# Efficient BitTorrent Handshake Obfuscation

Thomas Zink Marcel Waldvogel
Distributed Systems Group
University of Konstanz
78457 Konstanz, Germany
*first.last*@uni.kn

## Categories and Subject Descriptors
C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

## General Terms
BitTorrent

## Keywords
P2P, BitTorrent, traffic obfuscation, traffic hiding

## ABSTRACT
During the last decade, large scale media distribution populated peer-to-peer applications. Faced with ever increasing volumes of traffic, legal threats by copyright holders, and QoS demands of customers, network service providers are urged to apply traffic classification and shaping techniques. These highly integrated systems require constant maintenance, introduce legal issues, and violate both the net neutrality and end-to-end principles.

Clients see their freedom and privacy attacked. Users, application programmers, and even commercial service providers laboriously strive to hide their interests and circumvent classification techniques. While changing the network infrastructure is by nature very complex, and it reacts only slowly to new conditions, updating and distributing software between users is easy and practically instantaneous.

We present a new obfuscation extension to the BitTorrent protocol, which allows signature free handshaking. The extension requires no changes to the infrastructure and is fully backwards compatible. With only little change to client software, contemporary classification techniques can be rendered ineffective.

## 1. INTRODUCTION
The appearance of Peer-to-peer networks initiated the age of large scale multimedia and binary distribution over the networks. Currently, BitTorrent is one of the most prominent application protocols in use and responsible for large volumes of traffic.

Network providers see the use of P2P applications as a form of denial of service (DoS). It demands vast amounts of resources and threatens the quality of other services (QoS). Furthermore, copyright holders impose responsibility for transmitted content to network providers. As a result, network providers apply traffic classification and shaping techniques to satisfy demands for availability, security and QoS.

Those systems usually consist of a combination of Deep Packet Inspection (DPI) and some sort of statistical or behavior analysis. DPI matches packet contents to known protocol signatures. It is very expensive and prone to obfuscation and encryption but works optimal on well known signatures. Statistical /behavior analysis uses statistical information to evaluate the behavior of interesting flows or hosts and provides estimates about possible application layer protocols. Though accurate results cannot be guaranteed, it is possible to identify obfuscated and encrypted protocols. Traffic classification systems are highly integrated into the network infrastructure and cannot be changed easily. Updating signatures for identification is retroactive and requires continuous monitoring and analysis of communication protocols.

Clients have legitimate interest in hiding their intentions. Users and application programmers go to great lengths to obfuscate their traffic and data. Randomizing ports and data encryption are common methods. In addition, multiple services – both free as well as commercial – have surfaced that aim to improve privacy and anonymity.

We propose a new obfuscation extension that makes use of a globally shared secret to hide the infamous BitTorrent handshake. It is easy to implement, backwards compatible and does not require any changes to the BitTorrent infrastructure.

Only a few changes to the client's source code are necessary to effectively hide the BitTorrent traffic and to circumvent contemporary identification mechanisms while still maintaining compatibility to unmodified clients.

## 2. RELATED WORK

Traffic classification is a vast and heterogenous field with many different methodologies, applications and granularities. A study by Caida [15] reviews numerous papers that span over a decade of research. In general, the goal is to either specifically identify the layer 7 protocol, or to perform a coarse-grain classification according to some pre-defined categories. Methods are packet-based and flow-based, and concentrate on port matching, DPI and analysis of flow characteristics. Many of the proposed methods target specific applications. As a result contemporary classification systems utilize multiple stages and a combination of different methods. Extensive overviews of state-of-the-art traffic classification can also be found in [8, 12, 5, 11].

BitTorrent provides Message Stream Encryption (MSE)/ Protocol Encryption (PE). MSE utilizes a Diffie-Hellman-Merkle (D-H) key exchange [14] to negotiate and establish an encrypted connection. Peers try to establish an encrypted connection by initiating the key exchange. If it fails, they revert back to plain data transmissions. Otherwise the peers negotiate encryption and perform the BitTorrent handshake. Both peers can append random padding to the initial key exchange messages. MSE/PE has been subject to heated debate [13] and is fairly complex to implement. As a result, not all clients support encrypted connections.

