# Who is going to be the next BitTorrent Peer Idol?

Florian Adamsky*, Syed Ali Khayam†, Rudolf Jäger‡, Muttukrishnan Rajarajan*

*City University London, United Kingdom
Email: {Florian.Adamsky.1, R.Muttukrishnan}@city.ac.uk

†PLUMgrid, Inc., Sunnyvale, CA, USA
Email: akhayam@plumgrid.com

‡Technische Hochschule Mittelhessen University of Applied Sciences, Germany
Email: Rudolf.Jaeger@iem.thm.de

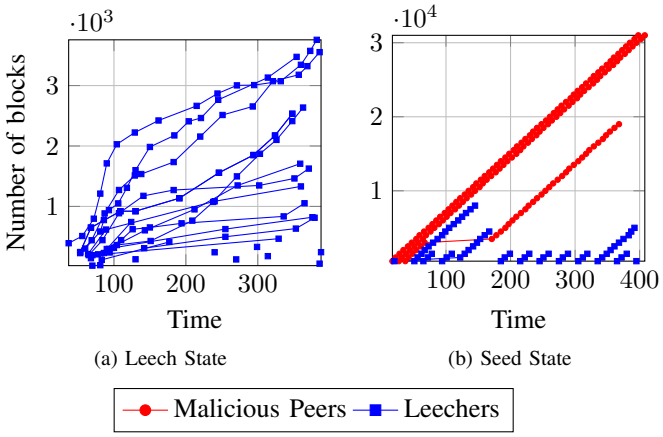
***Abstract*—Active measurement studies show that the Peer-to-Peer (P2P) file sharing protocol BitTorrent is highly under attack. Moreover, malicious peers can easily exploit the original seeding algorithm and therefore reduce the efficiency of this protocol. In this paper, we propose a novel seeding algorithm that requests peers to vote for their best sharing peers. Our results show that this incentive mechanism makes BitTorrent harder to exploit without losing performance. In some situations our algorithm even outperform other seeding algorithms. The peer exchange—that comes as a side effect—reduces the dependency on a centralized tracker and increases the robustness and the efficiency. We studied the effectiveness of our approach in a real testbed comprising 32 peers.**

## I. Introduction

Internet broadband penetration catalyzed a fundamental change in user's traffic characteristics with P2P file sharing. These protocols comprise 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 this kind of software. In addition to public litigation, active measurement studies [3], [4] have revealed many attacks on P2P systems, launched by companies hired by music and film industries. BitTorrent is the most successful P2P protocol and it causes the largest share of P2P traffic. Therefore, special attention must be given to its security. Hence, now it is important to investigate the threat landscape and to develop countermeasures.

Early P2P systems such as Gnutella made no use of an incentive mechanism. As a consequence, most of the peers were selfish and were only downloading but not uploading [5]. This problem is called *free-riding* and degrades the system performance and makes the network weak. Malicious peers can exploit this problem to attack the network. BitTorrent took this problem from the beginning into account and designed its peer selection algorithm in a tit-for-tat-ish way. This mechanism is called *choking algorithm*. A peer that has not the complete file (leecher) decides according to the download rate to whom it uploads.

The choking algorithm is, however, different when the peer has the complete file (seeder). According to *BitTorrent Enhancement Proposal (BEP)* 03 [6], a seeder uses its upload rate rather than its download rate to decide which peer to unchoke. In this state a seeder favors peers with the highest download rate. This does not provide a sharing incentive and can be easily exploited by malicious peers. Figure 1a depicts the upload piece distribution of the choking algorithm in leech state and Figure 1b in seed state. Every line represents another peer. The longer the line is, the longer the upload was.




(a) Leech State (b) Seed State

Fig. 1: Comparison between the upload piece distribution of the choking algorithms in leech state (a) and in seed state (b). The data was produced with 1 seeder that has 5 Mbit/s upload limit, 29 leechers with 1 Mbit/s and 3 fast peers with no limit.

The leecher in Figure 1a uploads pieces to multiple peers. This is the result of the incentive mechanism tit-for-tat. The peers from whom we have downloaded the most are allowed to upload from us. But the seeder in Figure 1b only uploads to a couple of peers, because these are the fastest downloaders. The 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 [7] [8]. To the best of our knowledge, the research community has not addressed this problem yet.

