Free Riding in BitTorrent is Cheap

Thomas Locher[1], Patrick Moor[2], Stefan Schmid[1], Roger Wattenhofer[1]
[1] Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland
{lochert, schmiste, wattenhofer}@tik.ee.ethz.ch
[2] Google Inc., Mountain View, CA 94043, USA
pmoor@google.com

ABSTRACT

While it is well-known that BitTorrent is vulnerable to selfish behavior, this paper demonstrates that even entire files can be downloaded without reciprocating at all in BitTorrent. To this end, we present *BitThief*, a free riding client that never contributes any real data. First, we show that simple tricks suffice in order to achieve high download rates, *even in the absence of seeders*. We also illustrate how peers in a swarm react to various sophisticated attacks. Moreover, our analysis reveals that *sharing communities*—communities originally intended to offer downloads of good quality and to promote cooperation among peers—provide many incentives to cheat.

1 INTRODUCTION

As pure peer-to-peer (p2p) systems are completely decentralized and resources are shared directly between participating peers, all p2p systems potentially suffer from *free riders*, i.e. peers that eagerly consume resources without reciprocating in any way. Not only do free riders diminish the quality of service for other peers, but they also threaten the existence of the entire system.

For that reason, it is crucial for any system without centralized control to incorporate a rigorous incentive mechanism that renders freeloading evidently unattractive to selfish peers. Unfortunately, however, many solutions so far either could easily be fooled or were unrealistically complex. Bram Cohen's BitTorrent protocol heralded a paradigm shift as it demonstrated that cooperation can be fostered among peers interested in the same file, and that concentrating on one file is often enough in practice. The fair sharing mechanism of BitTorrent is widely believed to strongly discourage freeloading behavior.

Contrary to such belief, we show that BitTorrent in fact does not provide sufficient incentives to rule out free riding. The large degree of cooperation observed in BitTorrent swarms is mainly due to the widespread use of obedient clients which willingly serve all requests from other peers. We have developed our own BitTorrent client *BitThief*[1] that never serves any content to other peers. With the aid of this client, we demonstrate that a peer can download content fast *without uploading any data*. Surprisingly, BitThief always achieves a high download rate, and in some experiments has even outperformed the *official client*. Moreover, while *seeders* ("altruistic peers") clearly offer the opportunity to freeload, we are even able to download content quickly if we ignore seeders and download solely from other peers that do not possess all pieces of the desired content (*leechers*). This implies that the basic piece exchange mechanism does not effectively restrain peers from freeloading.

Sharing communities are also investigated in this paper. By banning users with constantly low sharing ratios or by denying them access to the newest torrents available, such communities encourage users to upload more than they download, i.e., to keep their sharing ratio above 1. We will show that sharing communities are particularly appealing for free riders, and that cheating is easy.

We believe that the possibility to freeload which does not come at the cost of a considerably reduced quality of service (e.g., download rate) is attractive for users: Not only because wasting more expensive upload bandwidth is avoided, but also because—in contrast to downloading—merely the *distribution* of copyrighted media content such as music or video shared in p2p networks is unlawful in certain countries.

However, as more and more users decide to free ride, the usefulness of a p2p system will naturally decline. Thus, spreading such freeloading clients might prove to be an effective attack for corporations fighting the uncontrolled distribution of their copyrighted material.

2 BITTORRENT

The main mechanisms applied by *BitTorrent* are described in [4]; for additional resources including a detailed technical protocol, the reader is referred to *www.bittorrent.org*. Basically, BitTorrent is a p2p application for sharing files or collections of files. In order to participate in a *torrent download*, a peer has to obtain a *torrent metafile* which contains information about the content of the torrent, e.g. file names, size, tracker addresses, etc. A tracker is a centralized entity that keeps track of all the peers (TCP endpoints) that are downloading in a specific torrent swarm.[2] Peers obtain contact information of other participating peers by announcing themselves to the tracker on a regular basis. The data to be shared is divided into pieces whose size is specified in the metafile (usually a couple of thousand pieces per torrent). A hash of each piece is also stored in the metafile, so that the downloaded data can be verified piece

[1] Available at http://dcg.ethz.ch/projects/bitthief/.

