BitTorrent Traffic Obfuscation: A Chase towards Semantic Traffic Identification

Thomas Zink Marcel Waldvogel
Distributed Systems Group
University of Konstanz
first.last@uni-konstanz.de

Abstract—**With the beginning of the 21st century emerging peer-to-peer networks ushered in a new era of large scale media exchange. 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 systems usually are highly integrated to satisfy the harsh restrictions present in network infrastructure. They require constant maintenance and updates. Additionally, they have legal issues and violate both the net neutrality and end-to-end principles.**

On the other hand, clients see their freedom and privacy attacked. As a result, users, application programmers, and even commercial service providers laboriously strive to hide their interests and circumvent classification techniques. In this user vs. ISP war, the user side has a clear edge. While changing the network infrastructure is by nature very complex, and only slowly reacts to new conditions, updating and distributing software between users is easy and practically instantaneous.

In this paper we discuss how state-of-the-art traffic classification systems can be circumvented with little effort. We present a new obfuscation extension to the BitTorrent protocol that 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 are rendered ineffective. We argue, that future traffic classification must not rely on restricted local syntax information but instead must exploit global communication patterns and protocol semantics in order to be able to keep pace with rapid application and protocol changes.

I. Introduction

The appearance of peer-to-peer networks started the age of large scale multimedia and binary distribution over the networks. BitTorrent has become one of the most prominent application protocols in use and is responsible for large volumes of traffic. Copyright holders, however, try to impose responsibility about transmitted content to network providers. Large traffic volume, especially when crossing network boundaries, means higher cost. Since P2P applications demand vast amount of resources and threaten the quality of other services (QoS) Network providers feel urged to see their use as a form of denial of service (DoS). To satisfy demands for availability, security and QoS, and to ease the displeasure of copyright holders, the network providers apply traffic classification and traffic shaping techniques. Users on the other hand have legitimate interest in hiding their intentions. As a result, 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. In this paper we show how only a few changes to the client's source code allows effective hiding of BitTorrent traffic.

Contemporary traffic classification systems usually consist of a combination of Deep Packet Inspection (DPI) and some sort of statistical or behavior analysis. DPI is a very expensive task both in space and time that only works on well-known signatures. It is prone to obfuscation and encryption but works reliably on plain text packets. Statistical/behavior analysis uses statistical information to evaluate the behavior of interesting flows or hosts and give estimates about possible application layer protocols. Methods range from simple port matching to bayesian analysis and other machine learning techniques. Though accurate results cannot be guaranteed, it is possible to identify obfuscated and encrypted protocols. The more accurate the results have to be, the more expensive and sophisticated behavior analysis gets.

Updating signatures and fingerprints for identification is retroactive and requires continuous monitoring and analysis of communication protocols. While this poses no significant problem for low-speed software-based classification systems, it is extremely difficult and resource intensive for high-speed classification hardware. Since traffic identification has to be done on wire-speed these systems are highly integrated into the network infrastructure and cannot be changed easily. A change of rules usually results in a change of hardware that needs to be verified and distributed. While changing network infrastructure hardware is difficult the distribution of updated P2P client software is trivial.

We propose a new obfuscation extension that aims to hide the infamous BitTorrent handshake. Our approach makes use of a globally shared secret to encrypt the payload and applies flow obfuscation techniques to obfuscate flow features. Thus it targets both signature-based DPI as well as statistical classification systems. It is easy to implement, backwards compatible and does not require any changes to the BitTorrent infrastructure. It circumvents contemporary identification mechanisms, both signature-based as well as statistics-based, while still maintaining compatibility to unmodified clients. The goal is not to provide a high degree of privacy, rather to demonstrate that minimal effort in protocol design requires significant changes in traffic classification systems.

II. Related Work

Traffic classification is a vast and heterogenous field with many different methodologies, applications and granularities. A study by Caida [32] 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, e.g. peer-to-peer, VoIP, streaming or standard. Methods are packet-based and flow-based, and concentrate on port matching, DPI, and analysis of flow characteristics (see Figure 1). Many of the proposed methods target specific applications or application domains and are neither designed to nor capable of identifying all protocols. As a result, contemporary classification systems utilize multiple stages and a combination of available methods. Extensive overviews of state-of-the-art traffic classification can also be found in [20], [28], [17], [24].

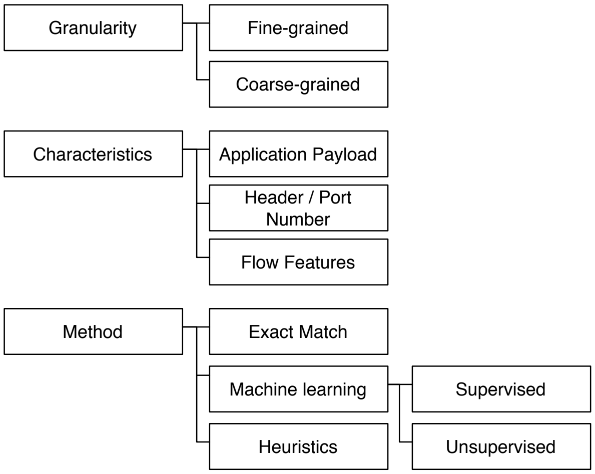

Fig. 1. Taxonomy of internet traffic classification as analyzed and proposed by Zhang *et al.* [32]

Encryption is an effective way of protecting the traffic from unauthorized observation. It usually requires the parties to exchange some key information and negotiate an encrypted connection. In case of BitTorrent, Message Stream Encryption/Protocol Encryption (MSE/PE), provides a means to encrypt traffic between supporting peers. MSE utilizes a Diffie-Hellman-Merkle (D-H) key exchange [30] to negotiate and establish an encrypted connection. Supporting peers first try to establish an encrypted connection by initiating the key exchange. If it fails, they revert back to unencrypted handshake and send plain data. Otherwise the peers negotiate encryption and afterwards perform the (encrypted) BitTorrent handshake to identify the protocol. This is required since the D-H key exchange is generally independent of the application protocol. Both peers can append random padding to the initial key exchange messages, which provides more variance in packet sizes.

MSE/PE has been subject to heated debate [29], cannot be considered standard and is fairly complex to implement. As a result, not all clients support encrypted connections. To the best of our knowledge there are no clients in the wild that actually implement MSE correctly as defined by the proposed standard.

A D-H key exchange is also used for eDonkey protocol encryption [3], [4]. Skype uses RC4 to encrypt signaling traffic while the actual VoIP packets between peers is encrypted using AES [15], [19]. Furthermore, Skype can use both TCP and UDP as well as a range of different codecs to transport and encode the VoIP messages and dynamically adjust to different network environments [16]. The authors of [31] give an overview of VoIP identification and possible countermeasures.