The main idea of this paper is a novel seeding algorithm that makes use of the voting method *Borda Count (BC)* to decide which peer is allowed to download. It requires peers to send votes for their best sharing peers periodically to all seeders. The seeders award each sharing peer with points according to the position in the vote. Afterwards, the algorithm sorts the list of requesting peers by their score and allows the most sharing peers to download. This provides sharing incentive and is more resilient against malicious peers than previous algorithms. As a side effect, the algorithm exchanges peer information with the seeder and can therefore increase the stability and the robustness of BitTorrent. The contributions of this paper are as follows:

- We explain the details of our novel seeding algorithm and show its implementation details in Section III.
- We analyze the performance of this algorithm in different environments in a real testbed system with 32 peers running a commonly-used BitTorrent library in Section IV-B.
- We investigate the peer exchange mechanism in Section IV-C and show that a seeder with our algorithm has connections to more peers.
- In Section IV-D we attacked each seeding algorithm in its own way and outline their vulnerability.

## II. Background and Problem

In this section we first introduce the BitTorrent terminology used in this paper. We then briefly discuss the used BitTorrent's choking algorithms and show their weaknesses.

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

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

**Free Rider:** A peer, which only downloads data and denies uploading to other peers.

**Peer Set:** Each peer maintains a list of other peers that it knows about within a swarm. We call this list peer set.

**Active Peer Set:** Active peer set for a peer is the subset of its peer set that it can send data to.

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

**Interested:** $P$ has data that $Q$ wants to acquire.

**Piece:** The downloaded file is divided into equally sized parts called Pieces.

BitTorrent's success can be reduced to its piece and peer selection algorithm, which are: rarest piece first and choking algorithm. Since every peer has a limited number of *unchoke slots* [1], the choking algorithm decides which peer from the peer set is getting a slot. The decision criterion becomes different whether the peer is a leecher or a seeder. In leech state, the choking algorithm works in a tit-for-tat-ish way and favors peers who uploads. For instance, a leecher $L$ has an upload capacity of 30 and two upload slots. In the previous upload round $L$ downloaded from the peers $A$, $B$, $C$ with a rate of 10, 5 and 15. In the next round, the choking mechanism would choke peer $B$ and unchoke peer $A$ and $C$, since they shared most.

[1]We use the term unchoke slot and upload slot interchangeably.

If this mechanism would not have an exception, new peers would not get a chance to join the swarm, since they have nothing to share. This exception exists in leech and seed state and is called *optimistic unchoke slot*. A peer assigns this slot to a random peer. Thus, a peer can try out new peers. Leecher and seeder manage $u$ unchoke slots and $o$ optimistic unchoke slots. The BitTorrent specification sets $u = 3$ and $o = 1$ and lists the following reasons for choking: a) it prevents free riders, b) it ensures a consistent download rate, c) the TCP congestion control behaves poorly, when sending over many connections at once.

The choking algorithm in seed state does not work in a tit-for-tat-ish way, since the seeder does not need anything from the other peers. The following seeding algorithms have been included to different BitTorrent clients:

***Fastest Upload (FU)*** is the original seeding algorithm described in BEP 03 [6]. It favors the peers who upload the fastest. This algorithm is still used by around 29.5 % of the clients, according to Table I.

***Round Robin (RR)*** is a well-known algorithm, especially as a scheduling algorithm for processes in *operating systems (OSs)*. In context of BitTorrent, it gives each peer a constant number of pieces and rotates every $n$ pieces.

***Longest Waiter (LW)*** was introduced with the BitTorrent mainline client version 4.0.0 without an announcement. It was first mentioned by Legout et. al. [7]. This algorithm sorts all unchoked and interested peers according to their time they were last unchoked. LW unchokes the $u$ peers who have wainted the longest. Each unchoked peer gets at least two rounds until LW unchokes them.

***Anti Leech (AL)*** is a seeding algorithm, introduced by Chow et. al. [9]. They found out that a 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 counteract this situation, their seeding algoritm prefers peers when they only have a few pieces or when they have nearly all pieces. The following equation is used to calculate a score for each peer:

$$AL(p) = \begin{cases} F - F(p) & \text{if } (F(p) < \frac{F}{2}) \\ F(p) \cdot \frac{1000}{F} & \text{otherwise,} \end{cases} \quad (1)$$

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 $F(p)$.

