# Stealing Bandwidth from BitTorrent Seeders

Florian Adamsky[a], Syed Ali Khayam[b], Rudolf Jäger[c], Muttukrishnan Rajarajan[a]

[a] *City University London, London, England*
[b] *PLUMgrid, Inc., Sunnyvale, CA, USA*
[c] *THM University of Applied Sciences, Campus Friedberg, Germany*

**Abstract**

BitTorrent continues to comprise the largest fraction of Internet traffic. While significant progress has been made in understanding the BitTorrent choking mechanism, its security vulnerabilities have not been investigated thoroughly. This paper presents an experimental analysis of bandwidth attacks against different choking algorithms in the BitTorrent seed state. We reveal a simple exploit that allows malicious peers to receive a considerably higher download rate than contributing leechers, therefore introducing significant efficiency degradations for benign peers. We show the damage caused by the proposed attack in two different environments: a lab testbed comprising 32 peers and a PlanetLab testbed with 300 peers. Our results show that 3 malicious peers can degrade the download rate up to 414.99 % for all peers. Combined with a Sybil attack that consists of as many attackers as leechers, it is possible to degrade the download rate by more than 1000 %. We propose a novel choking algorithm which is immune against bandwidth attacks and a countermeasure against the revealed attack.

*Keywords:* Peer-to-peer, BitTorrent, Attacks, Countermeasures

## 1. Introduction

High market penetration of broadband connectivity in the past decade has catalyzed a fundamental change in user's traffic characteristics with Peer-to-

*Email addresses:* Florian.Adamsky.1@city.ac.uk (Florian Adamsky), akhayam@plumgrid.com (Syed Ali Khayam), Rudolf.Jaeger@iem.thm.de (Rudolf Jäger), R.Muttukrishnan@city.ac.uk (Muttukrishnan Rajarajan)

Peer (P2P) file sharing content comprising a considerable fraction of today's Internet traffic [1, 2]. Simultaneous to widespread usage of P2P software, a global debate continues to take place on copyright violations perpetuated through P2P software. In addition to public litigation, active measurement studies [3, 4] have revealed many attacks on P2P systems, allegedly launched by companies hired by the music and film industries. Hence, at this time it is important to investigate the threat landscape for P2P systems.

Considering the BitTorrent ecosystem, an attacker has four major components to attack: leechers, seeders, peer and torrent discovery. Peer discovery techniques have evolved with the introduction of Distributed Hash Table (DHT) [5], Peer Exchange (PEX) [6] and Local Peer Discovery (LPD) [7]. Also, the change from major torrent discovery websites (e.g. PirateBay) to magnet links [8] makes the torrent discovery process more robust against attacks. Consequently, leechers and seeders represent the most vulnerable parts in this ecosystem. One attack that is directed against leechers and seeders is the bandwidth attack.

Dhungel et al [9] defined a bandwidth attack as a malicious peer who manages to download from the seeder with the highest speed. As a result, the malicious peer allocates a slot from the active peer set. We add a new attack dimension to this vector, where an attacker gets more bandwidth by one of the following scenarios:

- incorrect protocol implementation;
- incorrect protocol specification;
- programming errors in the BitTorrent client; and/or
- implementation errors in the transport protocol (e.g. TCP [10], Micro Transport Protocol (uTP) [11]).

In this paper, we investigate the vulnerability of different chocking algorithms in seed state against bandwidth attacks and reveal a vulnerability caused by incorrect specification of the BitTorrent fast extension. This extension was introduced to ramp up the bootstrapping time for new peers. However, a malicious peer can exploit this extension to steal bandwidth even when a peer is choked. We show that this attack can create significant reductions in download rates for all participating peers.

The key contributions of this paper are as follows:

- We evaluate the effectiveness of bandwidths attacks on different chocking algorithms in seed state on a lab testbed system with 32 peers running commonly-used BitTorrent client software.
- We repeat the experiment in a large-scale scenario on PlanetLab with 300 peers to validate the attacks' utility and effectiveness.
- We show how an attacker can exploit a programming error in the choking algorithm to steal large amounts of bandwidth from a seeder.
- We show a vulnerability caused by an incorrect protocol specification, analyze its impact empirically, and propose, implement and evaluate a countermeasure to patch the vulnerability.
- We propose a countermeasure against the allowed fast attack and propose a novel seeding algorithm which is resilient against bandwidth attacks. We evaluate the proposed algorithm for performance, stability and security.

## 2. Related Work

In this section, we discuss the related work that has influenced and inspired this paper.

Adar and Huberman [12] analyzed the user traffic on Gnutella and found that 70 % of all Gnutella users share no files. They argue that free riding is a major threat to P2P networks which leads to degradation of the system performance and introduces vulnerabilities in the system.

Dhungel et al. [13] provided the first investigation of bandwidth and connection attacks on BitTorrent. They defined bandwidth attacks as peers who try to allocate an upload slot from the seeder as soon as possible to nip the seeder in the bud. Their measurements showed that bandwidth attacks are rather ineffective, and it is only possible to increase the download time up to 10 %. In another work, Dhungel et al. [4] came to the conclusion that it is not possible to nip the seeder in the bud. The present work, on the contrary, shows that bandwidth attacks can be launched effectively against seeders. The same authors also studied connection and piece attacks against leechers in detail [14].

Liogkas et al. [15] designed and implemented three selfish-peer exploits to obtain bandwidth without sharing pieces with other peers. In the first

exploit, their client only downloads pieces from the seeder. Seeders can be easily identified as they advertise themselves by sending a `HAVE_ALL` message or a complete bitfield. The second exploit attempts to download only from the fastest peers. This exploit observes the frequency of the `HAVE` messages from the victim. This information is exploited to rougly calculate the download rate of the peer. The last exploit introduces false but rare pieces to attract high bandwidth leechers. This attack exploits the vulnerability that a peer can announce pieces which it does not own. They concluded that their exploits delivered significant benefits, but also that BitTorrent proved to be quite robust against them. Extending this work, Locher et al. [16] developed a selfish BitTorrent client called *BitThief*, which never serves any content to other peers. This client exploits optimistic unchoking and does not perform any chokes or unchokes, and never announces any pieces. The results of this study showed that BitThief succeeded in downloading the complete file in any case. In rare cases, their client even outperformed the mainline client. In both of these works, the focus of the attack was to download the complete file without sharing upload bandwidth. While prior work in this domain focusses on downloading of a complete file, we instead investigate the effectiveness of attackers which are only interested in degrading system efficiency but are not not interested in data integrity.

Defrawy et al. [17] showed that it is possible to launch a distributed denial-of-service (DDoS) attack on BitTorrent. DDoS belongs to the category of bandwidth attacks. An attacker can set a victim as one of the trackers. All future peers attempt to contact the victim, consequently flooding the victim with BitTorrent packets.

Piatek et al. [18] performed a measurement of millions of BitTorrent users and showed that the performance and availability of BitTorrent is quite poor. These measurements motivated the authors to design and implement a new one-hop reputation protocol for P2P networks. The idea of this protocol is to encourage persistent contribution incentives and rewarding contributions. Every client maintains a history of interactions, which serve as intermediaries attesting of the behavior of others. While this protocol limits free-riding, it is hard to compare their protocol with the seeding algorithm proposed in this paper (Section 6.2) as one-hop reciprocation changes the standard BitTorrent protocol behaviors.

We have shown in our previous work [19] that it is possible to exploit the choking mechanism in leech state. This can destabilize BitTorrent's clustering to attack high bandwidth leechers. One disadvantage of this attack is

that an attacker has to wait until the victim optimistically unchokes him. Since this aspect makes the attack somewhat ineffective, in this work we investigate attacks from a malicious peer towards specific seeders, without the need for the malicious peer to be an active swarm member.

## 3. Background

This section gives a brief overview of the BitTorrent protocol and its fast extension, with special emphasis on those parts of the protocol that can be exploited.

### *3.1. Terminology*

In view of a lack of uniform terminology in BitTorrent and P2P communities, we first introduce the terminology that will henceforth be used in the paper.

**Peer** A node that runs a BitTorrent client.

**Swarm** All the peers sharing a torrent are called a swarm.

**Torrent** A file that contains metadata about the swarm and the distributed files.

**Leecher and Seeder** A peer is a leecher when it is downloading content of a torrent. A peer is a seeder when it has downloaded all the content and is sharing it with other leechers.

**Free Rider** A peer that only downloads data and denies uploading to other peers.

**Choked** Peer $P$ is choked by peer $Q$ when $Q$ does not send data to $P$.

**Interested** Peer $P$ has data that peer $Q$ wishes to acquire.

**Active Peer Set** Active peer set for a peer $P$ is a subset of peers that are interested and $P$ has unchoked them.

**Piece** The download is divided into equally sized parts called *Pieces*.

**Block** Pieces are again divided into equally sized *Subpieces* or *Blocks*.

We use the terms *choking algorithm in seed state* and *seeding algorithm* interchangeably.

*3.2. BitTorrent protocol overview*

Two unique distinguishers of BitTorrent are its bandwidth choking and rarest-piece-first mechanisms. BitTorrent uses these two complementary techniques to avoid free-riders and to ensure a tit-for-tat-ish way of resource sharing [20, 21]. The choking algorithm decides which peers may download and which may not. Choking is done for the following reasons:

- it prevents free-riders;
- it ensures a consistent download rate;
- TCP's congestion control behaves poorly when sending over many connections at once.