Protocol encryption/obfuscation only provide privacy for the specific protocol or application. Anonymity networks like TOR [13] or I2P [5] offer anonymity and privacy on the network layer. Practically any application can be routed through the anonymity network. The data is usually routed through multiple proxies and encrypted multiple times on its way through the network. The demand for anonymity and privacy is high enough to justify commercial services like ItsHidden [6] and BTGuard [2] that offer anonymous network infrastructures for paying customers. However, the downside of anonymity networks is decreased network performance due to extensive rerouting and encrypting. In addition, anonymity networks might target or exclude specific applications like peer-to-peer or prevent connections with peers that are not part of the anonymity network.

With the exception of Skype, protocol obfuscation is usually limited to packet content. Flow features are rarely or not sufficiently disguised. In [11], [24] the authors show that sophisticated statistical methods can identify obfuscated and encrypted protocols with over 90% accuracy. Their SPID algorithm (Statistical Protocol IDentification) computes session fingerprints based on “meters” and compares them to pre-learned protocol models. SPID provides a multitude of different “meters” which can be trained, configured and included independently. Some meters are designed for, and only effective on specific protocols. This adds a lot of complexity but also flexibility to the identification process. The authors conclude that contemporary protocol obfuscation is not sufficient to hide traffic from statistical classification due to relatively strong flow characteristics. Based on their evaluation the authors suggest three protocol design improvements to bypass traffic classification and shaping (see Figure 2). In addition to payload obfuscation they suggest concealing flow features using 1) randomized flushing of data streams 2) random padding and 3) random changes of flow directions. We take this as a starting point to improve BitTorrent obfuscation.

III. Obfuscating BitTorrent

Following [24], we discuss how their proposed design suggestions can be applied to BitTorrent, and show how small changes allow effective payload and flow obfuscation. We propose an obfuscation extension, which consists of multiple,

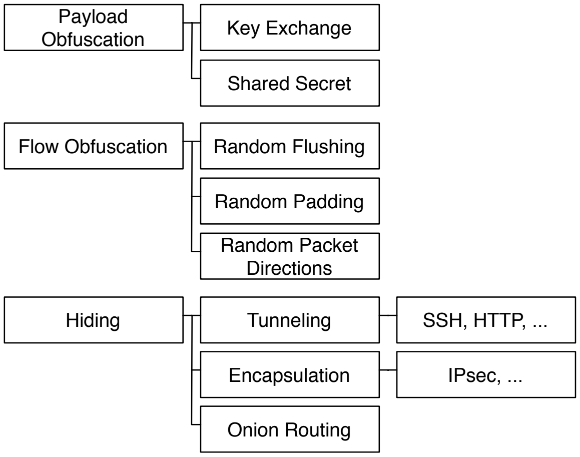

Fig. 2. Effective Traffic obfuscation as suggested by John *et al.*[24]

independent techniques that address both the payload as well as flow features. It is easy to implement and introduces very little overhead to cpu and bandwidth usage. First, we use a *Shared Random Secret* (section III-A) and the target peer's *Peer ID* to obfuscate the handshake message payload. Thus, the packet contents appear random to an observer, defeating DPI systems. Second, we introduce a new message type, called *Padding Message* (section III-C), which allows injection of a random number of random bytes into BitTorrent flows. This raises variance in packet lengths and payload values and thus increases the difficulty of statistical fingerprinting. Third, we introduce *Random Flushes* (section III-D) to further randomize packet sizes and also packet frequency. We also discuss the applicability of *Random Packet Directions* (section III-E) and how it could be implemented in BitTorrent clients. To maintain compatibility with other BitTorrent clients and reduce pressure on the network we introduce a *Magic Peer ID* (section III-B) that can signal obfuscation support prior to the exchange of messages between peers. Finally, we address and discuss limitations of our proposal (section III-F).

Notice that the proposed techniques do only address peer-to-peer communication. The scheme does not obfuscate peer-to-tracker traffic and thus does not prevent an observer from extracting information using tracker traffic. As a result it cannot protect against the so called Sandvine attack. Tracker-to-peer traffic obfuscation that addresses the Sandvine attack problem has been proposed in [23]. We also expect that a generalization of our approach could be applied to tracker traffic.

A. Obfuscated Handshaking

The BitTorrent protocol specification [18] defines the Peer Wire Protocol, the actual protocol used to establish connections and to exchange messages. It requires the peers to do an initial BitTorrent handshake directly following the TCP handshake. This BitTorrent handshake uses a fixed string signature at a fixed position that is easily detectable using simple exact string matching. It is depicted in Figure 3. Contemporary DPI systems like openDPI [7] or l7-filter [1] simply compare the first 20 bytes to the pattern "0x13BitTorrent protocol". A match clearly indicates a BitTorrent handshake and leads to immediate service identification, which in turn allows guilt-by-association attacks.

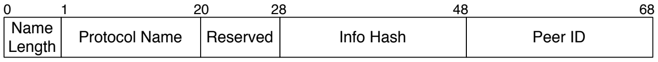

Fig. 3. BitTorrent handshake message. *Name Length* is set to "0x13", *Protocol Name* equals "BitTorrent protocol". The *Reserved* field is used to transmit extension support. *Info Hash* is the globally known hash of the torrent files info hash value. *Peer ID* is the clients random peer ID

Standard BitTorrent does not specify nor condone any obfuscation methods. After much debate *MSE* is now the defacto standard for encrypted BitTorrent connections. However, [24] 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.

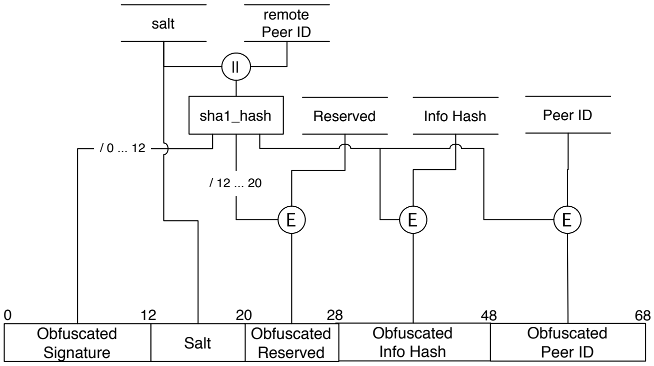

Fig. 4. Obfuscation of the handshake message. A random salt is concatenated with the remote peer ID and hashed. The hash is used to obfuscate the handshake by applying simple bit extraction and XOR operations. Concatenation is denoted by (||), (E) is a symmetric cryptographic function, in our case XOR

Since the BitTorrent handshake is the most discriminating feature of the BitTorrent protocol, concealing the handshake message itself will nullify the effect of most DPI systems. In the obfuscated handshake message all fields are obfuscated by using the sha1 hash value of the target peer's "peer ID" concatenated with a randomly generated "salt". Figure 4 and Listing 1 show the procedure and the resulting handshake message.