Table I lists prominent BitTorrent clients and their default choking algorithm in seed state, arranged by their market share based on the statistics from [10].

All seeding algorithms that we introduced, have at least one of the following problems:

### A. Missing Incentive Mechanism

According to BitTorrent's choking mechanism, a selfish peer who does not upload to others would get punished. The same peer would get rewarded by a seeder with an unchoke slot, if

TABLE I: BitTorrent clients in combination with the used Seeding Algorithm order by market share according to [10].

| # | 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 |

the peer downloads fast enough. Let us take the example from Section II to illustrate the problem: if peer $A$ and peer $B$ have the same bandwidth capacity and $A$ is selfish and $B$ is not. Peer $A$ downloads from $S$ with its full rate. But $B$ uploads and downloads pieces with other peers and thus is not able to download with the full rate. Therefore, $S$ would unchoke $A$ instead of $B$.

Liogkas et. al. [11] implemented such a selfish BitTorrent client that ignores leechers and only tries to download from seeders. Seeders can be easily identified, since they advertise themselves by sending a `HAVE_ALL` message or a complete bitfield. Consequently, this means that the seeding algorithm tolerates free riding which can be explained by the missing incentive mechanism.

### B. Security Problems

The choking algorithm in seed state is vulnerable to bandwidth attacks. The idea behind a bandwidth attack is to disturb the distribution of files via BitTorrent. A seeder manages $u$ unchoke slots and $o$ optimistic slots. A malicious peer that runs a bandwidth attack tries to occupy most of the seed's unchoke slots. If an attack is successful and the seeder's unchoke slots are all occupied by attackers, then the seeder distributes $o \cdot \frac{b}{u+o}$ of its upload bandwidth $b$ to good peers. Our previous work [12] has shown that 3 malicious peers can degrade the download rate up to 415 % for all peers. Combined with a Sybil attack [13] that consists of as many attackers as leechers, it is possible to degrade the download rate by ten times.

## III. Seeding Algorithm: Peer Idol

We propose a new seeding algorithm that is fast, hard to exploit and ensures that only peers that have shared are getting unchoked. We call this seeding algorithm *Peer Idol (PI)*. If a leecher wants to acquire pieces from a seeder, it has to send every unchoke round a new BitTorrent message to the seeder, that we call a *vote*. This vote contains a list of $n$ peers. The peer who sends the vote chooses leechers according to their download rate. The notation $A \succ B$ indicates that the requesting peer has downloaded more from peer $A$ than from peer $B$. Therefore a vote with $n = 3$ looks like $(A \succ B \succ C)$.

The seeder who gets the vote awards each peer that is in the vote with points: Peer $A$ gets 3 points, peer $B$ gets 2 points and peer $C$ gets 1 point. Every unchoke round the seeder sorts all candidates by their score of each peer and unchokes the peers with the highest score. These peers can download for two consecutive rounds until the seeder calculates the scores again. This avoids quick choking and unchoking, known as *fibrillation*. In mathematical terms: if $N = \{1, 2, \ldots, n\}$ is the peer set of the seeder, then $I \subseteq N$ contains the peers which are interested. Then for every peer $p \in I$ we calculate the PI score as follows:

$$PI(p) = \sum_{i=1}^{|I|} V_i(p). \tag{2}$$

The function $V_i(p)$ returns 1 if peer $i$ voted for peer $p$ and 0 if not. If two peers have the same score, the peer that has waited the longest gets a higher priority. After each unchoke round, which BEP 03 defines as 10 seconds, the score of all peer is reset to 0. If a vote contains peers to whom the seeder is not connected to, it will add this peer in a list of potential peer candidates. This list contains peers from *Local Peer Discovery (LPD)*, *Peer Exchange (PEX)* and *Distributed Hash Table (DHT)*. If the seeder has less peers than `max_connection`, it randomly chooses peers from this list and tries to connect to it.