A D-H key exchanged is also used for eDonkey protocol encryption [1, 2]. Skype uses RC4 to encrypt signaling traffic while the actual VoIP packets between peers is encrypted using AES [3, 7]. Furthermore, Skype can use different transport protocols and codecs to dynamically adjust to different network environments [4].

Protocol obfuscation is usually limited to packet content. Flow features are rarely or insufficiently disguised. In [11] the authors show, that sophisticated statistical methods can identify obfuscated and encrypted protocols with over 90% accuracy. The SPID algorithm (Statistical Protocol IDentification) computes session fingerprints based on "meters" and compares them to pre-learned protocol models. A multitude of different "meters" can be trained and included independently which adds a lot of complexity but also flexibility to the identification process. The authors conclude that contemporary protocol obfuscation is not able to hide traffic from statistical classification due to relatively strong flow characteristics. They suggest concealing flow features using randomized flushing, random padding and random changes of flow directions. We take this observation as a starting point to improve BitTorrent obfuscation.

## 3. OBFUSCATING BITTORRENT

The BitTorrent protocol specification [6] defines the Peer Wire Protocol, which is used to establish connections and exchange messages. It requires the peers to do an initial BitTorrent handshake directly following the TCP handshake. This BitTorrent handshake (Figure 1) uses a fixed string signature at a fixed position that is easily detectable using simple string matching. Contemporary DPI systems like openDPI [1] or l7-filter [2] simply compare the first 20 bytes to the pattern "0x13BitTorrent protocol" to identify BitTorrent traffic.

[1]`http://www.opendpi.org/`, retreived 12.01.12

[2]`http://l7-filter.sourceforge.net/`, retreived 12.01.12

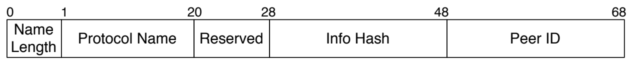

**Figure 1: BitTorrent handshake message. *Name Length* equals "0x13", *Protocol Name* equals "BitTorrent protocol". The *Reserved* field is used to signal extension support. *Info Hash* is the sha1 hash of the torrent file's info hash value. *Peer ID* is a client's random peer ID**

*MSE* is now the de-facto standard for encrypted BitTorrent connections. However, [11] has shown, that even *MSE* can be detected due to characteristics of the key exchange and too little variance in padding implementations. While the lack of padding can easily be fixed, *MSE* suffers from high complexity and impact on cpu usage as well as bandwidth.

Following [11], we analyze how BitTorrent obfuscation can be improved. We propose an obfuscation extension that consists of multiple independent techniques that address both the payload data as well as flow features. First we use the globally known *Info Hash* and the target peer's *Peer ID* to obfuscate the first 28 bytes of the handshake. Thus, the packet contents appears random to an observer, defeating DPI systems. Second, we introduce a new message type called *Padding Message*, which allows injection of a random number of random bytes into BitTorrent conversations. This raises variance in packet lengths and payload values and thus increases the difficulty of statistical fingerprinting. We also discuss the applicability of random flushing and random changes of flow direction.

Note that the proposed techniques do only address peer-to-peer communication and not tracker traffic. As a result it does not protect against the so called Sandvine attack. Tracker-to-peer traffic obfuscation that addresses the Sandvine attack problem has been proposed in [10].

### 3.1 Obfuscated Handshaking

Since the BitTorrent handshake is the most discriminating feature of the BitTorrent protocol, concealing the handshake messages itself will nullify the effect of most DPI systems. We propose an obfuscation technique, which is easy to implement and introduces very little overhead to cpu and bandwidth usage.