The 8 byte salt is generated randomly for each distinct handshake and written to the message. It is only used once together with the target peer ID as key for the cryptographic function. It can thus be also thought of as a publicly shared nonce.The target peer ID is used to prevent an observer from extracting all decoding information from the actual message.

Listing 1. Obfuscating the handshake

```
obfuscate (handshake hs, peer_id remote)
    {
        byte salt[8] = randombytes(8);
        byte key[28] = concat(remote,
            salt);
        sha1_hash digest = hash(key);
        byte rsvd[8] = E(hs.rsvd, digest
            (12,20));
        byte ih[20] = E(hs.ih, digest);
        byte pid[20] = E(hs.pid, digest);
        hs = concat(digest,salt,rsvd,ih,
            pid);
}
```

An observer would need to extract IPs, ports, and associated peer IDs from tracker responses to lookup the input for the hash function. The `sha1` hash value of {remote peer ID, salt} is then used as key to disguise all message fields. We use `sha1` since it is already a standard hash function in BitTorrent. The first 12 bytes of the hash value are written to the beginning of the handshake followed by the one-time salt itself. Next, a symmetric encryption algorithm is used to obfuscate the fields `Reserved`, `Info Hash`, and `Peer ID`. In all cases `hash(remote peer ID, salt)` serves as the input key to encrypt the fields' contents. In our scheme we use `XOR` for encryption. It is cheap, fast, and provides enough scrambling to effectively hide the information from an observer. However, `XOR` does not exhibit strong encryption and an attacker could use sequences of handshake messages to extract information through correlation. If a stronger encryption is required `XOR` can be replaced by any symmetric encryption algorithm, e.g. `AES`.

Though the handshake itself appears to be random, it still shows unique flow features that can be exploited for identification. Handshake messages are always 68 bytes long and are the first packets that are sent in alternating directions. If an outgoing packet with 68 bytes and random content is followed by an incoming packet with equal features one can assume to observe an obfuscated BitTorrent handshake.

To maintain compatibility with other clients, peers supporting the obfuscated handshake extension first try to connect using an obfuscated handshake and fall back to standard handshake if it fails. The proposed order of peer wire connection is thus *encrypt* followed by *obfuscate* followed by *plaintext*.

B. Magic Peer ID

One downside of the obfuscated handshake extension is, that it will fail if one side does not support obfuscation. In that case the target peer will sever the connection and reestablishment is required. The probability of that 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 peers support obfuscation. The peer ID is chosen such that its *sha1 hash value* shows a specific 2-byte value at a predefined position. This 2-byte value indicates support for the proposed extension. Following we will refer to a peer ID with the said characteristic as "magic" and to a peer that has a magic peer ID as "magic peer". Listing 2 shows the magic peer ID generation.

Listing 2. Magic peer ID generation

```
peer_id generate_magic_peer_id() {
        peer_id pid;
        while (!is_magic(pid)) {
                random_peer_id(pid);
        }
        return pid;
}

bool is_magic(peer_id pid) {
        sha1_hash digest = hash(pid);
        return !(digest & 0xFFFF);
}
```

Thus, a peer ID is considered magic, iff the last 2 bytes of its sha1 hash value are zero. Of course, any other predefined value at predefined positions would do. However, value and position do neither influence performance nor applicability, so we use the last two bytes for the sake of simplicity.

Finding a suitable magic peer ID requires continuous random generating and testing during startup and reduces the possible ID address space. However, computation is cheap and should on average not take more than 2^{16} tries. Reduction in address space is also negligible and will not effectively limit the number of unique peers in a swarm. In fact, it provides more randomness than the currently used peer ID conventions which use much higher numbers of fixed bytes to encode client software and version [21].

Since the peer ID is transmitted with the metainfo file by the tracker, each peer can easily determine which one supports handshake obfuscation w.h.p.. A peer receiving the peer list from the tracker will then test each peer ID of target peers for being magic. If the peer ID is magic, the according peer supports the extension w.h.p.. For any peer that does not support the extension the ordinary BitTorrent handshake is used to establish a connection. Peers that do support the extension will be connected using an obfuscated handshake. If this fails, the client falls back to ordinary handshake. Naturally there is a margin of error if a peer chooses a magic peer ID by accident without supporting the obfuscation extension. However, in that case obfuscation will fail and the connecting peer will reconnect using the ordinary handshake.

Using the peer ID to transport information allows to reduce connection establishment overhead significantly. In case of MSE/PE a peer always tries encryption first and falls back to ordinary handshaking. Thus, if the target peer does not support encryption the connection will be ceased and a new connection is necessary. When propagating support of our obfuscation extension with the magic peer ID reestablishment is only required with a very low probability in case the peer ID appears magic by accident. However, usage of the magic peer ID is *not* a requirement for obfuscation to work. It is simply a means to announce obfuscation support prior to the exchange of actual peer messages and thus reducing the probability of unwanted reconnects.

The BitTorrent specification [18] suggests that peers should not make any assumption using the peer ID since it is supposed to be completely random. However, in practice, clients use the peer ID to transmit information about the client implementation and version. [21] describes a number of peer ID styles, the most prominent being “Azureus” and “Shad0w” style. These styles also use the peer ID to transmit information and reduce the randomness even more than the suggested magic peer ID scheme. It is worth mentioning that some clients – like Transmission – dedicate quite some computational effort to identify the peer’s client software. This is because some clients implement their own, incompatible extension protocols. In order to always support the correct extension protocol version, client software and version information is needed. The magic peer ID style proposed here does not transport this information. If really needed, it is possible to generate magic peer IDs based on “Azureus” and “Shad0w” styles. This, however, limits the randomness and reintroduces patterns that can be exploited by DPI systems, which is not advised.

C. Random Padding

Random padding can be used to effectively 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 [24], and as a result can be detected.

BitTorrent’s specification [18] also specifies messages according to the type-length-value (TLV) standard. More specifically, BitTorrent messages are encoded as `<length><type><payload>`. The first integer denotes the length of the message (counting the `type` field) followed by one byte which denotes the type of the message followed by the message payload itself. Nine messages are specified, some clients also implement a tenth message. Keep-alive messages are the sole exception to this encoding standard, since they can be empty.