[2] Recently, a distributed tracker protocol has been proposed. It is implemented by most modern clients.

by piece. Peers participating in a torrent download are subdivided into *seeders* which have already downloaded the whole file and which (altruistically) provide other peers with any piece they request, and *leechers* which are still in progress of downloading the torrent. While seeders upload to all peers (in a round robin fashion), leechers upload only to those peers from which they also get some pieces in return. The peer selection for uploading is done by unchoking a fixed number of peers every ten seconds and thus enabling them to send requests. If a peer does not contribute for a while it is choked again and another peer is unchoked instead.

The purpose of this mechanism is to enforce contributions of all peers. However, each leecher periodically *unchokes* a neighboring leecher, transferring some data to this neighboring peer for free (called *optimistic unchoking* in BitTorrent lingo). This is done in order to allow newly joined peers without any pieces of the torrent to *bootstrap*. Clearly, this unchoking mechanism is one weakness that can be exploited by BitThief.

3 BitThief: A Free Riding Client

In this section we provide evidence that, with some simple tricks, uploading can be avoided in BitTorrent while maintaining a high download rate. In particular, our own client *BitThief* is described and evaluated. BitThief is written in Java and is based on the official implementation[3] (written in Python, also referred to as *official client* or *mainline client*), and the *Azureus*[4] implementation. We kept the implementation as simple as possible and added a lot of instrumentational code to analyze our client's performance. BitThief does not perform any chokes or unchokes of remote peers, and it never announces any pieces. In other words, a remote peer always assumes that it interacts with a newly arrived peer that has just started downloading. Compared to the official client, BitThief is more aggressive during the startup period, as it re-announces itself to the tracker in order to get many remote peer addresses as quickly as possible. The tracker typically responds with 50 peer addresses per announcement. This parameter can be increased to at most 200 in the announce request, but most trackers will trim the list to a limit of 50. Tracker announcements are repeated at an interval received in the first announce response, usually in the order of once every 1800 seconds. Our client ignores this number and queries the tracker more frequently, starting with a configurable interval and then exponentially backing off to once every half an hour. Interestingly, during all our tests, our client was not banned by any of the trackers and could thus gather a lot of peers. The effect of our aggressive behavior is depicted in Figure 1. Finally, note that it would also be possible to make use of the *distributed tracker protocol*.[5] This protocol is useful if the main tracker is not operational. Thus far, we have not incorporated this functionality into our client however.

[3] See http://bittorrent.com/.

[4] See http://azureus.sourceforge.net/.

[5] See http://www.bittorrent.org/Draft_DHT_protocol.html.

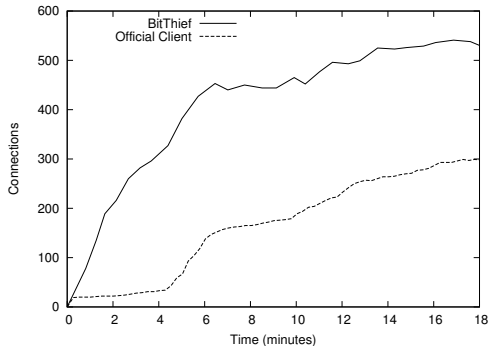

Figure 1: Number of open connections over time. In comparison to the official client, BitThief opens connections much faster.

Having a large number of open connections improves the download rate twofold: First, connecting to more seeders allows our client to benefit more often from their round robin unchoking periods. Second, there will be more leechers in our neighborhood that include BitThief in their periodical optimistic unchoke slot. Opening more connections increases download speed linearly, as remote peers act independently of the number of our open connections. However, note that opening two connections to the same peer does not help, as the official client, Azureus, and presumably all other clients as well immediately close a second connection originating from the same IP address.

Our experiments with BitThief demonstrate that the common belief that the performance will degrade if a large number of TCP connections is maintained simultaneously is unfounded. On the contrary, more connections always help to increase the download rate when using BitThief. The reason why the total number of TCP connections is kept small in BitTorrent might be that a moderate number of connections suffice to saturate the average user's bandwidth when following the real protocol, and no further gain could be achieved by connecting to more peers.