The scoring is a well-known election method in political voting and is called *Borda Count (BC)*, named after Jean-Charles de Borda [14], although it was developed multiple times independently. We also have tested the *Condorcet Method (CM)*, but noticed two major problems: Firstly, the computation complexity of CM is $\mathcal{O}(N^2)$, since all peers have to compete with each other. In comparison, the complexity of BC is $\mathcal{O}(1)$, since the seeder adds the increases the vote counter of that specific peer. Secondly, the score of the last peer would be $CM(p) = 0$, since it would loose all pairwise comparisons.

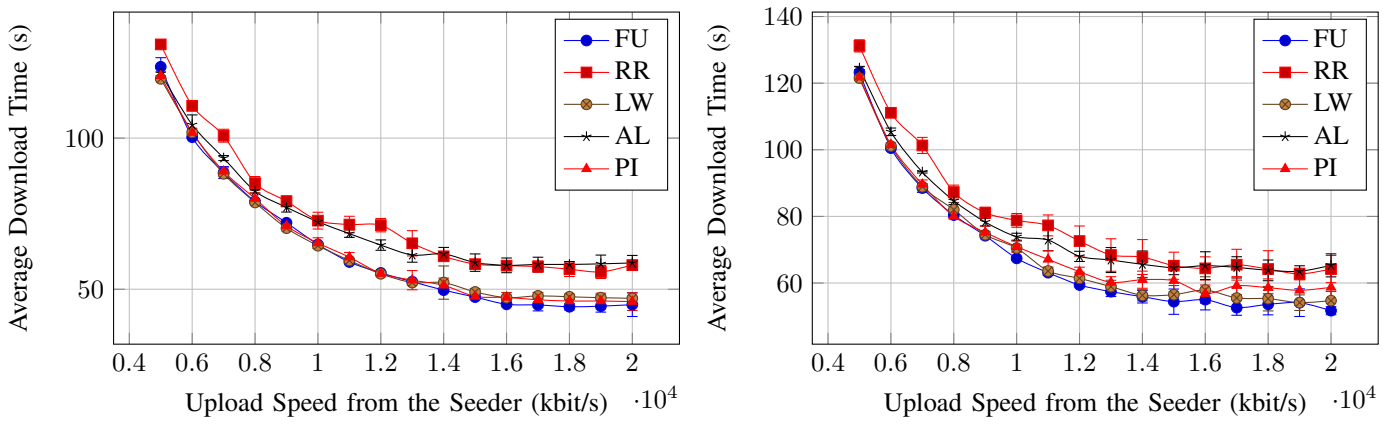
We defined additional security rules to protect against fraud. A peer has to send a vote in order to get voted. This ensures, that peers send votes to the seeder. But an exception can be made, if the seeder has not enough peers to upload. A peer gets disconnected and blacklisted if the vote contains:

- more than $n$ peers
- the IP address of the requesting peer
- repeated peers

### A. Implementation details

We implemented PI as an extension of the BitTorrent protocol based on BEP 10 [15]. The advantage of this approach is that we are not interfering with the standard protocol. Moreover, the BitTorrent handshake contains reserved bits that can be used to communicate which extension a peer supports. This makes it easy for a peer to send votes only to seeders that supports it. We set the number of votes to three through out all experiments—it is the same number that BEP 03 sets as unchoke slots. Yet PI has an extra message overhead through the votes.

A PI vote contains a four bytes length field, followed by a one byte *Message Type (MT)*. The message type of an extension message is 20, according to BEP 10 [15]. The next field is a one byte *Extended Message Type (EMT)* that distinguishes the extension. The payload of this packet is a




(a) Leechers have no upload or download limit

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

Fig. 2: Effects of the different seeding algorithms on the average download speed in different environments. The upload speed of the seeder was gradually increased. The error bars show 99 % confidence intervals.

bencoded dictionary with an item `vote`, that contains the voted peers in *compact* [2] format. We have not considered IPv6 in our implementation. But instead of the item `vote`, it would be possible to use `vote6` and then include the 18-octet IPv6 addresses. Considering the 6 bytes of static fields of the extension message and 12 bytes bencoded dictionary overhead, a PI message with 3 votes would have 28.8 kbit/s for IPv4 and 57.6 kbit/s for IPv6.