In the obfuscated handshake message the fields `Name Length`, `Protocol Name` and `Reserved` are obfuscated by using the sha1 hash value of the target peer's "peer ID" concatenated with the shared "info hash". Since sha1 produces a 20 byte hash value, we actually use two hashes to produce 28 bytes. Figure 2 shows the procedure.

The target peer ID and the info hash are concatenated twice in altering sequence to be used as input for two sha1 hash operations. Note, that this scheme requires the peer not to support the compact peer list extension [9] which allows trackers to return a more compact peer list, excluding the peer ID. The target peer ID is used to prevent an observer

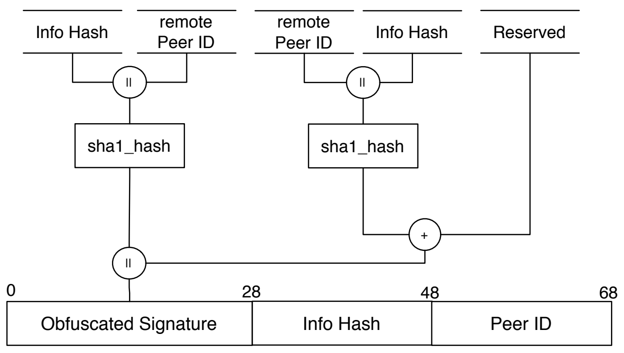

**Figure 2: Obfuscation of the handshake message. The fields *Name Length*, *Protocol Name* and *Reserved* are replaced with sha1 hash values of the *Info Hash* concatenated with the remote peer ID. Concatenation is denoted by (||), XOR is denoted by (+)**

from extracting all decoding information from the actual message. An observer would need to associate IPs and ports with peer IDs from tracker responses to derive the input for the hash function. The first hash value is used to disguise the `Name Length` and `Protocol Name` fields. Eight bytes of the second hash value are XOR'ed with the `Reserved` value to produce a new one. This introduces a higher variance in byte values to increase difficulty for statistical fingerprinting. Sha1 is used as hash function since it is already used in BitTorrent. The fields `Info Hash` and `Peer ID` remain unchanged.

Though the handshake itself appears to be random, it still shows unique features that can be exploited for identification. The handshake message is always 68 bytes long and both request and response have the same info hash value. That is, bytes 28 to 47 are equal in two consecutive packets with opposite directions. In addition, incoming handshake messages will have equal byte values in the first 48 bytes for each unique {info hash, peer ID} pair. The fields `Info Hash` and `Peer ID` will have equal values in every handshake message (either as request or response) that is related to the same torrent file and sent from the same peer.

## 3.2 Random Padding

[11] suggests using random padding to conceal flow features. In case of *MSE*, padding is also used during key exchange, although the implemented padding length is insufficient to provide enough variance in packet lengths [11]. Introducing random padding to the handshake solves the length issue. We believe that the equal info hash is not really a problem, since an observer needs to keep state and payload data to do the actual comparison. The same applies for the {info hash, peer ID} pairs. The observer needs to keep variance of the first 48 bytes of inbound handshake messages. These messages can appear on different ports and will differ in case the peer shares multiple files. There are possible solutions like concealing the info hash and including the source peer ID in the hashing process. To maintain compatibility with other clients, peers supporting the obfuscated handshake extension first try to connect using an obfuscated handshake,and if this fails, they fall back to standard handshake.

BitTorrent's specification also specifies messages according to the type-length-value (TLV) standard. Specifically, messages are encoded as `<length> <type> <payload>`. The first integer denotes the length followed by one byte which denotes the type followed by the message payload. Nine messages are specified, some clients also implement a tenth message.

We suggest a new message type called padding with a randomly chosen length and random payload. It can be appended to conversations at all time, however, is especially encouraged during handshaking which makes an early identification using statistical fingerprints extremely difficult. The message is depicted in Figure 3.