The algorithm works in a tit-for-tat-ish way and favors the peers who are uploading more actively. Each connection has two states on either end: uninterested/interested and unchoked/choked. A peer is interested when the opposite side has data which the peer wants to acquire. Choking means that a peer does not send data until unchoking happens. Data will only be sent when one side is interested and the other side is unchoked. By default every peer has 4 unchoke slots and decides which peers get unchoked according to the following policy:

- Every 10 seconds all peers are ordered by their download rates. The top 3 peers in this ordered list are unchoked.
- Every 30 seconds one peer is chosen randomly to get unchoked. This is called *optimistic unchoking*.

Optimistic unchoking allows new peers, which have nothing to share, to get pieces more quickly to engage in the tit-for-tat. This bootstrapping process takes several minutes to take action. Data between peers is only be sent when one peer is interested and the other one is unchoked. The decision of which peer will be choked is based on the peers' upload rates, and the top 4 peers (i.e. peers with the highest upload rates) will be unchoked. The choking algorithm is different when a peer is in seeder state. We will introduce four different seeding algorithms in Section 4.1. In the next section, we describe an extension to the BitTorrent protocol to speed up the bootstrapping process.

*3.3. Allowed Fast Extension*

In 2008, Harrision and Cohen released the BitTorrent Enhancement Proposal (BEP) 6, which describes a new extension called *Fast Extension* [22]. It contained four extensions which are: state machine reworking, have-all, suggest-piece, and allowed-fast [23]. We focus on the allowed fast extension. The BitTorrent choking mechanism has one disadvantage: If a peer has no data to share, it needs several minutes to ramp up before it can be a full member of a swarm. The allowed fast extension tries to speed this process up. Without this extension a peer without data needs to wait until another peer optimistically unchokes it. The allowed fast extension enables downloading even if a peer is choked. This allows a peer to get pieces quickly and to engage in BitTorrent's tit-for-tat.

If a peer asks for a piece but is choked, it gets an allowed fast message that contains a list of pieces that can be downloaded. This list is called *allowed fast set* and is generated according to the pseudocode given in Algorithm 1.

```
1  x ⟵ 0xFFFFFF00 & ip;
2  x.append(infohash);
3  while |x| < k do
4  |   x ⟵ SHA1(x);
5  |   for i ← 0 to 4 do
6  |   |   break if(|a| == k);
7  |   |   piece ⟵ partition first 4 Bytes from x;
8  |   |   add piece to a if it's not already there;
9  |   end
10 end
```

**Algorithm 1:** Algorithm that generates the allowed fast set.

Lines 1 and 2 in Algorithm 1 remove the last octet from the IP address of the peer, concatenate it with the infohash of the torrent, and save it to a variable $x$. Then the algorithm (line 4) calculates the SHA1 [24] hash of the variable $x$. Lines 5–9 slice four bytes from the beginning of the SHA1 hash and interpret that as an integer value. This integer is the piece number. If this piece is not yet in the allowed fast set, it is added to the set. This gets repeated until the allowed fast set contains $k$ pieces. BEP 6 sets $k = 10$, however peers are free to adjust $k$ to suit their load. The next section describes a vulnerability of this extension and shows a way to exploit it.

## 4. Bandwidth Attacks

In a bandwidth attack, a malicious leecher steal bandwidths by occupying an unfair number of a seeder's unchoke slots. As a result, the download speed from benign leechers gets reduced. Additionally, if the seeder has set a seeding ratio, the seeder will stop seeding earlier. These attacks are hard to detect as an attacker running a bandwidth attack only needs a misbehaving BitTorrent client.

A peer maintains $n$ connections to other peers. From these $n$ peers, a seeder gives an unchoke slot to $u$ peers and an optimistic unchoke slot to $o$ peers. The BitTorrent specification sets $u = 3$ and $o = 1$. If an attacker can exploit the choking algorithm in seed state and occupy those $u$ slots, then other peers may starve. This is because the other peers can only get pieces through the optimistic unchoke slot, since this slot is assigned randomly and therefore hard to attack. We denote the upload capacity of the seeder $b$. TCP distributes the available bandwidth evenly across all active connections. Consequently the bandwidth consumption from a malicious peer is:

$$b_A = u \times \frac{b}{u+o}. \tag{1}$$

The bandwidth consumption of the other peers, which get pieces through the optimistic unchoking, then is :

$$b_L = o \times \frac{b}{u+o}. \tag{2}$$

Since $u > o$, a bandwidth attack can generate significant damage to the overall performance of a swarm. Combined with a Sybil attack, an attacker can attack all seeders in a swarm, as a single attacker can be easily detected. Since there are different peer selection algorithms, the attacker needs different exploits for those algorithms. We examined the default seeding algorithm of prominent BitTorrent clients and list those in Table 1—entries are ordered by the latest market share statistics from [25].

In the next sections, we will describe and show how an attacker can exploit different choking algorithms in seed state to occupy an unfair number of the unchoke slots.

### *4.1. Seeding Algorithms*

While significant progress has been made in understanding the BitTorrent choking mechanism, its security vulnerabilities have not been investigated

Table 1: BitTorrent clients in combination with the used Seeding Algorithm order by market share according to [25].

| # | Client | Version | Market Share | Seeding Algorithm |
|---|---|---|---|---|
| 1 | uTorrent | 3.2.2 | 47.97 % | Longest Wait |
| 2 | Vuze | 4.8.1.2 | 22.49 % | Fastest Upload |
| 3 | Mainline | 7.7.2 | 13.01 % | Longest Wait |
| 4 | Transmission | 2.61 | 7.00 % | Fastest Upload |
| 5 | Unknown | | 5.22 % | n/a |
| 6 | Libtorrent | 0.16.10 | 1.02 % | Round Robin |

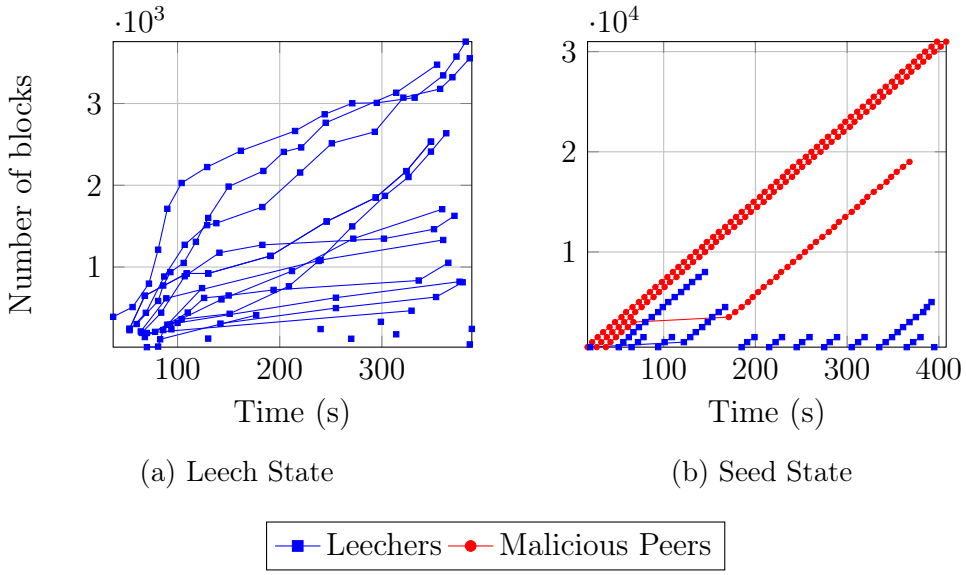
thoroughly. Neither is there an incentive in BitTorrent to stay as a seeder (making seeders rare), nor is it possible for a seeder to verify if an unchoked peer is behaving properly. This property can be exploited by a malicious peer. In Figure 1a one can see the upload piece distribution of the choking algorithm in leech state and Figure 1b in seed state. Every line represents a different peer; the length represents the upload duration.

Figure 1a depicts the result of the tit-for-tat incentive mechanism for a leecher that uploads pieces to multiple peers. Only peers who uploaded to that leecher are in turn allowed to download. However, the seeder in Figure 1b only uploads to a couple of peers, because the malicious peers have the fastest download speeds. Other peers can only download pieces via the optimistic unchoke slot, where one peer is chosen randomly to download. This example shows that the seeding algorithm specified in BEP 03 is unfair and can be exploited by malicious peers [21] [18]. In the following subsections we show different seeding strategies that are used by popular BitTorrent clients and different ways to exploit them.

#### *4.1.1. Fastest Upload*

The Fastest Upload (FU) algorithm is similar to the choking mechanism in leecher state, except that a seeder uses the download rate, instead of the upload rate, to select candidates to unchoke. It follows that this algorithm favors the fastest downloaders. The assumption behind this algorithm was that a fast downloader is also a fast uploader and therefore the distribution is faster when favoring the fastest downloader. This assumption was based on a false premise as asynchronous Internet connections (e.g. Digitial Subscriber Line (DSL)) are still used widely in home and small office networks.