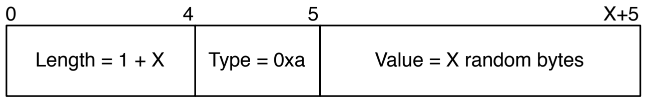

Fig. 5. Padding message. A random number of random bytes. Introduces randomness in both packet length as well as content value which makes statistical fingerprinting difficult

We suggest a new message type, called “padding” with a randomly chosen length and random payload, which can be appended to conversations at all time. However, since data transfer usually appears already pretty random and consumes much bandwidth, it is better suited for small messages. Its main purpose is to add variance and randomness to handshake and signaling messages, which makes an early identification using statistical fingerprinting extremely difficult. The message is depicted in Figure 5. Listing 3 shows an example implementation.

Listing 3. Padding message

```
byte[] padding () {
        int len = rand(1,1400);
        byte type = 0xa;
        byte value[len] = randombytes(len
            );
        byte msg[len+4] = concat(len,type
            ,value)
        return msg;
}
```

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 1,400 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 type will always equal $0xa$. This will lead to recognizable patterns in handshake messages whenever padding is appended. To counter this effect we suggest scrambling the padding message header fields during the handshake. For this we use a similar approach as with the `Reserved` field. We extract 5 bytes from the random hash and `XOR` them with the padding message header. Again, any other symmetric cryptographic function can be used instead.

Naturally, the cost of adding random garbage is bandwidth. So it is best suited for small-sized messages including the handshakes and any signal traffic.

D. Random Flush

While random flushing of TCP streams is generally possible, it requires disabling Nagle’s algorithm [26] and probably changing socket buffer sizes, which impacts overall TCP performance and usually is not advised. Since the operating system is in control of transport layer services, random flushes might not be an option on all systems, although, all major operating systems provide functions to change TCP related parameters.

In case of P2P networks, the majority of packets are used for file exchange. A strong flow feature of P2P flows is therefore *high rates of large packets in short intervals over a long period of time*. This is, however, true for any file exchange protocol, including FTP, HTTP or even SSH in case of file transfers. Nevertheless, packet sizes and frequency can be a strong indicator for file sharing. Random flushes provide more variance in packet sizes, but they neglect transmission frequency. Since packets are flushed prematurely, the overall number of packets, number of bytes – due to header overhead

– and conversation time will increase. The overall cost in terms of bandwidth over time will rise significantly.

This additional cost is neither in the interest of the consumer nor the provider. All parties have to pay the price for ignoring net neutrality. It would be best to refrain from using packet sizes and rates for identification and abstain from random flushes whenever possible. However, if needed, we suggest to also apply randomized transmission frequency to further add variance to packet rates.

E. Random Packet Directions

The BitTorrent specification states that "peer connections are symmetrical [...] and data can flow in either direction." [18]. Both the seeders and leechers can and do initiate communication. In practice, however, peers that open a connection always initiate the key exchange and handshake. This observation can be exploited to infer BitTorrent key exchanges and handshakes by just looking at the directions of the first few packets.

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 and be part in many swarms, the target peer has no possibility to know for sure, for which info hash the connection has been established. It has to wait for the info hash, provided by the handshake to identify the shared file. The only exception is, if the listening as well as connecting peer only share one common file with the same info hash, in which case the listening host could look up the needed information in the tracker response dictionary and craft the handshake himself.

Theoretically, either peer could initiate the handshake and key exchange. This would, however, require a change in BitTorrent's file sharing incentive. Usually, connecting peers announce the info hash of the shared file during handshaking and than actively request file chunks using `interested` messages. When the direction of handshakes randomly changes, the peer that establishes the connection would wait a random amount of time and listen for incoming handshakes. In this case the interested peer might initiate the handshake after a timeout or receive a handshake request before the timeout. Since the shared file is announced in the handshake the interested peer looses control about which files are downloaded over the connection.

Although random changes of packet directions is possible it requires significant changes in sharing behavior and a means of globally guaranteeing that all files are equally well shared, which is out of scope of this paper.

F. Limitations

The proposed payload obfuscation scheme requires the knowledge of the target peer's peer ID prior to initiate the handshake. This prevents a client from supporting the compact peer list extension [22], which allows trackers to return a more compact peer list that excludes the peer ID and only sends binary encoded {IP, Port} pairs. As a result pressure on the trackers rises. Although most trackers do support the compact peer list extension it is generally not a requirement and trackers do return the full peer list if requested. Still, it poses additional costs for obfuscation.

Since peer-to-tracker traffic is usually not obfuscated, an observer could capture this traffic and use the information to decode and identify obfuscated handshakes. However, this would require the observer to keep vast amounts of state including IPs, Ports and associated peer IDs. In addition he needs to observe all connections between collected hosts and actively try to decode the whole messages. While this is theoretically possible it is practically not applicable. The amount of space and time needed to perform this kind of attack is not available for many years to come on high-speed network infrastructure. Even if the exchange channel is insecure the obfuscation is practically still robust against exploitation. Furthermore, if an attacker can intercept tracker traffic he could simply disrupt all BitTorrent traffic using the *Sandvine attack*. There would actually be no point in trying to break obfuscation. As stated in section I the goal of our obfuscation extension is not to provide cryptographically secure privacy, but to prevent early identification as simple as possible.

Another limitation of using the target peer ID for obfuscation is the usage of PEX and DHT to exchange peers without a tracker. Both extensions allow peers to directly exchange compact peer lists. Since these lists do not include the peer IDs communication with exchanged peers cannot be directly obfuscated. Peers still announce to the trackers so it would be possible to learn the peer IDs over the tracker channel. Another possible solution is to take a globally known value as substitute in cases where the target peer ID is unknown. We are also evaluating ways to use other sources like torrent file content and magnet links for sources of a globally known shared value.

A potential risk of being exposed as a BitTorrent peer is the fallback to plaintext handshaking. This is done to maintain compatibility with older clients and standard even with MSE/PE. However, if really required the client could be configured to only allow connections using MSE/PE and obfuscation as fallback. However, this does not prevent other peers to try to connect using plaintext handshakes which can then be easily detected.

IV. Evaluation

We implemented the proposed method in *ttorrent 1.0.4* [27], a java BitTorrent library which is very lightweight and easily extensible. *Ttorrent* is an all-in-one solution, providing an API for clients, trackers and torrent files alike. This allowed us to use one library to rebuild the whole BitTorrent infrastructure in a controllable and, more importantly, easily evaluable fashion. 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 [9] as a testbed and deployed our modified *ttorrent* client on 70 nodes which all initially acted as leechers. The tracker and initial seeder ran on dedicated virtualized servers and tracked/seeded slitaz [10], an open source Linux distribution. We captured all traffic on the initial seeder's interface using *tcpdump* [12], which mimics the capability and view of the network access provider. In addition, we also tested interaction with a variety of popular BitTorrent client software and opentracker [8], which is the mostly used tracker software world wide.

