompson. *The BitTorrent Effect.*
http://www.wired.com/wired/archive/13.01/bittorrent.html,
January 2005. (retrieved on Oct. 9th, 2011)

[22] Arno Wagner, Thomas Dübendorfer, Lukas Hämmerle and Bernhard Plattner. Flow-based Identification of P2P Heavy-Hitters.

Appendix A

Appendix Chapter

A.1 Modified classes

ArrayUtil.java
BitThiefConfiguration.java
BitTorrentConnection.java
ConnectionCloseReason.java
ConnectionInitiator.java
GlobalSettingsWindow.java
ListUtil.java
PacketFactory.java
Packetizer.java
PacketSocket.java
PacketSocketImpl.java
PeerManager.java
PeerManagerTest.java
PeerRecord.java
T4TConnection.java
T4TPacketSocket.java
TrackerClientTest.java
TrackerRequest.java
TrackerRequestBuilder.java
TrackerRequestTest.java
TrackerResponseTest.java

A.2 Added classes

ArbitraryPacket.java
Base32.java
ByteFormatter.java
CryptoConnection.java

CryptoPacket.java
CryptoPacketFactory.java
CryptoPacketizer.java
CryptoPacketSocket.java
DHPubKeyPacket.java
EncryptionSettingsWindow.java
HashPacket.java
HashWrapper.java
IALengthPacket.java
IAPacket.java
InfoPacket.java
RandomUtils.java
SHA1.java
SHA1Hasher.java
TransportCipher.java