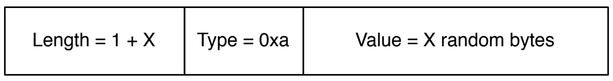

**Figure 3: Padding message. A random number of random bytes. Introduces randomness in both packet length as well as content value which makes statistical fingerprinting difficult**

Whenever the peer decides to use padding, the message is randomly generated and can be appended to any data due to be sent. We suggest using up to 1400 bytes to provide randomness close to the MTU. One downside of the message scheme is that the first two bytes in the length field will always be zero, while message type will always equal $0xa$.

## 3.3 Magic Peer ID

The downside of using obfuscation is additional overhead due to reconnects if the connection is severed. The probability of this happening can be vastly reduced by encoding obfuscation support in the chosen peer ID. Since the peer IDs are announced by the tracker, peers in the swarm can easily determine which fellow peers support obfuscation. To indicate obfuscation support, the peer ID is chosen such that its sha1 hash value shows a specific 2-byte value at a predefined position. A peer that wants to connect to another peer first hashes the target peer's peer ID and checks the hash value for the predefined 2-byte indicator. On a match, the target peer supports obfuscation with high probability. Following we will refer to such a peer ID as "magic" and to a peer that has a magic peer ID as "magic peer". The process requires continuous random generating and testing of peer IDs during startup. However, computation is cheap and should on average not take more than $2^{16}$ tries. Naturally there is a margin of error if a peer chooses a magic peer ID by accident without supporting the obfuscation extension. Note, that the "magic" peer ID is not a requirement, nor a guarantee for obfuscation support. It is a means to reduce the probability of reconnects.

## 3.4 Random Flush and Packet Direction

[11] proposes random flushing and changes in packet directions to further hide flow features. However, random flushing of TCP streams cannot easily be performed since it requires disabling Nagle's algorithm, a procedure that might

not be possible on all systems. It would also impact overall TCP performance.

The BitTorrent specification states, that "peer connections are symmetrical [...], and data can flow in either direction." [6]. Both the seeders and leechers can and do initiate communication. However, peers that open a connection always initiate key exchange and handshake. This could be exploited for identification.

There is a simple reason for this behavior. The connecting peer wants to share a specific file identified by the info hash. Since peers can share many files, the target peer cannot know for which info hash the connection is established prior to the handshake. Theoretically, either peer could initiate the handshake or key exchange. This would, however, require the peers to keep dictionaries of IP, peer ID and info hash pairs and also a method to guarantee that all files are equally shared.

Considering all the problems, flushing the TCP stream is impractical, and changing the direction of the key exchange and handshake requires significant effort.

## 4. EVALUATION

We implemented our obfuscation scheme in *ttorrent 1.0.4* [3], a lightweight java BitTorrent library that is easily extensible. It is an all in one solution, providing an API for clients, trackers, and torrent files alike. We also implemented a reference application in c as baseline and for testing purposes. It only provides the absolutely necessary functions and has about 100 lines of code. The basic implementation into *ttorrent* required changes to less than 200 lines of code.

We used Planet-Lab[4] as a testbed and deployed our modified *ttorrent* on 70 nodes which all initially acted as leechers. The tracker and initial seeder ran on dedicated machines and tracked a single file. We captured all traffic on the initial seeder's interface using *tcpdump*[5], which mimics the capabilities and view of the network access provider. In addition, we also tested interaction with a variety of popular BitTorrent client software and opentracker[6], which is the mostly used tracker software world wide.

We then analyzed the captured traffic using OpenDPI[7] and SPID[8]. OpenDPI is Ipoque's open source version of its commercial PACE engine and a state-of-the-art DPI representative, while SPID is a statistical protocol identification algorithm, which uses trainable protocol models that can dynamically be applied to identify 17 protocols. In both cases we did not apply any optimizations but used the provided demo applications with default settings. Only in case of OpenDPI did we implement per flow output, to verify and compare the results with SPID that outputs per flow results by default.