We then analyzed the captured traffic using OpenDPI [7], [25], SPID [11], [24], and picDFI [14]. 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 that uses trainable protocol models which can dynamically be applied to identify l7 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's, which outputs per flow results by default. PicDFI is an experimental identification algorithm targeted at resource restricted environments that uses very basic behavior analysis heuristics.

A. Client Compatibility

To test communication of magic peers with ordinary peers, we introduced popular clients to the swarm. We tested the following client software.

- Vuze 4.7.0.0
- uTorrent 1.0.3 and 1.5.11
- Transmission 2.41 and 2.42
- rtorrent 0.7.9 and 0.8.9 (libtorrent-rakshasa)
- libtorrent-rasterbar 0.15.7

This client 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 some 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. We assume, that Transmission's peer ID generation algorithm exhibits a higher probability of producing peer IDs that appear *magic*, we did not verify that assumption, though.

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.

B. Magic Peer ID generation

We generated 100k magic peer IDs to analyze the number of random tries and time needed to find a magic peer ID. The generated peer IDs are also analyzed regarding their byte value distribution.

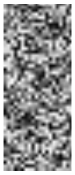

Fig. 6. Byte values of 50 random magic peer IDs. Each row represents a peer ID, the columns encode the byte value, darker means lower

Figure 6 shows the byte values of a selection of 50 of the 100k randomly generated magic peer IDs. As can be seen they appear completely random and look like noise. This is true for the whole set of magic peer IDs. Figure 7 shows the standard deviation and mean of the ids' byte values. Both are pretty linear with only small fluctuation in value. In addition, the standard deviation is also pretty high and the measured means are close to the theoretical means.

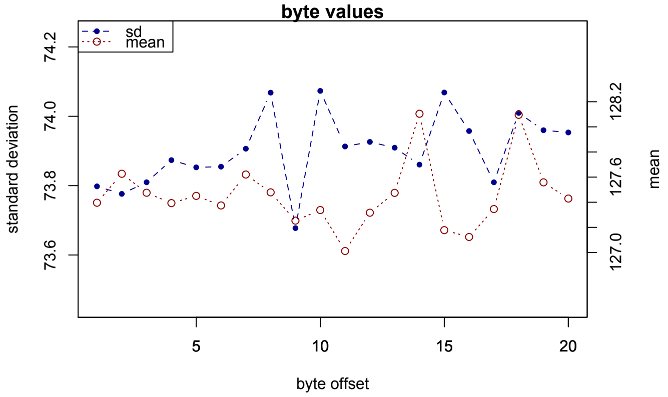

Fig. 7. Standard deviation and mean of magic Peer ID bytes

Figure 8 shows the number of tries needed to generate the magic Peer IDs. Although, the number of tries increased notably with the maximum value encountered being $766,152$, it still took only roughly one second to find. In the worst case, the startup delay introduced by searching for a magic peer ID might be enough to be recognizable by and annoying to the user. However, the Peer ID is generated only during startup and then remains constant for the remainder of the session. In extreme cases, searching a magic peer ID could be canceled in favor of decreasing startup time, sacrificing the ability for obfuscation. In addition, the client could cache previously found magic peer IDs, or search for them during idle times and save /reuse them for subsequent sessions.

As expected, on average the client needed roughly 2^{16} tries and about 100 ms to find a suitable magic peer ID.

C. Handshake message

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

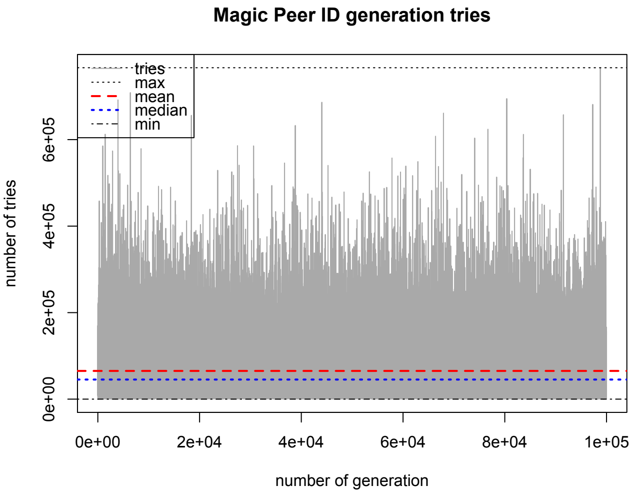

Fig. 8. Byte values of 100k magic peer IDs

to be sent immediately after the TCP handshake it can easily be observed by looking for packets with a relative sequence number of 1 in any tcp stream (assuming all packets have been captured).

Figure 9 shows the first 300 bytes of 300 of the recorded handshake messages. These packets have been recorded early at the start of the simulation. Each row denotes a handshake message with earlier messages appearing higher. The columns represent the byte offset. Byte values are encoded in color, black represents 0 while white equals 255. Since nearly all messages have individual lengths, missing byte values have been replaced with zero. Note, that handshake requests and responses are not required to appear adjacent to each other. In fact, most of the request/response pairs are delayed.

The image appears mostly random. The only recognizable pattern is the random length of the message, which is a result of adding random padding. Handshake messages are thus always between 68 and 1500 bytes long. Although, not a strong feature in itself, it might still be exploitable when combined with other flow features.

Figure 10 shows the standard deviation and mean of the handshake messages' byte values up to offset 100. It can clearly be seen that the standard deviation is high and stable throughout the whole message.

D. Random Flush

To evaluate the effect of random flushes, we deployed one peer seeding an image of Ubuntu 11.10 and one leeching peer. For comparison, the file is exchanged twice. First, the seeder does not use random flushing. For the second exchange, the seeder sets the *TcpNoDelay* property and continually retrieves x bytes from the message queue, where x is random and $1 \leq x \leq 1401$, and writes them to the socket's output stream. The traffic is captured on the leeching peer's interface. We then analyze the traffic regarding number of packets and bytes sent during the conversations. The results are shown in table I.