This algorithm was used widely in early versions of BitTorrent clients,




(a) Leech State (b) Seed State

Figure 1: Comparison between the upload piece distribution of the choking algorithms in (a) leech state and in (b) seed state. The data was produced with the cluster environment from Section 5.1 with 1 seeder that has 5 Mbps upload limit, 29 leechers with 1 Mbps and 3 fast peers with no limit (malicious peers).

and is still used by 29.49 % BitTorrent clients (e.g. Vuze, Transmission), according to Table 1. It can be easily exploited by an attacker that has a high download capacity [18]. A BitTorrent client sorts all peers according to the download rate. If an attacker downloads faster than the other peers, then it is possible to occupy the unchoke slots.

*4.1.2. Round Robin*

Round Robin (RR) is a well-known OS scheduling algorithm that gives each process equal time frames. In the context of BitTorrent, RR algorithm operates on the number of pieces, with the upload slots rotating every $n$ pieces.

An attacker has no possibility to get a permanent slot. Every peer gets the same amount of pieces. However, if there are multiple attackers connected to the seeder then the benign peers can be forced to wait wait longer to acquire their pieces.

*4.1.3. Anti Leech*

Chow et al. [26] found out that leechers' progress is slower at the beginning (when a leecher has only a few pieces) and at the end (when the leecher has difficulties to find peers with interesting pieces). To solve this problem, they introduced a new seeding algorithm that prefers peers when they only have a few pieces and when they have nearly all the pieces. We are not aware of any implementation of it apart from libtorrent (versions 0.16.0 and higher), where they called this algorithm Anti Leech (AL). From every peer $p$ in the peer list, the seeder calculates a score according to equation 3:

$$a(p) = \begin{cases} f - f(p) & \text{if } f(p) < \frac{f}{2} \\ f(p) \times \frac{1000}{f} & \text{otherwise,} \end{cases} \tag{3}$$

where the number of pieces from the complete file is denoted as $f$ and the number of downloaded pieces from a peer $p$ is denoted as $f(p)$. The seeder orders the peers in the peers list according to their score and unchokes the first three.

The authors mentioned that this algorithm can be exploited by an attacker by pretending to have none or nearly all pieces. However, they argue that this is hard, since it requires source code modification. We disagree with this argument because client-side source code changes can be trivially introduced by malicious peers. To avoid this attack, the authors propose a simple countermeasure that keeps track of the uploaded pieces to a specific peer and blocks a peer if it has reached that limit.

*4.1.4. Longest Waiter*

Starting with version 4.0.0, the mainline client introduced a new algorithm in seed state, that we call Longest Waiter (LW). It is first mentioned by Legout et al. [21]. This algorithm orders all unchoked and interested peers according to the time they were last unchoked.

LW is mostly used from clients that are programmed by BitTorrent Inc. This algorithm is comparable with RR, since an attacker has no possibilities to get any advantages over the benign peers. However, every additional attacker who is connected to the seeder increases the wait time for the benign peers.

*4.2. Programming Errors in Clients*

Programming errors like buffer or heap overflows can cause significant damage on a remote host since an attacker can exploit this error to exe-

cute malicious code. In a P2P network, an attacker can additionally exploit programming errors to cause free riding or to steal bandwidth from other peers.

During our research, we have found such a programming error in libtorrent[1]. In our experimental evaluation in Section 5, we found out that the seeding algorithm RR is the most vulnerable one. Since RR cannot be influenced by an attacker, it was clear that this artifact was caused by a programming error. Together with the author of libtorrent, we exposed a faulty counter variable which counted the number of bytes during the last unchoke round. This counter was being reset for every unchoke round (typically 15 seconds) and not every unchoke. This led to a situation where an attacker does not lose his permanent slot. This can be seen in the experimental evaluation in Section 5, as we informed the author who quickly wrote a patch for this issue to solve it in future versions. In the following sections, we will refer to malicious peers exploiting this implementation vulnerability asRound Robin Fixed (RF).

### *4.3. Allowed Fast Extension Attack*

The allowed fast extension has two security vulnerabilities. The first one is integrated into Algorithm 1, which removes the last octet of the IP address of the peer. This ensures that a peer that has more IP addresses in a network cannot get more pieces. The second consideration is that an attacker might download the allowed pieces over and over again, as the BEP 6 specification states that: "*A peer MAY reject requests for already Allowed Fast pieces if the local peer lacks sufficient resources, if the requested piece has already been sent to the requesting peer, or if the requesting peer is not a starting peer*". According to RFC 2119 [27] "MAY" means optional, thus implyingthat it is possible to ask for the same pieces over and over again. To test this attack, we modified a BitTorrent client that exploits this vulnerability.

#### *4.3.1. Attack Description*

The pseudocode which attacks the allowed fast extension is described in Algorithm 2.

The attacker peer first contacts the victim and sends the victim a BitTorrent handshake. During the handshake, the victim tells the attacker if

[1] `http://www.rasterbar.com/products/libtorrent/index.html`

```
1  foreach incoming message M do
2      switch M do
3          case HAVE_ALL or HAVE
4              Reply with HAVE_NONE;
5              Send INTERESTED;
6          case ALLOWED_FAST or CHOKE
7              add piece to S_1;
8              begin thread with endless loop
9                  forall the pieces of S_1 do
10                     Send REQUEST for piece;
11                     Wait n seconds;
12                 end
13             end
14     endsw
15 end
```

**Algorithm 2:** Pseudocode of the Fast Extension Attack.

he/she supports the fast extension. After the handshake, the victim sends a message with an indication of whether it is a seeder (`HAVE_ALL` ) or a leecher (`HAVE` message). In both cases, the attacker replies with a `HAVE_NONE` message, followed by an `INTERESTED` message (see line 3–5 in Algorithm 2). If the victim's BitTorrent client fully implements the allowed fast extension, the attacker receives an `ALLOWED_FAST` packet with a piece number.

After the initial handshake, the attacker (lines 9–14) starts a new thread for every `ALLOWED_FAST` packet it received. Those threads contain an endless loop that requests all pieces from the *allowed fast set*. This ensures that the attacker gets the pieces when choked and unchoked, thereby allowing the attacker to steal bandwidth permanently.

*4.3.2. Affected Clients*

Table 2 lists prominent BitTorrent clients that we have tested for the vulnerability described in the last section. Entries in the table are ordered by the latest market share statistics from the Tribler[2] P2P research team [25]. uTorrent, the client with the largest market share, supports the fast exten-

[2] `http://www.tribler.org/`

Table 2: BitTorrent clients order by market share according to [25]. Column **Vulnerable** shows if the client supports the allowed fast semantics and if it is vulnerable to the proposed attack.

| # | Client | Version | Market Share | Vulnerable |
|---|---|---|---|---|
| 1 | uTorrent | 3.2.2 | 47.97 % | No |
| 2 | Vuze | 4.8.1.2 | 22.49 % | Yes |
| 3 | Mainline | 7.7.2 | 13.01 % | No[3] |
| 4 | Transmission | 2.61 | 7.00 % | No |
| 5 | Unknown | | 5.22 % | n/a |
| 6 | Libtorrent | 0.16.10 | 1.02 % | Yes |

sion, but only half of the semantics of the allowed fast part. This is because BitTorrent Inc. has called upon academics [28] to study the consequences of this extension, which will not be implemented until more insight experimental results are available. In this regard, to the best of our knowledge, this the first paper to investigate the security consequences of the allowed fast extension. Vuze, the second most widely-used protocol, is vulnerable to the proposed attack. However, Vuze is only partly affected as it allows pieces to be downloaded 64 times and then all further requests are rejected. Nevertheless we have listed Vuze as vulnerable since it is possible to reconnect and restart the attack. The mainline BitTorrent client is similar to uTorrent. Transmission supports the fast extension, but the code was commented at the time of writing of this paper–clearly the possibility of including the extension in future releases is being considered right now. To test libtorrent, we used Deluge that makes use of this library. This client is also vulnerable to the proposed attack. Based on these statistics, it is clear that a large fraction of BitTorrent clients in the world are vulnerable to the fast extension attack.

The next step is to test this attack in a real network to see how the BitTorrent protocol reacts to it.

## 5. Experimental Evaluation

In this section, we describe the experimental setup and show the results under the following attack scenarios: In the first experiment, we attack the initial seeder with different number of attacks and different upload limits and

[3]At the moment, but code is in mainline so will be vulnerable soon.

compared the results with an experiment without an attacker. In the second experiment, we simulate a Sybil attack where an attacker uses multiple compromised computers to launch the attack.

To judiciously evaluate the attacks' effectiveness, our experiments are repeated on two testbeds: a Cluster lab environment and PlanetLab. We now describe both these testbeds and outline our experimental results on these testbeds.

*5.1. Experimentation on the Cluster Testbed*

We perform experiments using private torrents in our testbed consisting of 32 nodes with a controller and a monitor node. These nodes are desktop machines, which are running the BitTorrent Transmission version 2.61 and Deluge version 1.3.5 (libtorrent 0.16.10) over Ubuntu GNU Linux 12.04.1 LTS. A network diagram can be seen in Figure 2. We wrote a distributed experiment library in Perl to simultaneously control all nodes. This software monitors and records the status of each peer at every second. This includes: nodes in peer set, nodes in active peer set, interested and choked states for each of the peers in the active peer set, and upload and download rate with each of the peers in the active peer set. In all experiments, the seeder distributes a file with a size of 100 MiB and a piece size of 64 KiB. The controller node executes the experiments and monitors each of these nodes.