[3]`http://turn.github.com/ttorrent/`, retreived 12.01.12

[4]`http://www.planet-lab.org`, retreived 12.01.12

[5]`http://www.tcpdump.org/`, retreived 12.01.12

[6]`http://erdgeist.org/arts/software/opentracker/`, retreived 12.01.12

[7]`http://www.opendpi.org/`, retreived 12.01.12

[8]`http://sourceforge.net/projects/spid/`, retreived 12.01.12

### 4.1 Client Compatibility

To test communication of magic peers with ordinary peers we introduced popular clients to the swarm, including Vuze, uTorrent, Transmission, rtorrent, and libtorrent-rasterbar. This set covers the vast majority of BitTorrent clients in the wild today and is a decent representation of actual swarms.

All clients could successfully participate in the obfuscated swarm. We also forced random magic clients to initiate an obfuscated conversation regardless of the target peer's peer ID to test the effect of fast reconnects. As expected, the obfuscated handshake was rejected but the subsequent plain handshake was accepted at all times. We did observe random attempts to perform an obfuscated handshake with Transmission clients. Apparently, the chosen peer ID had magic properties.

We also tested the behavior of a magic peer participating in an ordinary swarm. Again, the magic peer behaved as expected and could communicate, download, and seed in the ordinary swarm. We encountered no connection problems or noticeable delays.

### 4.2 Magic Peer ID generation

We generated $100k$ magic peer IDs to analyze the number of random tries needed to find a magic peer ID.

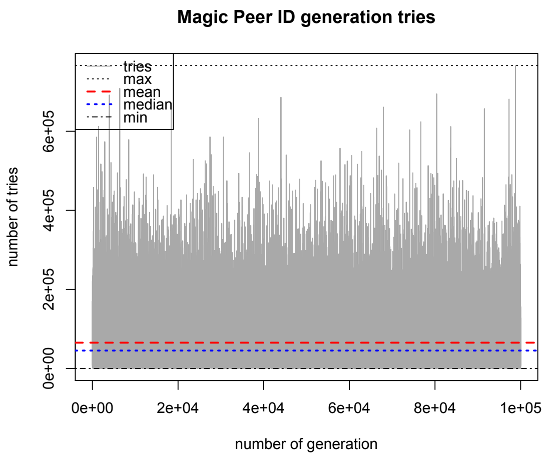

**Figure 4: Number of tries to generate the magic Peer ID**

Figure 4 shows the number of tries needed to generate the magic Peer IDs. The maximum number of tries is pretty high $766,152$. In the worst cases, the delay introduced by searching for a magic peer ID might be enough to be recognizable and annoying to the user. However, the Peer ID is generated only during startup and then remains constant for the remainder of the session. As expected, on average the client needed roughly $2^{16}$ tries to find a suitable magic peer ID.

### 4.3 Handshake message

We extracted all handshake messages from the recorded traffic to analyze them for randomness and traits that could be exploited for identification.

Figure 5 shows the standard deviation and mean of the handshake messages' byte values. It can clearly be seen that the standard deviation of bytes 28 to 47 (info hash) as well as bytes 68, 69 , 72 (message header) are all zero. In addition, standard deviation is also only medium for the first 28 bytes.

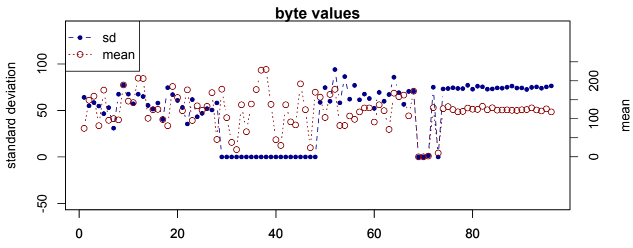

**Figure 5: SD and mean of handshake message bytes**

Since we only shared a single file, the info hash is equal in all of the recorded handshakes, which will not be the case for peers sharing more then one file. The number of different info hashes is, however, equal to the number of files shared.