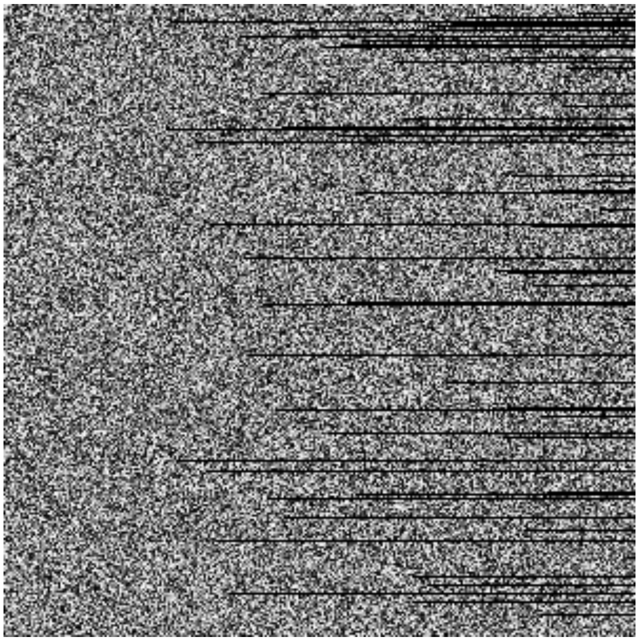

Fig. 9. 300 recorded Handshake messages. Each row corresponds to a message, the columns represent byte values. For each message 300 bytes are shown. A recognizable pattern is the variable length of the padding messages (horizontal black bars)

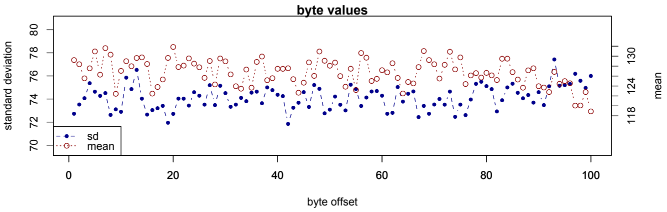

Fig. 10. Standard deviation and mean of handshake message bytes

					bytes		
Flush	**#pkts**	**#secs**	**sum**	**min**	**max**	**avg**	**stdev**
no	821k	139	785.3m	66	1,434	956.35	650.14
yes	2,8m	893.6	917.5m	66	1,434	326.76	411.16

TABLE I
PACKET SIZE COMPARISON IN THE PRESENCE AND ABSENCE OF RANDOM FLUSHING

When using flushing the number of packets sent is 3.41 times higher than using standard transmission policy. The overhead in terms of total number of bytes is about 16%. Regarding conversation time, random flushing needed nearly 6.5 times longer to exchange the file. Interestingly, although the average packet size is much smaller in presence of flushing, variance and standard deviation do not increase but instead actually decrease compared to no flushing. The reason can be seen by examining the actual frequencies shown in fig. 11.

In case of no flushing packet sizes show frequency peaks at both the small and big extremes. Most packets are bigger

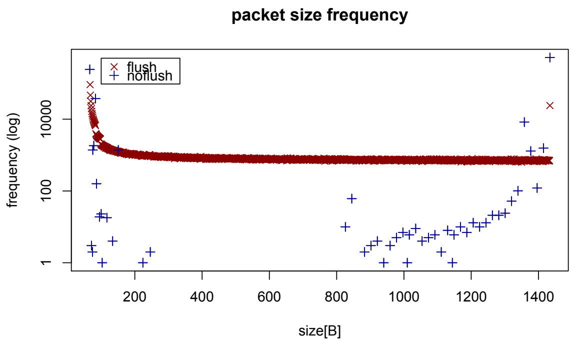

Fig. 11. Frequency of packet sizes in presence and absence of random flushing. Note that the figure only shows flushing, padding of small packets is not applied and discussed independently in section IV-C. That explains why the curve is not flat

Algorithm	total flows	Identified	
		protocol	number flows
OpenDPI	403	unknown BitTorrent	374 29
SPID	389	unknown HTTP	360 29
picDFI	793	unknown P2P	431 361

TABLE II
IDENTIFICATION RESULTS

than 1,200 bytes. There are no packets with sizes between 300 and 800 bytes. This is because the messages sent are either file transfers, resulting in big packets, or signaling, like keep-alives, and file chunk requests. Signaling packets are small and they appear quite often. For the same reason, frequency also peaks for small packets in case of flushing. However, instead of clusters of big packets, the sizes are equally distributed between 200 and 1,400. Naturally, this leads to lower variance. To also eliminate the peak of small packets, random padding can be applied to signaling messages, increasing their size, and, thus, flattening the frequency distribution.

As shown, random flushing proves to be quite expensive in consumed bandwidth over time. This adds significant pressure on the network, which should be avoided whenever possible. When random flushing is applied, smaller packets should be expanded with random padding to produce a more equal distribution.

However, it is questionable if the additional costs are justified. Short term, statistical identification tools that use packet sizes and rates can be fooled. But on the long term, high frequencies of random sized packets is a strong flow feature in itself. Therefore, when disguising flow features through random padding and flushing, these techniques should be rarely applied and probably limited to handshake and signal messages, and not actual file transfer.

E. Identification

To test the effect of handshake obfuscation and random padding on contemporary identification mechanisms, we analyzed the recorded traffic using state-of-the-art DPI and statistical traffic identification systems. Note, that for identification analysis we only applied payload obfuscation and random padding since we concentrate on handshake obfuscation. Flushing is only required and useful for large packets that are exchanged later in the flow.

As DPI system we used Ipoque's OpenDPI [25]. For statistical analysis we used SPID [24] a system that computes fingerprints and compares these to pre-learned protocol models to perform identification. A multitude of different meters can be used to describe a known protocol which allows different levels of detail but also introduces a high complexity. In addition, we used picDFI [33], an efficient flow inspection and classification algorithm, which specifically targets P2P applications is based on the assumption, that P2P uses both TCP and UDP on the same port within a short period of time. It is targeted for resource restrictive environments with the intent not to give highly accurate results but to provide preliminary analysis and hints about interesting flows.

The results are shown in table II. As can be seen, both, DPI as well as SPID, were not able to detect and identify the obfuscated BitTorrent traffic. OpenDPI only identifies 29 flows as BitTorrent. These are all seeder to tracker announces that OpenDPI recognizes by searching for the string "GET " and then parsing the whole tracker get request looking for the client software and info hash values. These flows are equivalent to the 29 HTTP flows detected by SPID. Tracker requests look pretty much like HTTP requests and SPID was not able to distinguish between those two. Since OpenDPI aims at early detection by looking for the string "0x13BitTorrent protocol" it was unable to recognize the obfuscated handshakes and thus failed to identify the BitTorrent flows. One reason why SPID fails to detect BitTorrent correctly, is probably the use of payload based fingerprints to describe flow features. Since our obfuscation scheme scrambles the bytes and the length of the handshake, variance in byte values and packet length is high and will match to protocol models with similar behavior. In fact, in most cases SPID reported higher similarities of the flows to eDonkey and ISAKMP than to BitTorrent. Interestingly, picDFI manages to classify about 45% of the flows correctly as P2P. However, picDFI is only able to distinguish between P2P and non-P2P traffic and is not a substitute for full-fledged classification systems. The reason why picDFI could identify those flows is because it was able to correlate UDP signal traffic and TCP file exchange traffic on some time frames. The difference in number of total flows is a result of different flow descriptors and flow timeouts.