In contrast to other BitTorrent clients, BitThief does not apply the so-called *rarest-first policy*, but uses a simpler piece selection algorithm instead: We fetch whatever we can get. If our client is unchoked by a remote peer, it picks a random missing piece. Our algorithm ensures that we *never* leave an unchoke period unused. Furthermore, just like all other BitTorrent clients, we strive to complete the pieces we downloaded partially as soon as possible in order to check them against the hash from the metafile and write them to the harddisk immediately.

3.1 Seeders

We first tested the client on several torrents obtained from *Mininova*[6] and compared it to the official client.[7] By default, the official client does not allow more than 80 connections. In order to ensure a fair comparison, we removed this limitation and permitted the client to open up to 500

[6] See http://www.mininova.org/.

[7] Official client vers. 4.20.2 (linux source). Obtained from bittorrent.com, used with parameters: `--min_peers 500 --max_initiate 500 --max_allow_in 500`.

	Size	Seeders	Leechers	μ	σ
A	170MB	10518 (303)	7301 (98)	13	4
B	175MB	923 (96)	257 (65)	14	8
C	175MB	709 (234)	283 (42)	19	8
D	349MB	465 (156)	189 (137)	25	6
E	551MB	880 (121)	884 (353)	47	17
F	31MB	N/A (29)	N/A (152)	52	13
G	798MB	195 (145)	432 (311)	88	5

Table 1: Characteristics of our test torrents. The numbers in parentheses represent the maximum number of connections BitThief maintained concurrently to the respective peer class and is usually significantly lower than the peer count the tracker provided. μ and σ are the average and standard deviation of the official client's download times in minutes. The tracker of Torrent F did not provide any peer count information. Based on the number of different IP addresses our client exchanged data with, we estimate the total number of peers in this torrent to be more than 340.

connections. In a first experiment, we did not impose any restrictions on our client, in particular, BitThief was also permitted to download from seeders. The tests were run on a PC with a public IP address and an open TCP port, so that remote peers could connect to our client. We further blocked all network traffic to or from our university network, as this could bias the measurements. The properties of the different torrents used in this experiment are depicted in Table 1. Note that the tracker information is not very accurate in general and its peer count should only be considered a hint on the actual number of peers in the torrent.

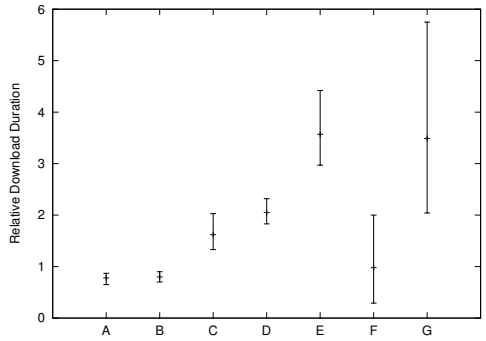

Figure 2: Relative download times for six torrents. The download time of the official client is normalized to 1.0. Every torrent was downloaded three times with both clients. The plot shows relative download times with the fastest run at the lower end of the bar, the average running time at the level of the horizontal tick mark, and the slowest run at the upper end of the bar.

The results are summarized in Figure 2. As a first observation, note that in every experiment, BitThief succeeded eventually to download the entire file. More interestingly, the time required to do so is often not much longer than with uploading! Exceptions are Torrents E and G, where there are relatively few seeders but plenty of leechers. In that case, it takes roughly four times longer with our client. However, the download came at a large cost for the official client as it had to upload over 3.5GB of data. Torrents A, B and F also offer valuable insights: In those torrents, BitThief was, on average, slightly faster than the official client, which uploaded 232MB in a run of torrent A and 129MB in a run of Torrent B. We conclude that in torrents with many peers, particularly seeders, and in torrents for small files, BitThief seems to have an advantage over the official client, probably due to the aggressive connection opening.

3.2 Leechers

In this section, we further constrain BitThief to only download from other leechers. Interestingly, as we will see, even in such a scenario, free riding is possible.