## IV. Experimental Evaluation

### A. Experimental Setup

We perform experiments using private torrents in our testbed consisting of 32 nodes with an additional node that controls and monitors the experiments. These nodes are desktop machines, which are running libtorrent 0.16.10 over Ubuntu GNU Linux 12.04.1 LTS. In all experiments, the seeder distributes a file with a size of 500 MiB and a piece size of 256 KiB. The controller node executes the experiments and monitors each of these nodes.

All experiments simulated a flash crowd scenario to get reproducible results with 1 seeder and 31 leechers. 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.

### B. Performance

Performance is the unique selling point of BitTorrent. A new seeding algorithm should not make BitTorrent in any case slower. Therefore, we designed four performance experiments to find out how fast PI is in different environments. Our hypothesis is that PI will not be slower than the other seeding algorithms, even with the message overhead described in Section. This is based on the observation that PI favors sharing peers. In the first experiment, we compared the performance of the seeding algorithms in an optimal environment, where the leechers had no download or upload limits. The seeder had an upload speed limit that was gradually increased. Figure 2a shows the average download speed of all peers depending on the upload speed from the seeder.

Figure 2a shows the results of the first experiment. We limited the domain of all results to 5–20 Mbit/s, since there is no significant difference between the algorithms in the domain from 1–4 Mbit/s. Beginning from 5 Mbit/s RR and AL move apart from FU, LW and PI. To reduce redundancy, RR represents the slower group and PI the faster group. The seeding algorithm RR ranges from 55.7–567 s and has a mean value of 120.5 s. Compared with 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.

### C. Stability

We understand stability in terms of BitTorrent that the service still works as expected, even if some peers go offline. Let us suppose, several peers would go offline in a swarm with a low peer set cardinality. The consequence would be either that peers starve, since they do not have enough peers to finish the download, or the peers have to request the tracker for more peers, which 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 the seeder has a connection to. Table II shows the number of connected peers to the seeder.

TABLE II: The number of connected peers of the seeder.

| Seeding Algorithms: | FU | RR | LW | AL | PI |
|---|---|---|---|---|---|
| Number of Connected Peers: | 5.1 | 5.1 | 4.8 | 4.9 | 29.0 |

All seeding algorithms except PI only use the tracker to

[2]The compact format represents an IPv4 address with 6 bytes: 4 bytes IPv4 address and 2 bytes port number.

get new peers. The results show that these algorithms request the tracker once or twice to get new peers. The seeder that makes use of PI had a connection to nearly all peers. This is because with every vote that contains unknown peers, the seeder save these peers to a list for a possible connection candidates. If the seeder has to less peers, it chooses randomly a peer from that list and tries to connect it. Wu et. al. [16] studied peer exchange in BitTorrent and concluded that peer exchange significantly reduces the download time. This lowers the dependency of a central tracker and increases the stability and robustness of BitTorrent.

### D. Security

In the next experiment, we investigate how vulnerable the PI algorithm is to bandwidth attacks and compare the results with the other algorithms. For that purpose, we included 3 malicious peers in the experiment that attack each algorithm in its own way. The attackers connect to the seeder 5 seconds before the leechers. This ensures that the attackers are getting the unchoke slots first. We measured how many attackers and leechers were unchoked. The comparision can be seen in Figure 3.

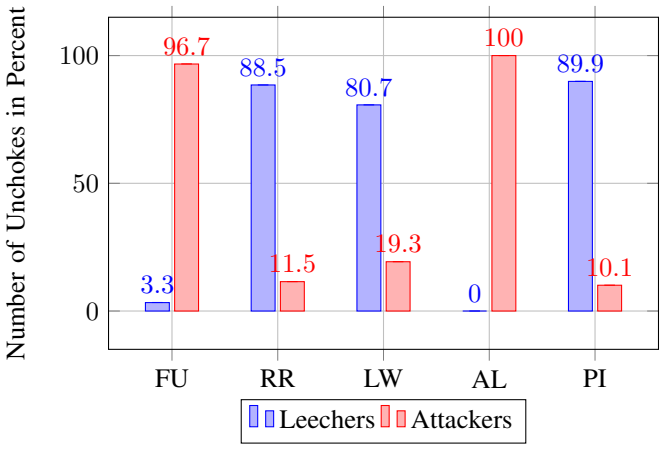



Fig. 3: The unchoke ratio of attackers and leechers of the different seeding algorithms witout the optimistic unchoke slot. The data was produced with 1 seeder that has 5 Mbit/s upload limit, 29 leechers with 900 kbit/s download limit and 3 malicious peers. All values are average values of ten iterations. The 99 % confidence intervals of all values is $< 0.05$.