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. It is worth mentioning that in our simulations a very simple behavior based algorithm performs much better than sophisticated DPI and SPI systems.

V. Conclusion and future work

The intention of this paper is not to present an optimal privacy solution for BitTorrent networks, rather to show that contemporary traffic identification systems can easily be fooled.

We presented a novel BitTorrent obfuscation scheme that is easy to implement, backwards compatible and fairly efficient. It circumvents all 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, including keeping large amounts of state data. Even state-of-the-art statistical identification tools can easily be fooled and circumvented by applying basic payload and flow feature obfuscation.

Established identification paradigms seem to come to the end of applicability. With an increasing effort of programmers, users, and even commercial service providers, packet and traffic features become more and more disguised.

We argue that future traffic identification must not rely on payload and flow based data but instead concentrate on global application domain specific features which are unlikely to change. The old meme of the need to identify the exact application layer protocol is outdated and violates network neutrality and the end-to-end principle. Knowledge about the usage of layer 7 protocols does not generally allow statements about the actual application domain. In the end, the most important information for the network provider should be how his resources are used and how usage can be optimized. Not what his customers are talking about. The ongoing ISP-vs-Customer war achieves very little at the expense of much resources.

Future work will explore possibilities to introduce more randomness into the obfuscation process and apply more sophisticated flow feature obfuscation. A promising approach seems to disguise traffic to appear to originate from other applications and protocols. This “Any Signature Protocol” should be able to transport information using the syntax of other well-known protocols, like HTTP, SMTP, or DNS without exposing application specific signatures. Regarding identification and classification, a thorough examination of internet application domains is due. The result would be a *taxonomy of internet traffic* that would allow classification of any traffic into application domains based on the actual semantics and not syntax of conversations.

References

[1] “Application layer packet classifier for linux,” online, retrieved 2012.01.12. [Online]. Available: http://l7-filter.sourceforge.net/

[2] “Btguard anonymous bittorrent services,” retrieved 2012.01.12. [Online]. Available: http://btguard.com/

[3] “Edkobfuscation,” online, retrieved 2012.01.12. [Online]. Available: http://mldonkey.sourceforge.net/EDKObfuscation

[4] “emule protocol obfuscation,” online, retrieved 2012.01.12. [Online]. Available: http://wiki.emule-web.de/index.php/Protocol_obfuscation

[5] “I2p anonymous network,” online, retrieved 2012.01.12. [Online]. Available: http://www.i2p2.de/

[6] “It’s hidden,” retrieved 2012.01.12. [Online]. Available: http://itshidden.com/

[7] “Opendpi,” online, retrieved 2012.01.12. [Online]. Available: http://www.opendpi.org/

[8] “opentracker,” online, retrieved 2012.01.12. [Online]. Available: http://erdgeist.org/arts/software/opentracker/

[9] “Planet-lab,” online, retrieved 2011.11.11. [Online]. Available: http://www.planet-lab.org

[10] “Slitaz,” online, retrieved 2012.01.12. [Online]. Available: http://www.slitaz.org/

[11] “SPID Statistical Protocol IDentification,” online, retrieved 2012.01.12. [Online]. Available: http://sourceforge.net/projects/spid/

[12] “tcpdump,” online, retrieved 2012.01.12. [Online]. Available: http://www.tcpdump.org/

[13] “The Onion Router,” online, retrieved 2012.01.12. [Online]. Available: https://www.torproject.org/

[14] authors removed, “title removed,” review anonymization.

[15] T. Berson. (2005) Skype Security Evaluation. online. Anagram Laboratories. Retrieved 2012.01.12. [Online]. Available: http://www.anagram.com/berson/abskyeval.html

[16] 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.

[17] 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*, vol. 11, no. 3, pp. 37–52, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1109/SURV.2009.090304

[18] B. Cohen, “The BitTorrent Protocol Specification,” online, 01 2008, retrieved 2012.01.12. [Online]. Available: http://bittorrent.org/beps/bep_0003.html

[19] F. Desclaux, “Skype uncovered,” online, 2005, retrieved 2012.01.12. [Online]. Available: http://www.ossir.org/windows/supports/2005/2005-11-07/EADS-CCR_Fabrice_Skype.pdf

[20] P. Gupta and N. McKeown, “Algorithms for packet classification,” *IEEE Network*, vol. 15, pp. 24–32, 2001.

[21] D. Harrison, “Peer ID Conventions,” online, 2008, retrieved 2012.01.12. [Online]. Available: http://www.bittorrent.org/beps/bep_0020.html

[22] ——, “Tracker Returns Compact Peer Lists,” online, 2008, retrieved 2012.01.12. [Online]. Available: http://www.bittorrent.org/beps/bep_0023.html

[23] D. Harrison, A. Ciani, A. Norberg, and G. Hazel, “Tracker Peer Obfuscation,” online, 01 2008, retrieved 2012.01.12. [Online]. Available: http://bittorrent.org/beps/bep_0008.html

[24] E. Hjelmvik and W. John, “Breaking and Improving Protocol Obfuscation,” Chalmers University of Technology, Tech. Rep. 123751, 2010. [Online]. Available: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=123751

[25] Ipoque, *OpenDPI Integration Manual*, 1st ed., Ipoque, September 2009. [Online]. Available: http://opendpi.org/

[26] J. Nagle, “RFC 896: Congestion control in IP/TCP internetworks,” Jan. 1984, status: UNKNOWN. [Online]. Available: ftp://ftp.internic.net/rfc/rfc896.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc896.txt

[27] M. Petazzoni, “A bittorrent library; in java, for java,” retrieved 2012.01.12. [Online]. Available: http://turn.github.com/ttorrent/

[28] D. E. Taylor, “Survey and taxonomy of packet classification techniques,” *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[29] various, “Debate over protocol encryption,” online, 2007, retrieved 2012.01.12. [Online]. Available: http://forum.utorrent.com/viewtopic.php?id=5742

[30] Vuze, “Message stream encryption,” online, retrieved 2012.01.12. [Online]. Available: http://wiki.vuze.com/w/Message_Stream_Encryption

[31] M. Waldvogel, M. Muncan, and M. Patidar, “Stealth dos,” 2006.