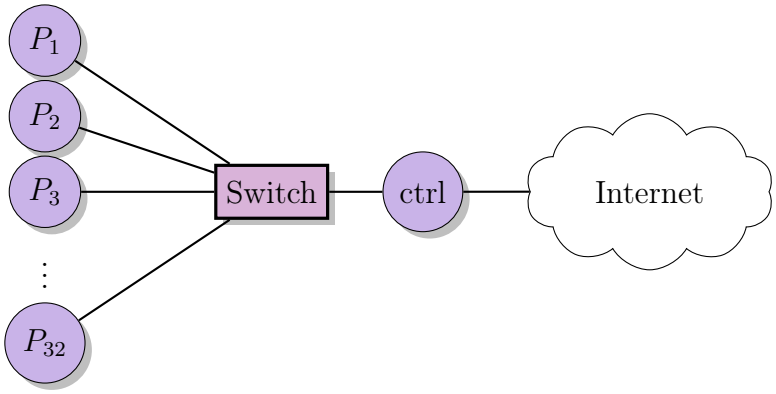



Figure 2: Network diagram of the Cluster testbed.

All experiments simulated a flash crowd scenario to get reproducible results. To create a more realistic scenario, every node generated random HTTP background traffic. Every leecher disconnected after receiving a complete copy of the file and only the initial seeder stayed connected for the complete duration of the experiment—this is because BitTorrent does not reward

a seeder to stay active. Finally, every experiment was repeated 10 times and the average values for these experiments are reported in subsequent sections. In this section we experimentally evaluate the effectiveness of the proposed attack. We use a *delay ratio* ($d$) metric to quantify the effectiveness of an attack:

$$d = \frac{t_d^{'}(x) - t_d(x)}{t_d(x)}, \tag{4}$$

where the average download time of an arbitrary piece $x$ without the attack is denoted as $t_d$, while the download time with an ongoing attack is denoted as $t_d^{'}$.

In this experiment, we attacked the initial seeder with 1, 2, 3 and 4 attackers and compared the results with an experiment without an attacker. We repeat this experiment for every seeding algorithm mentioned in Section 4.1. We set the upload limit from the seeder to 1, 5 and 10 Mbps in different experiments. Leechers do not have up or download limits.

*5.1.1. Seeder with 1 Mbps Upload Limit*

Figure 3 shows the average download time with an increasing number of attackers. Our first observation from Figure 3a is that RR is the most vulnerable algorithm—with one attacker, RR has a delay ratio up to 42.17 %, with two attackers up to 105.60 %, with three attackers a sharp jump up to 328.76 %, and with four attackers the average download time increases by 414.80 %. This is the highest increase and can be explained by the fact that the RR implementation in libtorrent was incorrect and favored attackers—as explained in Section 4.2. The allowed fast attack has not increased RR significantly.

The second most vulnerable algorithm is FU. The growth of the average download time from FU is somewhat linear. It starts with an increase of up to 29.63 % with one attacker and ends eventually with an increase of up to 189.07 % with four attackers. However, it is worth mentioning that the allowed fast attack can increase the results from the bandwidth attack against FU. Both scenarios with one and two attackers the allowed fast attack is around 10 % higher than the bandwidth attack. With three attackers, the difference is up to 70 % and with four attackers it is up to 45 %.

What is interesting to note from the results of the AL algorithm is that there is a big difference between a bandwidth attack and the allowed fast attack. With three attackers, the bandwidth attack achieves a $d$ of up to

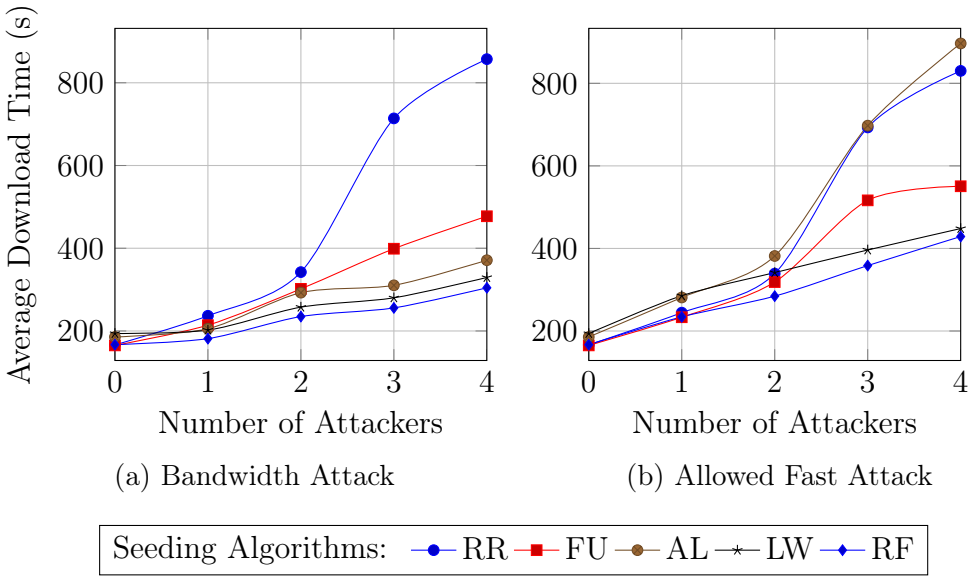



Figure 3: File transfer of a 100 MiB file with a piece size of 64 KiB via BitTorrent with a seeder that has a 1 Mbps upload limit. Subfigure (a) shows a normal bandwidth attack and subfigure (b) shows a bandwidth attack combined with the Allowed Fast Attack.

67.23 % and the allowed fast attack achieves a $d$ up to 275.80 %. This can be explained by the flash crowd situation, since nearly all peers have the same score at the beginning and at the end of this scenario. If peers have the same score, libtorrent prefers the peer which have waited the longest. This indicates that the bandwidth attack against AL only takes effect when the other peers have some pieces. However, the allowed fast attack steals bandwidth even when choked. We believe that the effect of bandwidth attack against AL is higher in real-world, because a flash crowd scenario can only appear at the beginning of a torrent. The algorithms LW and RF are relatively resilient against bandwidth attacks. LW reached with four attackers up to 69.84 % and RF up to 82.31 %.

*5.1.2. Seeder with 5 Mbps Upload Limit*

Figure 4 depicts an attack against a seeder with 5 Mbps upload limit. Contrary to the attack against the seeder with 1 Mbps upload limit, the most vulnerable algorithm is not RR, rather it is FU. This indicates that the programming error is only visible when the seeder has low bandwidth

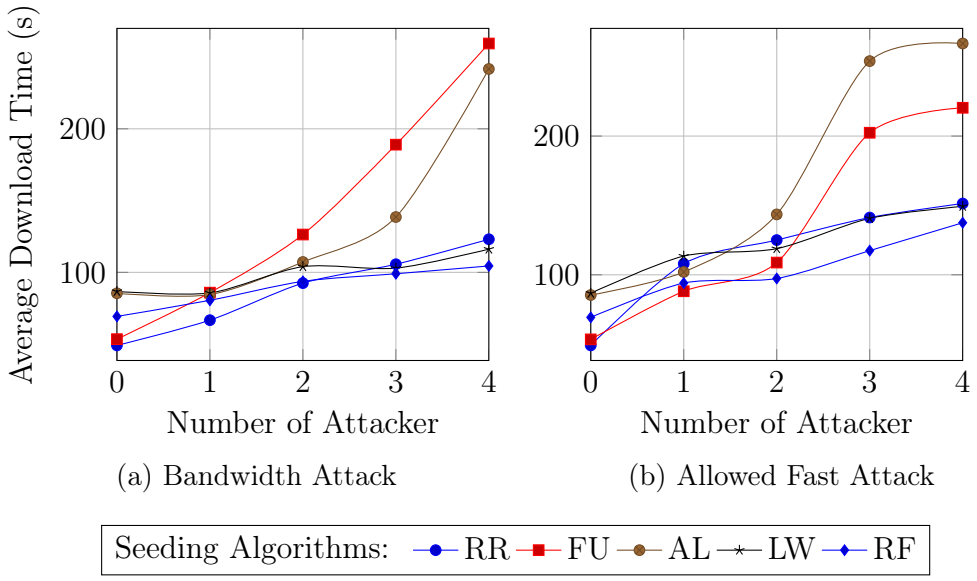



Figure 4: File transfer of a 100 MiB file with a piece size of 64 KiB via BitTorrent with a seeder that has 5 Mbps upload limit. Subfigure (a) shows a normal bandwidth attack and subfigure (b) shows a bandwidth attack combined with the allowed fast attack.

capabilities. The $d$ of FU with one bandwidth attacker is up to 60.64 % and with four up to 385.41 %. As written in Section 5.1, the leechers and the attackers have the same bandwidth capabilities. Nevertheless, the attacker is able to request more pieces than the competitive leechers using the simple attack script in Algorithm 2.