Seeders are identified by the *bitmask* the client gets when the connection to the remote peer is established, and the *have-message* received every time the remote peer has successfully acquired a new piece. As soon as the remote peer has accumulated all pieces, we immediately close the connection. We conducted the tests at the same time as in Section 3.1 and also used the same torrents. The running times are depicted in Figure 3. It does not come as a surprise that the average download time has increased. Nevertheless, we can again see that all downloads finished eventually. Moreover, note that the test is slightly unfair for BitThief, as the official client was allowed to download not only from the leechers, but also from all seeders! In fact, in some swarms only a relatively small fraction of all peers are leechers. For example in Torrent C, merely 15% are leechers, and BitThief can thus download from less than a sixth of all available peers; nevertheless, BitThief only requires roughly 5 times longer than the official client.

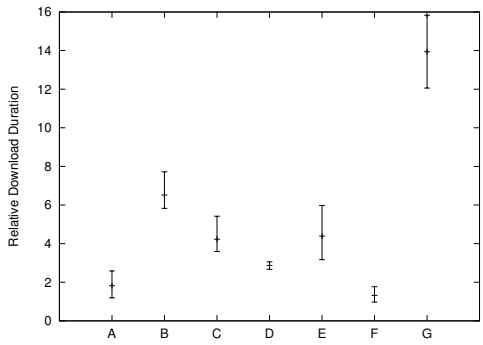

Figure 3: Relative download times of BitThief for six torrents *without downloading from any seeders*. The download time of the official client is normalized to 1.0. As in the first experiment, the torrents were downloaded three times with the official client and three times using BitThief restricted to download from leechers only. The bars again represent the same minimum, average and maximum running times.

We conclude that even without downloading from seeders, BitThief can download the whole torrent from leechers exclusively. Therefore, it is not only the seeders which provide opportunities to free ride, but the leechers can be exploited as well.

3.3 Further Experiments

The measurements presented so far have all been obtained through experiments on the Internet and hence were subject to various external effects. For example, in case BitThief

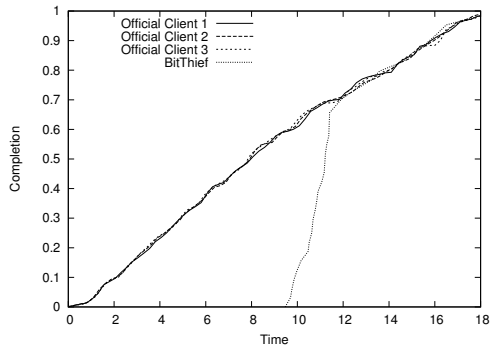

Figure 4: Download times for three official clients and one BitThief client in the presence of a slow seeder. BitThief starts downloading 9 minutes later than the other clients, but catches up quickly. Ultimately, all clients finish the download roughly at the same time.

was allowed to download from seeders, it sometimes downloaded at a high rate, but then—a few minutes later—the download rate declined abruptly due to a powerful seeder having left the network. In order to get reproducible results, we set up a pet network environment on a host, consisting of a private tracker, a configurable number of official clients as seeders and leechers, and one instance of our own client. We evaluated different scenarios. In the following, our main findings will be summarized briefly.

In scenarios with many seeders and only very few leechers, our client will download most data from seeders. As the leechers often do not fill up all their upload slots with other leechers,our client is unchoked all the time, yielding a constant download rate.

More interesting are scenarios with a small number of seeders. A fast seeder is able to push data into the swarm at a high rate and all the leechers can reciprocate by sharing the data quickly with their upstreams fully saturated. In this situation, it is difficult for our client to achieve a good downstream: We only get a small share of the seeders' upstream and all the other leechers are busy exchanging pieces between them. Hence, we only profit from the optimistic unchoke slots, which results in a poor performance. However, note that many leechers will turn into seeders relatively soon and therefore our download rate will increase steadily.