The malicious peers just have to download faster than its competitors to exploit the FU. 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 3 show, that FU is quite vulnerable against those kind of attacks. Only 3.3 % leechers were unchoked and 96.7 % attackers. This confirms that FU unchokes the fastest downloaders and since the attackers are faster and earlier, FU unchokes almost exclusively the attackers.

To attack RR we used the same attack script against FU, since there is no vulnerability that can be exploited, except to introduce more attackers. In comparison to FU, RR unchokes 88.5 % leechers and only 11.5 % attackers. The probability that RR chooses an attacker is $P(A) = \frac{n_a}{n_p}$, where $n_a$ is the number of attacker 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 RR is $P(L) = \overline{P(A)} = \frac{n_p - n_a}{n_p}$. RR 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. To exploit this algorithm, the attacker are not sending any `HAVE` messages to the seeder. As a result, $F(p)$ from Equation 1 is always 0 and therefore $AL(p) = F$, which is the highest score for a peer. The benign leechers have to be content with the optimistic unchoke slot. Their AL score is getting lower than the malicious peers, as soon as they have one piece. This security weakness is described in the discussion section of their paper [9]. The authors also presented an idea to prevent that attack, which is not implemented in the current libtorrent version.

To exploit PI, the attack script sends every 10 second a vote to the seeder, which contains the other attackers. The attackers can only vote for $n - 1$ attackers, since the requesting peer is not allowed to include itself to the vote. This means, the vote from the attackers only contains two valid peers. As the results depict, 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.

What impact has the unchoke ratio on the average download time of the leechers? In the next experiment we included 3 malicious peers with no upload and download limit and 29 leechers with an upload limit of 20 Mbit/s. Like in the previous experiment, we attacked each seeding algorithm in its own way and increased the upload limit of the seeder gradually. The results can be seen in Figure 4.

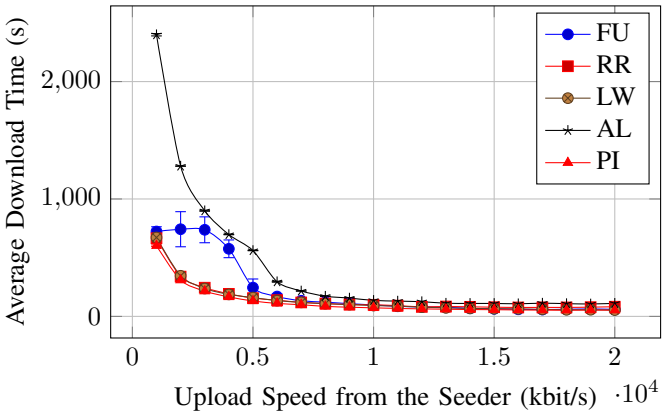



Fig. 4: Average download speed of the different seeding algorithm with 3 attackers with no up or download limit and 29 leechers with an upload limit of 20 Mbit/s. The error bars show 99 % confidence intervals.

The leechers suffer most from the attack against a seeder that uses AL as its seeding algorithm. Their average download time range from 105.6–2398.8 s which results in a mean of 397.1 s. The next algorithm that is quite vulnerable is FU. Average download time from the leechers with seeder with FU range from 58.5–742.4 s with a mean of 216.6 s. We observed a higher deviation when the seeder has a low upload capacity. A low upload capacity entails that the attackers and leechers are downloading nearly with the same speed. This means, it depends on the efficiency of the attackers and leechers.

The download time from the leechers of RR range from 76.2–665.7 s with a mean of 148.5 s. After 12 Mbit/s, RR becomes slower than LW. LW, however, ranges from 53.5–671.8 s with an average value of 140.9 s. The seeding algorithm PI is the fastest one. The average download time of the leechers range from 52.5–602.2 s with a mean of value of 122.2 s.