The next algorithm that is impacted adversely is AL. A bandwidth attack with one attacker is not significantly different from the experiment without attacker. However, the $d$ with two attackers reaches up to 25.52 %, with three attackers up to 62.21 %, and with four attackers up to 182.99 %. Similar to the attack experiment with 1 Mbps against AL, the allowed fast attack increases the impact. The allowed fast attack achieves against AL the following $d$ values: 19.45 %, 68.05 %, 197.44 % and 212.31 %. As in the previous experiment, the seeding algorithms RF and LW have the lowest impact. LW gets with a normal bandwidth attack and four attacker up to 33.93 % increased and with the allowed fast attack up to 72.19 %. RF is also increased up to 50.74 % and up to 98.44 % with the allowed fast attack.

### 5.1.3. Seeder with 10 MBit Upload Limit

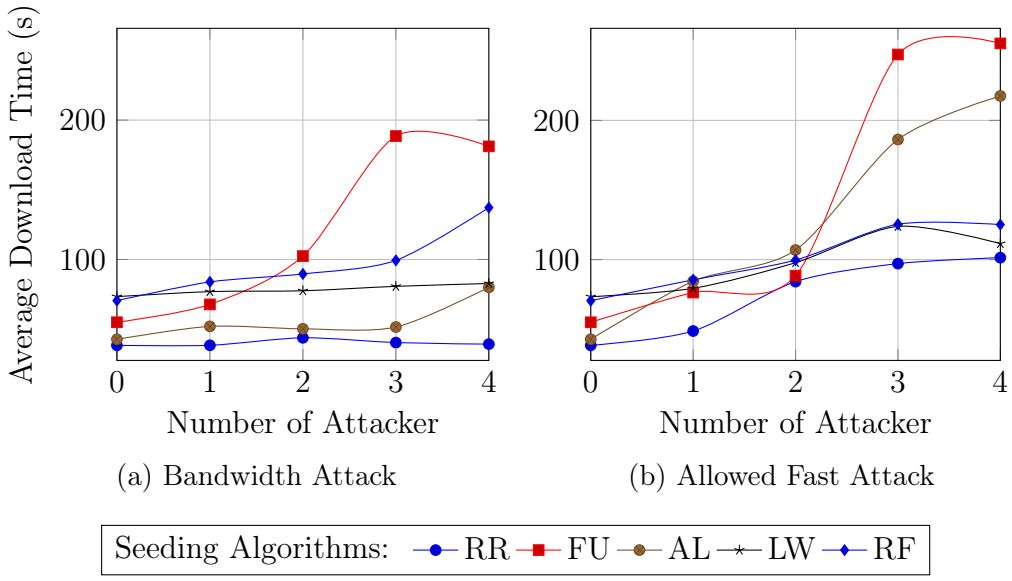



Figure 5: File transfer of a 100 MiB file with a piece size of 64 KiB via BitTorrent with a seeder that has 10 Mbit upload limit. Subfigure (a) shows a normal bandwidth attack and subfigure (b) shows a bandwidth attack combined with the allowed fast attack.

Finally, we repeat the experiment with a seeder having a 10 Mbps upload limit (Figure 5). Generally, it can be seen that the more bandwidth the seeder has, the more resilient it is against bandwidth attacks. In this experiment, the most vulnerable algorithm is again FU. The $d$ of FU with one, two, three and four bandwidth attackers is up to 23.29 %, 86.36 %, 242.97 % and 229.54 %, respectively. With the help of the allowed fast extension attack using three attackers, the $d$ can be degraded up to 349.94 %. For all other algorithms' degradation of $d$ is also significant but less than 100 %. The second most vulnerable algorithm was the AL seeding algorithm, similar to the experiment with 5 Mbps. The attack achieves with one attacker a $d$ of up to 97.66 % and with four attackers 409.15 %. For the other seeding algorithm, we only mention the results with four attackers. The broken round-robin implementation achieves a $d$ of up to 163.94 %, LW up to 52.00 %, RF up to 77.51 %.

*5.1.4. Launching Bandwidth Attacks in Sybil Mode*

In a Sybil Attack [29], an attacker injects multiple fake peers, which are all under the control of the attacker, into the network. We now evaluate the efficacy of the proposed allowed fast attack in Sybil mode.[4] In this experiment, with every iteration we increased the number of attackers and reduced the number of leechers until we have the same number of attackers and leechers. Figure 6 depicts the results with a seeder that has a 5 Mbps upload capacity.

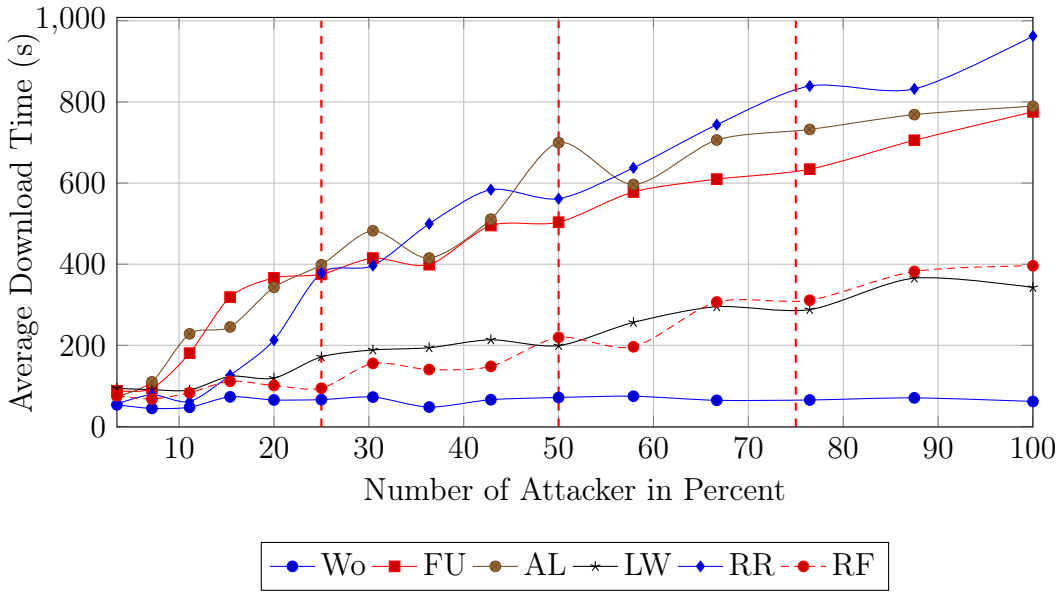



Figure 6: File transfer of a 100 MiB file with a piece size of 64 KiB via BitTorrent with a seeder that has 1 Mbps upload limit. Subfigure (a) shows a normal bandwidth attack and subfigure (b) shows a bandwidth attack combined with the Allowed Fast Attack.

The Sybil attack shows that as more attackers are injected into the swarm, the impact of the attack becomes progressively more severe. When 25 % from the swarm are attackers, then it is possible to increase the average download time for all peers up to more than 500 % if the seeder uses FU, AL or RR. The seeding algorithms LW or RF also concede a $d$ of more than 250 %. If there are half as many attackers as leechers, then the $d$ of the leechers increases up

[4]This is a realistic scenario as botnets are available for hire for as little as $0.50 per bot [30].

to 700 % if a seeder makes use of FU, AL and RR. However, if a seeder uses LW or RF then this value is more than 300 %. If an attacker introduces the same amount of attackers as leechers, then the most vulnerable algorithms FU, AL and RR increase the average download time up to 1000 %. RR is the worst hit with the highest $d$ of 1561.18 %. LW and RF concede greater than 500 % increase.

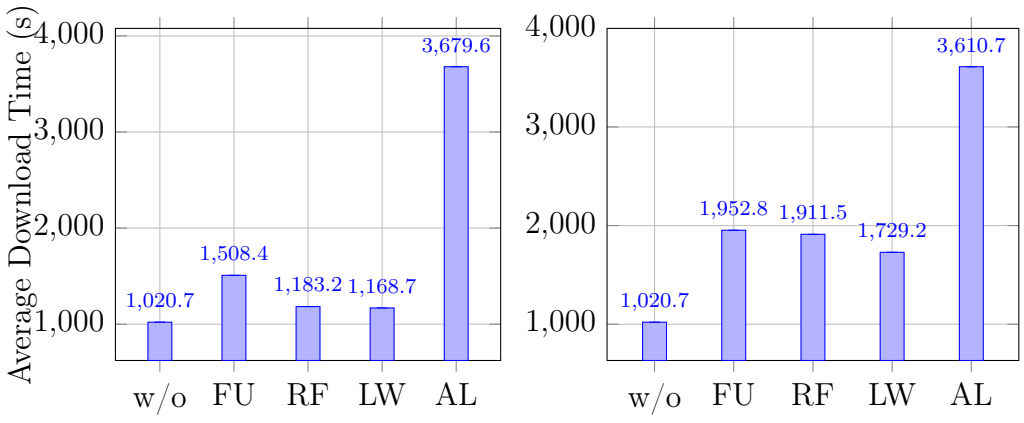
*5.2. Experimentation on the PlanetLab Testbed*

The next set of experiments were performed on the PlanetLab platform [5] to test the large-scale effects of bandwidth attacks. PlanetLab is a collection of machines distributed over the globe. All nodes are connected to the Internet and therefore experience all the facets of an unreliable network, like latency, packet loss, packet corruption, etc.