[32] 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. [Online]. Available: http://www.caida.org/research/traffic-analysis/classification-overview/

[33] T. Zink and M. Waldvogel, “Analysis and efficient classification of P2P file sharing traffic,” University of Konstanz, Tech. Rep., 2010. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:bsz:352-188702

Summary Review Documentation for

"BitTorrent Traffic Obfuscation: A Chase towards Semantic Traffic Identification"

Authors: Thomas Zink, Marcel Waldvogel

SUMMARY REVIEW

The paper is very well written, and on a very interesting topic. The problem of how to fool current traffic identification systems is interesting and relevant. The paper describes the design, implementation, and evaluation of a number of mechanisms to obfuscate BitTorrent traffic. Three obfuscation mechanisms are treated with more or less depth: the obfuscation of BitTorrent handshake messages, the injection of padding messages, and random flushes to randomize packet sizes and frequency. (Obfuscating tracker communications is considered orthogonal and is not discussed in the paper.) One problem tackled here is the one of having peers determine whether they support the obfuscation protocol without revealing this information to the censor (e.g., ISP). To do so, the authors propose to encode obfuscation support in peer IDs that are then published and distributed by trackers. Such a "magic" peer is a random number whose last two bytes are equal to zero. Handshakes are then obfuscated so magic peer IDs are not easily readable by the censor. A fair portion of the paper deals with efficiently generating such IDs. The authors also discuss how concealing BitTorrent flow features can be done with random padding, random flush, and changing packet direction. he system description was clear and crisp, as was the message that simple obfuscation can defeat sophisticated classification techniques. The challenges, the logic, and the techniques applied are explained sufficiently. The paper presents a proper evaluation of the techniques. The evaluation section of the paper shows that two DPI and statistical traffic identification systems out of three fail to recognize obfuscated BitTorrent flows.

Strengths: Some of the strengths pointed out by the reviewers include: the paper is well written, it presents a real implementation, and it clearly demonstrates the ease in which simple obfuscation techniques can defeat existing systems.

Weaknesses: The main issues raised by the reviewers were:

1) It seems that the obfuscation is highly dependent on the communicating parties being able to discover each other's peer-id through a secondary channel, namely, the tracker. How hard would it be similarly obfuscate tracker traffic? For modern BitTorrent systems that are trackerless and/or use PEX to discover new peers, how easy would it be to implement these obfuscation techniques? It seems that these techniques require some peer specific information to orchestrate the obfuscation, and the communication required to exchange that information cannot be obfuscated. Is it sufficient for the classification systems to just focus on identifying the non-obfuscated communication channels?
2) The paper states that it is easier for the client to introduce new obfuscation techniques than it is for the traffic classification systems to apply new identification rules. However, although it takes time to create a new classification rule to detect the signatures of a new obfuscation technique, once the rule has been created, it requires very little effort to disseminate it to all of the professionally managed traffic classification systems. It is indeed a cat-and-mouse game between the BitTorrent users and the traffic classification systems, but it seems that the game is currently not in the BitTorrent users favor and it is not convincing that there is any immediate need for classification systems to move towards exploiting global communication patterns, as suggested in the abstract and conclusion of this paper.
3) Exploring how current traffic identification techniques can be fooled is interesting. However, showing that a signature identification technique can be fooled by changing the signature to be identified is absolutely trivial and has no scientific merit.
4) The authors put much effort into scrambling the BitTorrent handshake message for a very poor result. Indeed, first it is not shown that simply changing the "BitTorrent protocol" string within the BitTorrent handshake is not enough to fool current techniques. Second, as the proposed system falls back to the regular handshake in case a peer does not understand the obfuscation, and as a peer has in the order of 50 neighbors, the probability to have at least one neighbor that uses the regular BitTorrent protocol is high. Because BitTorrent multiplexes all traffic on a single port, it will be easy to any traffic identification system to block all traffic for this IP address and port, and therefore, it is very easy to defeat the proposed obfuscation.
5) The authors spend quite a lot of time on the notion of magic ID ending with 00, but it is completely unclear what is the point of this magic ID. In BitTorrent, the reserved bytes in the handshake can be used for a specific flag for any purpose. Assuming that the authors want to use a specific convention in the peer ID, why don't they just truncate the random generated string by two bytes, and replace the last two bytes by zeros?
6) The evaluation section on identification lacks several important details. For example, why is detection by OpenDPI and SPID avoided but not by piCDFI? Furthermore, it would be better to provide information regarding each mechanism instead of aggregate identification results. An obvious question is whether obfuscating

the handshake is sufficient or whether the additional proposed mechanisms are also required.

Response from the Authors

First we would like to thank the reviewers for their comments and insights on the paper. We tried to address all of the comments for the camera ready version.

1) We added a new Subsection III.F to discuss technical limitations of our approach. Obfuscation of the tracker traffic is actually discussed in related work [23]. It is true that identification of tracker traffic allows identification of BitTorrent hosts. However, to decode the obfuscated traffic requires the observer to keep large amounts of state and perform decoding at line speed, which is impractical. PEX and DHT only support compact peer lists, which exclude the peer ID. If the peer ID is unknown another globally known value could be used as substitute. This could come from torrent files or magnet links.
2) We added more information to the Introduction. The assumption might be true for software-based low-speed classification systems. But we target high-speed network provider equipment. The classification algorithms and rules are usually implemented directly in hardware (FPGAs) since classification must be performed on wire-speed. Therefore, changes are not easy and also quite expensive.
3) This is discussed more deeply in the Introduction and Section III. Our approach not only changes the handshake signature but also eliminates statistical features to appear random on the wire. This not only fools signature-based but also statistical identification systems. And is difficult to adapt to.
4) This is discussed in the new Section III.F. Fallback to plaintext indeed poses a risk of exposure. If one must truly be obfuscated it is possible to only allow encrypted and obfuscated traffic. Fallback allows to be backwards compatible in case exposure can be risked.
5) There was quite some confusion regarding the magic peer ID. It is now discussed more deeply in section III.B. Not the magic peer ID ends with 00, but it's SHA1 hash value. Also, the magic peer ID is not a requirement for obfuscation, but a simple means to signal obfuscation support prior to actually exchanging messages. This significantly reduces network overhead.
6) The Evaluation has been expanded to answer these questions. Generally, each subsection in Evaluation deals with each mechanism independently. The identification results only aggregate payload obfuscation and random padding. OpenDPI is signature-based and had to fail. SPID also uses payload value ranges (among others) for identification. PicDFI is specifically designed for P2P and simply looks for TCP and UDP on the same port within a short time frame.