## V. Related Work

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

A number of research studies (e.g. [17], [11], [8], [18]) have focused the choking algorithm in leech state and investigated its fairness, robustness and performance. But only a few studies have focused the seeding algorithm. One of these few studies is the already introduced seeding algorithm AL from Chow et. al. [9] in Section II. To the best of knowledge, this is the only new seeding algorithm that has been proposed by the research community, yet.

Dhungel et. al. [19] were the first who investigated bandwidth and connection attacks in the area of 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. On the contrary, our previous work [12] shows that bandwidth attack can be an effective attack against seeders.

Piatek et. al. [20] found out by month-long measurement of millions of BitTorrent users, that the performance and availability in BitTorrent is 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. This is restricted to at most one level of intermediaries. This protocol limits free-riding. It is, however, hard to compare their protocol with our seeding algorithm, since it is a complete new protocol based on one hop reciprocation.

## VI. Conclusion and Future Work

We considered an important threat against seeding algorithms in BitTorrent. Our paper proposed a countermeasure against bandwidth attacks. This novel seeding algorithm for BitTorrent that we call Peer Idol introduces a new message type which contains votes for other peers. We evaluated this algorithm experimentally in terms of performance, security and stability. Our results support our hypothesis that PI is more robust against bandwidth attacks and does not lose performance compared to other algorithms. In our experiment, PI was even faster than RR and AL. We have shown that if a vote contains peers, which the seeder has not in its peer set, this is not a disadvantage. These peers, however, are getting saved to a candidate list and can be used for later contact. This reduces the dependency of a central tracker and increases the stability and robustness of BitTorrent. Summarized, our novel choking algorithm in seed state implements the incentive mechanism in BitTorrent consequently through all peers.

## References


[1] E. Van Der Sar. (2011) BitTorrent Traffic Surges After LimeWire Shutdown. [Online]. Available: http://goo.gl/VJsya

[2] ——. (2012) BitTorrent Traffic Increases 40 % in Half a Year. [Online]. Available: http://goo.gl/d4cGA

[3] P. Dhungel, D. Wu, B. Schonhorst, and K. W. Ross, "A measurement study of attacks on bittorrent leechers," in *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008, p. 7.

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

[5] E. Adar and B. A. Huberman, "Free Riding on Gnutella," *First Monday*, vol. 5, 2000.

[6] B. Cohen, "The Bittorrent Protocol Specification," Bittorrent, Inc., Tech. Rep., Feb 2008. [Online]. Available: http://bittorrent.org/beps/bep_0003.html

[7] A. Legout, G. Urvoy-Keller, and 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.

[8] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?" in *Networked Systems Design and Implementation*, 2006, pp. 1–14.

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

[10] E. Van Der Sar. (2011) uTorrent Keeps BitTorrent Lead, BitComet Fades Away. [Online]. Available: http://goo.gl/EImuS

[11] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)," in *5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, feb 2006, pp. 1–1.

[12] F. Adamsky, S. A. Khayam, R. Jr, and M. Rajarajan, "Experimental Evaluation of Bandwidth Attacks against Seeder," 2014, unpublished.

[13] J. R. Douceur, "The Sybil Attack," in *IPTPS*, 2002, pp. 251–260. [Online]. Available: http://www.springerlink.com/index/3an0ek5gfan3dtx9.pdf

[14] S. Stahl and P. E. Johnson, *Understanding Modern Mathematics*. Jones Bartlett Pub (Ma), 2006.

[15] A. Norberg, L. Strigeus, and G. Hazel, "BEP 0010: Extension Protocol," BitTorrent Inc., Tech. Rep., 2008. [Online]. Available: http://bittorrent.org/beps/bep_0010.html

[16] D. Wu, P. Dhungel, X. Hei, C. Zhang, and K. W. Ross, "Understanding Peer Exchange in BitTorrent Systems," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, 2010, pp. 1–8.

[17] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free Riding in BitTorrent is Cheap," in *Hot Nets 5*, 2006, pp. 85–90.

[18] S. Jun and M. Ahamad, "Incentives in BitTorrent Induce Free Riding," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, ser. P2PECON '05. ACM, 2005, pp. 116–121.

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

[20] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson, "One hop Reputations for Peer to Peer File Sharing Workloads," in *Networked Systems Design and Implementation (NSDI) 08*, 2008, pp. 1–14.