Our setup of the large-scale experiment consists of one seeder, one tracker and 300 leechers. This scale of this setup is inspired by similar experiments by prior studies; e.g. [31], [18] and [32]. All experiments simulated a flash crowd scenario and all peers have upload and download rates of 1 Mbps. This is a judicious bandwidth number, since much of the developing world is still operating in the 1 Mbps range [33]. We introduced 4 attackers (1.33 % of the leechers) that had no bandwidth limits. Figure 7a shows the results of the bandwidth attack and Figure 7b outlines the results of the allowed fast attack. All experimental results are averaged over ten runs.

The results show that AL is the most vulnerable seeding algorithm with a $d$ of 247.68 %, because of its missing security feature. Note that this value is quite close to the cluster result presented earlier which showed a delay ratio of 275.80 %. Similar to our previous experiments, the second most vulnerable algorithm is FU where a bandwidth attacker can increase the average download time of 300 leechers up to 47.78 % and with the allowed fast attack up to 92.32 %. Both RF and LW are relatively robust against bandwidth attacks with an increase of 15.92 % and 14.50 %. However, the allowed fast attack is able to increase the average download times of RR and LW up to 87.23 % and 69.41 %, respectively. We also repeated the experiment with less then 300 leechers and obtained similar results.

[5] `http://www.planet-lab.org/`




(a) Bandwidth Attack (b) Allowed Fast Attack

Figure 7: File transfer on PlanetLab of a 100 MiB file with a piece size of 64 KiB via BitTorrent. The data was produced with 1 seeder that has a 1 Mbps upload limit, 300 leechers with 1 Mbps download limits and 4 malicious peers. All values are averaged values of ten iterations.

### *5.3. Discussion*

We showed an in-depth security analysis of the different seeding algorithm of the BitTorrent protocol. The results show that the seeding algorithms RR, FU and AL are quite vulnerable, while RF and LW are not as vulnerable. A malicious peer that exploits these algorithms increase its seeding score and therefore gets more download time. It was not possible to cripple the network completely, because of the optimistic unchoke slot. However, with unlimited resources, an attacker can slow down the BitTorrent download for all peers to an unusable level.

We also provided large-scale experiments on PlanetLab to show the real-world effects of bandwidth attacks. While the attacks' impact on PlanetLab was a bit more contained than the cluster experiment, attack trends observed in the cluster testbed were validated by the PlanetLab experiments. We only had one seeder and in a real-world swarm there are usually more seeders. However, a malicious peer can attack multiple seeders at the same time—it is easy to find all the seeders in a swarm, as one just has to connect to the tracker frequently and wait for a peer that sends a complete bitfield or a `HAVE_ALL` message. The only exception here is a seeder that uses the super-seed [34] feature.

## 6. Countermeasures

In this section, we propose a countermeasure against the allowed fast attack and a novel seeding algorithm which is highly resilient against bandwidth attacks.

### *6.1. Allowed Fast Attack*

There are two possible countermeasures against the allowed fast attack. The first is to change the word *"MAY"* to *"MUST"* in the following sentence from BEP 6: "*A peer MAY reject requests for already allowed fast pieces (. . . )*". However, this countermeasure prevents retransmission of a damaged piece. Another countermeasure is to upper bound the number of pieces that can be downloaded; for example, the client Vuze has a limit of 64 times. While this strategy reduces the effectiveness of this attack, it is still possible to restart the attack after the limit is reached.

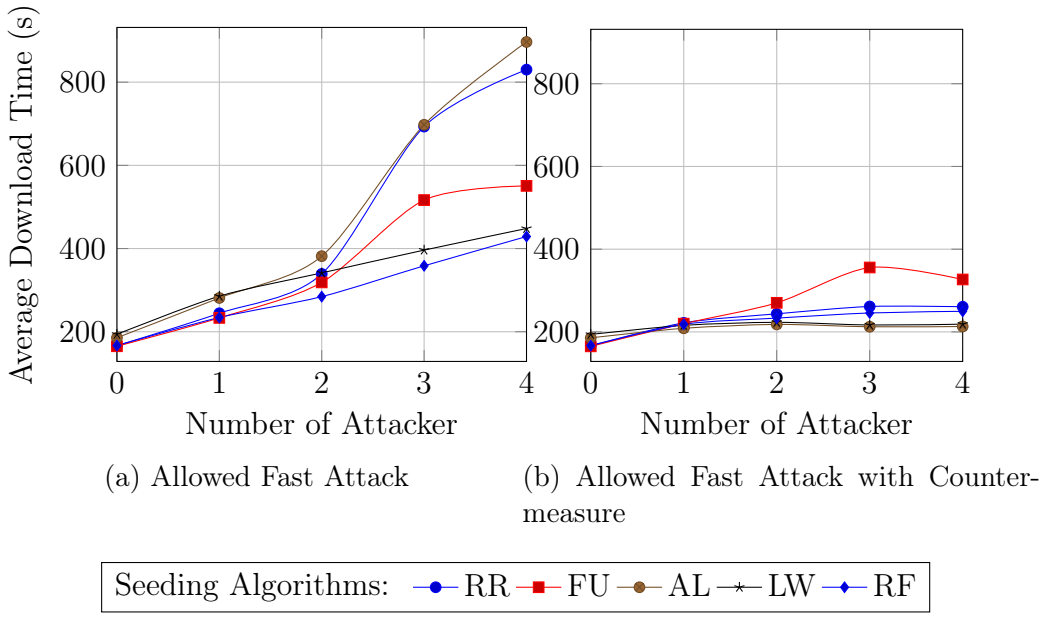
In light of the above discussion, an effective countermeasure must restrict the IP address of the peer which has reached the limit to avoid restarting of the attack. We implemented this countermeasure in libtorrent and repeated the experiment with a seeder having a 1 Mbps upload limit.

Figure 8 shows experimental results of implementing the countermeasure that limits the allowed fast pieces. Figure 8a shows the results from the allowed fast attack and Figure 8b shows the results with a patched libtorrent version. It can be seen that with the countermeasure in place the attack has nearly no effect on the seeder. However, the seeding algorithm FU is an exception where the attacker is not choked from the seeder because it is the fastest downloader. Our countermeasure strategy against the allowed fast attack has been contributed to the community in libtorrent 0.16.11 [6].

### *6.2. Bandwidth Attack*

As seen in our experimental evaluation, the seeding algorithms LW and RF are the ones which are most resilient against bandwidth attacks. However, an attacker just has to wait long enough to get a slot. For a working countermeasure against bandwidth attacks, the seeder needs to verify which peers have shared pieces with other peers and which ones have not. The seeder would then unchoke $u$ peers, which have shared the most number of

[6] `http://sourceforge.net/p/libtorrent/mailman/message/31298868/`




(a) Allowed Fast Attack

(b) Allowed Fast Attack with Countermeasure

Figure 8: File transfer of a 100 MiB file with a piece size of 64 KiB via BitTorrent with a seeder that has 1 Mbps upload limit. (a) Allowed fast attack; (b) Allowed fast attack with a patched seeder.

pieces. This requires a secure proof that a peer has shared pieces with others, that is hard to fake by an attacker.

We propose a new seeding algorithm that is fast, hard to exploit, and ensures that only peers that have shared are getting an unchoke slot. We call this seeding algorithm Peer Idol (PI) [35]. If a peer wants to download pieces from a seeder then it has to send a new BitTorrent message to the seeder, that we call *vote*. This message contains a list of $n$ peers which the requesting peer has downloaded the most pieces from. Thus a peer has to vote for other peers. The notation $A \succ B$ indicates that the requesting peer has downloaded more pieces from peer $A$ than from peer $B$. Figure 9 shows an example with 3 peers and one seeder which has two unchoke slots.

All leechers from the example in Figure 9 send their votes to the seeder $S_{PI}$. Peer $A$ favors the peers $C$, $B$ and $D$, as $A$ has downloaded the most from these peers. The seeder $S_{PI}$ awards each peer that is in the vote with points: Peer $C$ gets 3 points, peer $B$ gets 2 points and peer $D$ gets 1 point. We calculate the PI score as follows:

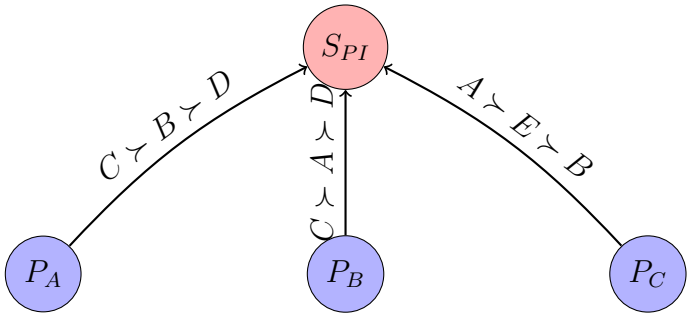


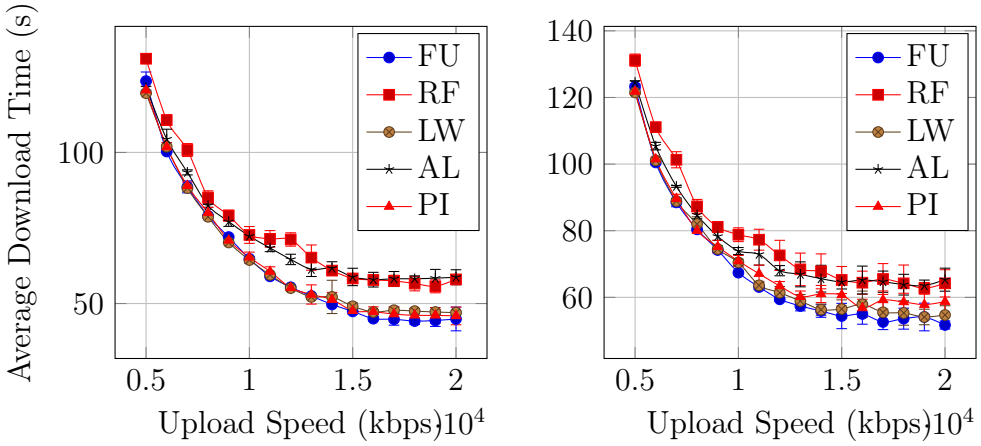

Figure 9: Example of how leechers send their votes to the seeder.