A slow seeder is not able to push data fast enough into the swarm, and the leechers reciprocate the newly arrived pieces much faster without filling all their upload slots. Although BitThief cannot profit from the seeders, it can make use of the leechers' free upload slots. The attainable download rate is similar to the one where there are many seeders. The download rate will go down only when BitThief has collected all pieces available in the swarm. When a new piece arrives, the leechers will quickly exchange it, enabling BitThief to download it as well with almost no delay. An experiment illustrating this behavior is given in Figure 4. Note that the execution shown in the figure is quite idealistic, as there are no other leechers joining the torrent over time.

In summary, the results obtained from experiments on the Internet have been confirmed in the experiments conducted in our pet network.

3.4 Exploiting Sharing Communities

Finding the right torrent metafile is not always an easy task. There exist many sites listing thousands of torrents (e.g., Mininova), but often the torrents' files are not the ones mentioned in the title or are of poor quality. Therefore, a lot of *sharing communities* have emerged around BitTorrent. These communities usually require registration on an invitation basis or with a limit on the number of active users. Finding good quality torrents in these communities is much more convenient than on public torrent repositories. Sharing communities usually encourage their users to upload at least as much data as they download, i.e., to keep their sharing ratio above 1. This is achieved by banning users with constantly low sharing ratios or by denying them access to the newest torrents available.

Andrade et al. [2] studied these communities and analyzed how sharing ratio enforcement influences seeding behavior. The authors find that seeders are staying in a torrent for longer periods of time, i.e., typically the majority of peers are seeders. These communities thus exhibit ideal conditions for BitThief, provided that we can find ways to access and stay in this communities without uploading.

We have found that this can often be done by simply pretending to upload. The community sites make use of the tracker announcements which every client performs regularly. In these announcements the client reports the current amount of data downloaded and uploaded. These numbers are stored in a database and used later on to calculate the sharing ratios. The tracker typically does not verify these numbers, although, in our opinion, it would be possible to expose mischievous peers: For instance, in a torrent with 100 seeders and just one leecher, it looks suspicious if the leecher is constantly announcing large amounts of uploaded data. Alternatively, the sum of all reported download and upload amounts could be analyzed over different torrents and time periods, in order to detect and ban dishonest peers.

The tracker can also be cheated easily: Clients can announce bogus information and fake peers so that the tracker's peer list fills up with dozens of clients which do not exist. The seeder and leecher counts reported by the tracker can therefore be misleading as there are usually not that many real peers downloading a given torrent. Even worse, peers asking a tracker for other peers can get a lot of invalid or stale information, which makes torrent starts slow.

An alternative is used by recent BitTorrent clients: A distributed tracker protocol which manages the torrent swarm. The technique of faking tracker announcements has been used in a couple of torrents in our tests and we now have a sharing ratio of 1.4 on *TorrentLeech*[8] without ever uploading a single bit.

An example which emphasizes how dramatic the difference between a community internal and an external

[8] See http://torrentleech.org/.

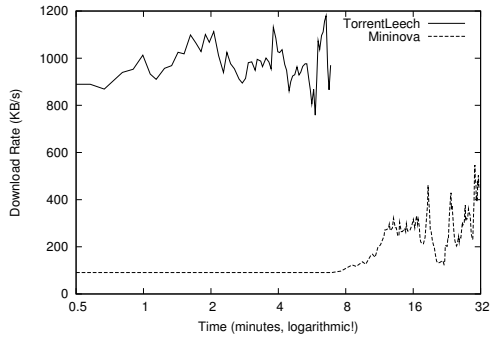

Figure 5: BitThief's download speed: comparison between a community version of a torrent and a torrent of the same file found on Mininova.