The message header's length field is an integer. The actual length is lower or equal to 1,400 bytes. The first two bytes of the header (offsets 68 to 69) will therefore always be 0, while the type field (offset 72) will always be 0xa. While three bytes might not be enough to reliably identify the handshake message, they introduce a reoccurring pattern that, combined with other traits, can be exploited.

The low standard deviation in the first 28 bytes is again a result of sharing only one file. Inbound handshakes will always have the same obfuscated signature for equal info hashes. The first 48 bytes of incoming handshake messages will be equal for any source sharing the same file.

### 4.4 Identification

The results of openDPI and SPID identification are shown in Table 1. Both, DPI as well as SPI, were not able to detect and identify the obfuscated BitTorrent traffic. OpenDPI identifies 29 flows as BitTorrent. These are all seeder to tracker announces, which OpenDPI recognizes by searching for the string `GET ` and then parsing the whole tracker request message. These flows are equivalent to the 29 HTTP flows detected by SPID. Tracker requests look similar to HTTP get requests and SPID was not able to distinguish between these two. In fact, in most cases SPID reported higher similarities of the detected flows to eDonkey and ISAKMP than to BitTorrent.

| Algorithm | total flows | Identified | |
|---|---|---|---|
| | | protocol | number flows |
| OpenDPI | 403 | unknown | 374 |
| | | BitTorrent | 29 |
| SPID | 389 | unknown | 360 |
| | | HTTP | 29 |

**Table 1: Identification results for OpenDPI and SPID**

This shows, that techniques using payload information and even flow features, can easily be fooled by disguising the traffic with properties similar to other well known protocols like HTTP or ISAKMP.

## 5. CONCLUSION AND FUTURE WORK

We presented a novel BitTorrent obfuscation scheme that is easy to implement, is backwards compatible, and fairly efficient. It circumvents contemporary identification tools and thus effectively hides BitTorrent traffic. Only small changes in applications require significant effort to update the network infrastructure in order to maintain identification. Even state-of-the-art statistical identification tools can easily be fooled and circumvented by applying basic payload and flow feature obfuscation.

Future work will explore possibilities to introduce more randomness into the obfuscation process, apply more sophisticated flow obfuscation, and to disguise the traffic to appear as produced by other protocols.

## 6. REFERENCES

[1] Edkobfuscation. online. retrieved 2012.01.12.

[2] emule protocol obfuscation. online. retrieved 2012.01.12.

[3] T. Berson. Skype security evaluation. online, 2005. retrieved 2012.01.12.

[4] P. Biondi and F. Desclaux. Silver needle in the Skype. Presentation at *BlackHat Europe*, `http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf`, Mar. 2006.

[5] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok. A survey on internet traffic identification. *Communications Surveys & Tutorials, IEEE*, 11(3):37–52, Aug. 2009.

[6] B. Cohen. The bittorrent protocol specification. online, 01 2008. retrieved 2012.01.12.

[7] D. Fabrice. Skype uncovered. online, 2005. retrieved 2012.01.12.

[8] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15:24–32, 2001.

[9] D. Harrison. Tracker returns compact peer lists. online, 2008. retrieved 2012.01.12.

[10] D. Harrison, A. Ciani, A. Norberg, and G. Hazel. Tracker peer obfuscation. online, 01 2008. retrieved 2012.01.12.

[11] E. H. W. John. Breaking and improving protocol obfuscation. Technical Report 123751, Chalmers University of Technology, 2010.

[12] D. E. Taylor. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.*, 37(3):238–275, 2005.

[13] various. Debate over protocol encryption. online, 2007. retrieved 2012.01.12.

[14] Vuze. Message stream encryption. online. retrieved 2012.01.12.

[15] M. Zhang, W. John, K. Claffy, and N. Brownlee. State of the art in traffic classification: A research review. In *PAM '09: 10th International Conference on Passive and Active Measurement, Student Workshop*, 2009.