$$PI(p) = \sum_{i=1}^{|I|} V_i(p). \tag{5}$$

The function $V_i(p)$ returns 1 or 0, depending on weather peer $i$ voted for peer $p$ or not. If $PI(p)$ of two peers is the same, then PI prefers the peer which has waited the longest. Every unchoked peer downloads for at least two unchoke rounds to avoid quick choking and unchoking, known as *fibrillation* [36]. According to Equation (5), Peer $C$ has the highest score with $PI(C) = 6$, followed by Peer $A$ with $PI(A) = 5$ and Peer $B$ with $PI(B) = 3$. Thus, $S_{PI}$ unchokes $C$ and $A$ and chokes $B$. If a vote contains peers unknown peer, the seeder will add these peer in a list of potential peer candidates. This list contains peers from LPD, PEX and DHT. If the seeder has to little peers, it randomly chooses peers from this list and tries to connect to it. In the example in Figure 9, $S_{PI}$ saves the two unknown peers $D$ and $E$ in its candidate list.

This score calculation is a well-known method in political voting and is called Borda Count (BC), named after Jean-Charles de Borda [37]. Instead of BC, we have also tested the Condorcet Method (CM) algorithm, but noticed two major problems: First, the computation complexity of CM is $\mathcal{O}(N^2)$ as all peers have to compete with each other. In comparison, the complexity of BC is $\mathcal{O}(1)$, because the seeder increases the vote counter of that specific peer. Secondly, since the last peer would loose all pairwise comparisons, the score of the last peer would always be $CM(p) = 0$.

We defined additional security rules to protect against attacks. A peer must send a vote in order to get voted. This ensures, that peers send votes to the seeder. But an exception can be made, if the seeder does not have enough




(a) Leechers have no upload or download limit

(b) Leechers have different download limits and no upload limit

Figure 10: Effects of the different seeding algorithms on the average download speed in different environments. The upload speed of the seeder was gradually increased. Source: [35].

peers. A peer gets blacklisted and disconnected if the vote contains more than $n$ peers, the IP address of the requesting peer or repeated peers. In the next subsections we evaluate the PI scoring method in terms of performance, stability and security.

*6.2.1. Performance*

One of BitTorrent's most important properties is performance, so a new seeding algorithm should not make BitTorrent slower. Our hypothesis is that PI will not degrade BitTorrent's performance, even with the message overhead of the vote message. This is based on the fact that PI reward sharing peers. In the first experiment, we compared the performance of the seeding algorithms in an optimal environment (described in Section 5.1), where the leechers had no download or upload limits. The seeder, however, had an upload speed limit that was gradually increased. Figure 10a shows the average download speed of all peers depending on the upload speed from the seeder.

The results in Figure 10a were limited to 5–20 Mbps, as we did not observe a significant difference between the algorithms in the 1–4 Mbps range. Beginning from 5 Mbps RF and AL move apart from FU, LW and PI. For simplicity, RF represents the slower group and PI the faster group. The

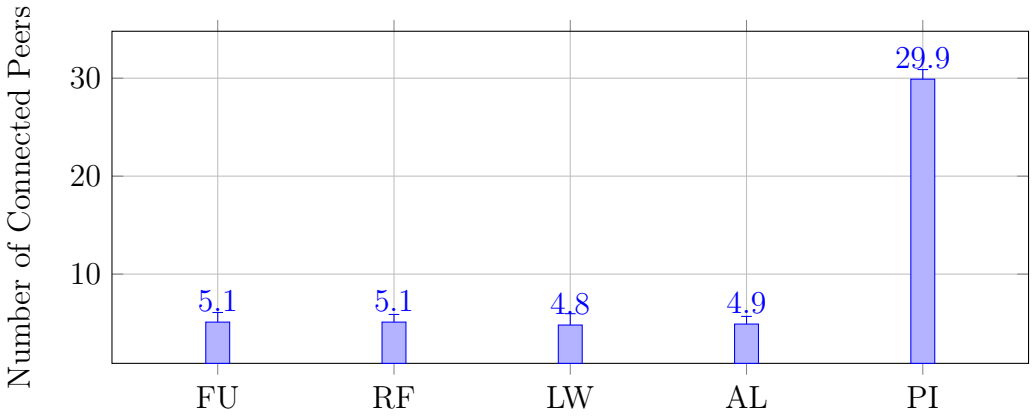



Figure 11: The number of connected peers of the seeder in a swarm of 32 peers. All values are average values of ten iterations.

seeding algorithm RF ranges from 55.7–567 s and has a mean value of 120.5 s. Compared with the faster group, the range is 45.9–566 s and the mean value is 111.5 s. The median difference between both groups is 10 s.

*6.2.2. Stability*

Stability implies that the service should continue working as expected, even if peers go offline.

If several peers go offline in a swarm with a low peer set cardinality, one of the following consequences would happen: Either peers starve, since they do not have enough peers to finish the download, or the peers have to request the tracker for more peers. The last consequence results in an increase in download time. Both consequences disrupt the stability of the service. Thus, we run an experiment and counted the number of peers connected to the seeder. Figure 11 shows the number of connected peers to the seeder.

PI uses the tracker and the voting mechanism to get new peers. All the other seeding algorithms only use the tracker to get new peers. The seeder that makes use of PI had a connection to nearly all peers. This is because the seeder saves unknown peers from votes to a list for a possible connection candidates. If the seeder has fewer peers, it chooses randomly a peer from that list and tries to connect to it. Wu et al. [38] studied peer exchange in BitTorrent and concluded that peer exchange significantly reduces the download time. Therefore peer exchange lowers the dependency of a central tracker and increases the stability and robustness of BitTorrent.

*6.2.3. Security*

We now investigate how vulnerable the PI algorithm is to bandwidth attacks and compare the results with the other algorithms. For that purpose, we introduced 3 malicious peers that attack each algorithm in its own way. The attackers connect to the seeder 5 seconds before the leechers to ensure that the attackers are getting the unchoke slots first. We counted how many attackers and leechers were unchoked and depicted the results in Figure 12.

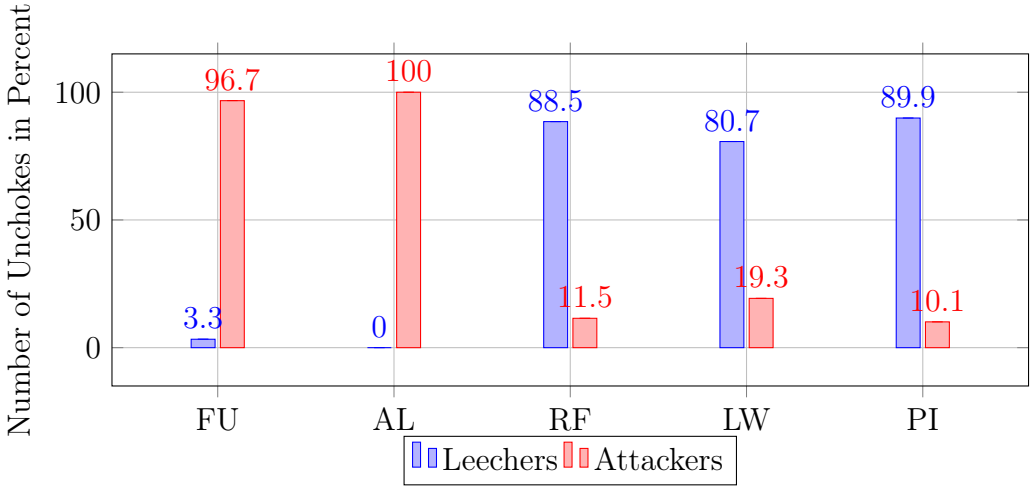



Figure 12: The unchoke ratio of attackers and leechers of the different seeding algorithms witout the optimistic unchoke slot. The data was produced with 1 seeder that has 5 Mbit/s upload limit, 29 leechers with 900 kbit/s download limit and 3 malicious peers. All values are average values of ten iterations. Source: [35].

To exploit the FU malicious peers just have to download faster than its competitors. Thus, we equipped the attackers with more bandwidth than its competitors. The attack script requests random blocks and tries to download as much as possible. The results in Figure 12 which shows, that FU is quite vulnerable against these attacks—only 3.3 % leechers were unchoked while 96.7 % attackers were unchoked.

In comparison to FU, RF unchokes 88.5 % leechers and only 11.5 % attackers. The probability that RF chooses an attacker is $P(A) = \frac{n_a}{n_p}$, where $n_a$ is the number of attackers and $n_p$ is the number of peers that the seeder has in its peer set. Therefore, the probability that a leecher is chosen by RF is $P(L) = \overline{P(A)} = \frac{n_p - n_a}{n_p}$. RF is quite robust against bandwidth attacks and gives each peer the same amount of pieces.

AL favors the peers that have nearly-all or nearly-none pieces. As a result, $F(p)$ from equation 3 is always 0, leading to $AL(p) = F$ which is the highest score for a peer. In other words, the benign leechers have to be content with the optimistic unchoke slot.

The script that attacks PI sends every 10 second a fake vote to the seeder, which contains the other attackers. A fake vote con only contain $n - 1$ attackers, since the requesting peer is not allowed to include itself to the vote. As the results depict in Figure 12, the PI algorithm is the most robust one against bandwidth attacks. It can be seen that PI only unchokes 10.1 % of the attackers and 89.9 % of the leechers.

## 7. Conclusions

This paper proposed an experimental evaluation of bandwidth attacks against seeding algorithms and presented a novel bandwidth attack that exploit the allowed fast extension. We have shown that the seeding algorithms FU, RR and AL are highly vulnerable. Multiple attackers were able to increase the average download time in our cluster testbed for all peers from 300 % to 414 % when the seeder made use of FU, RR or AL. Seeding algorithms RF and LW are less vulnerable, but it was still possible to increase the average download time from 70–150 %. We also provided large-scale experiments on PlanetLab to show the real-world effects of bandwidth attacks. While the attacks' impact on PlanetLab was a bit more contained than the cluster experiment, attack trends observed in the cluster testbed were validated by the PlanetLab experiments. We combined a Sybil attack with a bandwidth attack and showed that if an attacker introduces as many attackers as leechers, the average download time increases by up to 1000 %. We showed how a malicious peer can exploit the allowed fast extension to increase the impact of the bandwidth attacks significantly. In our analysis, we also found, reported, and helped in fixing a programming error in libtorrent that could be exploited by a malicious peer to gain more bandwidth. We also communicated our protocol enhancements to the libtorrent open-source community. Finally, we proposed a simple countermeasure to the allowed fast attack and a novel seeding algorithm that is resilient against bandwidth attacks.

## 8. Acknowledgement

We would like to thank the University of Applied Sciences Technische Hochschule Mittelhessen (THM) for providing us with the cluster we have used in our work. A very special thanks to Arvid Norberg, who worked with us to fix the round robin programming error and for this libtorrent library. We also thank Syed Fida Hussain Gilani for his help on using PlanetLab.

[1] E. Van Der Sar, BitTorrent Traffic Surges After LimeWire Shutdown, http://goo.gl/VJsya, 2011. Accessed: 21/06/2014.

[2] E. Van Der Sar, BitTorrent Traffic Increases 40 % in Half a Year, http://goo.gl/d4cGA, 2012. Accessed: 21/06/2014.

[3] P. Dhungel, D. Wu, B. Schonhorst, K. W. Ross, A measurement study of attacks on bittorrent leechers, in: Proceedings of the 7th International Workshop on Peer-to-Peer Systems, 2008, p. 7.

[4] P. Dhungel, X. Hei, D. Wu, A measurement study of attacks on bittorrent seeds, in: (ICC), 2011 IEEE, 2011, pp. 1–5.

[5] A. B. Loewenstern, BEP 0005: DHT Protocol, Technical Report, BitTorrent Inc., 2008. http://www.bittorrent.org/beps/bep_0005.html.

[6] User:AMC1, BitTorrent Peer Exchange Conventions, http://wiki.theory.org/BitTorrentPeerExchangeConventions, 2008. Accessed: 21/06/2014.

[7] A. Norberg, Local Peer Discovery Documentation, http://goo.gl/jAgTlC, 2009. Accessed: 21/06/2014.

[8] T. PirateBay, No more torrents=no changes anyhow, http://thepiratebay.se/blog/208, 2012. Accessed: 21/06/2014.

[9] P. Dhungel, D. Wu, X. Hei, B. Schonhorst, Is BitTorrent Unstoppable?, 2007.

[10] R. Sherwood, B. Bhattacharjee, R. Braud, Misbehaving TCP receivers can cause Internet-wide congestion collapse, in: Proceedings of the 12th ACM conference on Computer and communications security, 2005, pp. 383–392.

[11] F. Adamsky, S. A. Khayam, R. Jaeger, M. Rajarajan, Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver, in: International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2012, pp. 143–150.

[12] E. Adar, B. A. Huberman, Free Riding on Gnutella, First Monday 5 (2000).

[13] P. Dhungel, X. Hei, D. Wu, K. Ross, The seed attack: Can bittorrent be nipped in the bud?, Technical Report, Department of Computer and Information Science, Polytechnic Institute of NYU, 2008.

[14] P. Dhungel, D. Wu, K. W. Ross, Measurement and mitigation of BitTorrent leecher attacks, Elsevier Computer Communications 32 (2009) 1852–1861.

[15] N. Liogkas, R. Nelson, E. Kohler, L. Zhang, Exploiting BitTorrent For Fun (But Not Profit), in: Proceedings of the 5th International Workshop on Peer-to-Peer Systems, 2006, pp. 1–1.

[16] T. Locher, P. Moor, S. Schmid, R. Wattenhofer, Free Riding in BitTorrent is Cheap, in: Proceedings of the 5th Workshop on Hot Topics in Networks, 2006, pp. 85–90.

[17] K. El Defrawy, M. Gjoka, A. Markopoulou, BotTorrent: misusing BitTorrent to launch DDoS attacks, in: Proceedings of the 3rd USENIX workshop on Steps to reducing unwanted traffic on the Internet, 2007, pp. 1–6.

[18] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, Do incentives build robustness in BitTorrent?, in: Networked Systems Design and Implementation, 2007, pp. 1–14.

[19] F. Adamsky, H. Khan, M. Rajarajan, S. A. Khayam, POSTER: Destabilizing BitTorrent's Clusters to Attack High Bandwidth Leechers, in: ACM Computer and Communications Security 2011, 2011, pp. 1–1.

[20] B. Cohen, Incentives build robustness in BitTorrent, in: Workshop on Economics of Peer-to-Peer systems, 2003, pp. 1–1.

[21] A. Legout, G. Urvoy-Keller, P. Michiardi, Rarest first and choke algorithms are enough, in: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, New York, New York, USA, 2006, pp. 203–216.

[22] D. Harrison, B. Cohen, Fast Extension, Technical Report, BitTorrent Incorporation, 2008. http://www.bittorrent.org/beps/bep_0006.html.

[23] B. Cohen, Discussion of BEP 6: Fast Extension, 2008. Accessed: 21/06/2014.

[24] D. Eastlake 3rd, T. Hansen, US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF), 2011.

[25] E. Van Der Sar, uTorrent Keeps BitTorrent Lead, BitComet Fades Away, http://goo.gl/EImuS, 2011. Accessed: 21/06/2014.

[26] A. L. H. Chow, L. Golubchik, V. Misra, Improving BitTorrent: a simple approach, in: Proceedings of the 7th international conference on Peer-to-peer systems, IPTPS'08, 2008, p. 8.

[27] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, 1997.

[28] D. Harrison, Discussion of BEP 6: Fast Extension, 2008. Accessed: 06/08/2013.

[29] J. R. Douceur, The Sybil Attack, in: Proceedings of 1st International Workshop on Peer-to-Peer Systems, 2002, pp. 251–260. URL: http://www.springerlink.com/index/3an0ek5gfan3dtx9.pdf.

[30] Y. Namestnikov, The economics of botnets, Technical Report, Kaspersky Lab, 2009. https://www.securelist.com/en/analysis/204792068/The_economics_of_Botnets.

[31] M. Sirivianos, J. Han, P. Rex, C. X. Yang, Free-riding in BitTorrent Networks with the Large View Exploit, in: Proceedings of the 6th International Workshop on Peer-To-Peer Systems, 2007, pp. 1–7.

[32] Z. Chen, Y. Chen, C. Lin, N. V., P. Cao, Experimental Analysis of Super-Seeding in BitTorrent, in: Communications, 2008. ICC '08. IEEE International Conference on, 2008, pp. 65–69.

[33] R. L. Cottrel, How Bad Is Africa's Internet?, http://spectrum.ieee.org/telecom/internet/how-bad-is-africas-internet, 2013. Accessed: 12/06/2014.

[34] J. Hoffman, BEP 0016: Superseeding, Technical Report, BitTorrent Inc., 2008. http://www.bittorrent.org/beps/bep_0016.html.

[35] F. Adamsky, S. A. Khayam, R. Jaeger, M. Rajarajan, Who is going to be the next BitTorrent Peer Idol?, in: Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing, 2014, pp. 1–6.

[36] B. Cohen, The Bittorrent Protocol Specification, Technical Report, Bittorrent, Inc., 2008. URL: http://bittorrent.org/beps/bep_0003.html.

[37] S. Stahl, P. E. Johnson, Understanding Modern Mathematics, Jones & Bartlett Pub (Ma), 2006.

[38] D. Wu, P. Dhungel, X. Hei, C. Zhang, K. W. Ross, Understanding Peer Exchange in BitTorrent Systems, in: Proceedings of the 10th International Workshop on Peer-to-Peer Systems, 2010, pp. 1–8.