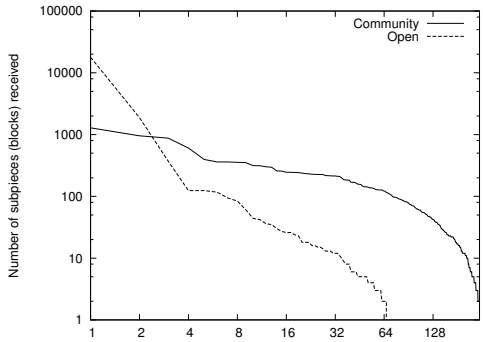

Figure 6: The logarithmically scaled list of peers ordered according to the number of provided blocks is plotted on the x-axis. The file size was 350 MB. On Mininova (*open*) and on TorrentLeech (*community*), BitThief connected to 309 and 349 peers, respectively. In the community network, the distribution is more balanced and BitThief is able to download from much more peers, while only a few peers contributed a large fraction of all blocks in the public torrent.

download can be, is given in Figure 5. We used a torrent that was published on TorrentLeech approximately 12 hours before conducting this experiment and looked for the same one on Mininova, where it had appeared 4 hours earlier. The torrent was 359MB in size on TorrentLeech and slightly smaller (350MB) on Mininova. We first downloaded the torrent three times from Mininova, then three times from TorrentLeech. The Mininova runs took 32/32/37 minutes, while on TorrentLeech the runs completed in 7:25/7:08/7:08 minutes, respectively. This is more than four times faster. Considering that there were only 25 (24 seeders, one leecher) peers in the TorrentLeech swarm and more than 834 (531 seeders, 303 leechers) peers in the other swarm, this is surprising.

As far as the individual contributions of the peers are concerned, we observed the following. While BitThief tends to benefit more from certain peers, generally seeders, in public torrents, a much larger fraction of all peers provides a considerable share of the file in sharing communities, and the distribution across peers is more balanced. This is probably due to the community peers' desire to boost their sharing ratios by uploading as much as possible. An experiment illustrating this point is depicted in Figure 6.

4 Sophisticated Attacks

While simple tricks often yield a good performance, BitTorrent has proved to be quite robust against certain more sophisticated attacks.

First, we have investigated an exploit proposed in [10], which truly violates the BitTorrent protocol: The selfish client announces pieces as being available even if it does not possess them. If such an unavailable piece is requested by a remote peer, the client simply sends random data (garbage). As only the integrity of whole pieces can be checked, the remote peer cannot verify the subpiece's correctness. Note that this behavior cannot be considered free riding in the pure sense, but it is a strategy that does not require to upload any *valid* user data.

In a first implementation, all requests are answered by uploading entire garbage pieces. As has already been pointed out in [10], this approach is harmful: Both the official client and Azureus store information from whom they have received subpieces and will thus immediately ban our IP address once the hash verification fails.[9] Consequently, we have tried to answer all requests for a piece except for one subpiece, which would force the remote peer to get that subpiece from a different peer. The idea is that the remote peer cannot tell which peer uploaded the fake data, as it might as well be the other peer which only supplied one subpiece. While the official client can indeed be fooled this way, Azureus is smarter and uses an interesting approach: Once it has determined that the piece is not valid, it looks up from which peer it received most subpieces. The piece is then reserved for that peer, and Azureus aims at fetching all remaining subpieces from the same peer. When refusing to answer these requests, the connection stalls, and eventually our IP address is banned. We have tried several tricks to circumvent these problems, but came to the conclusion that uploading random garbage, in any way, does not improve performance.

When establishing connections, peers inform each other about their download status by sending a list of pieces that they have already successfully downloaded. While the connection is active, peers send messages to each other for each new piece they downloaded. Therefore, a peer always know the progress of its neighbors. We sought to measure the influence that this information has on a remote peer. Currently, BitThief sends an empty list of available pieces during connection setup and it does not inform the remote peer about any new pieces it acquires. We tried announcing different percentages of all the pieces at the beginning of the connection, but our experiments showed that the performance is independent of the percentage, as long as not all of the pieces are available. However, announcing 100% of the pieces has disastrous consequences, as the remote peer considers BitThief a seeder and therefore does not respond to any piece requests.

[9] Note that an appealing solution would be to fake entire pieces by using contents yielding the same hash values. Unfortunately, however, the computation of such SHA-1 hash collisions is expensive and would yield huge tables which cannot be stored in today's databases.

BitThief profits from the optimistic unchoke slots of leechers and from the round robin unchoke scheme of seeders. Thus, a client could possibly increase the chance of being unchoked by being present in the remote peer's neighborhood more than once. This is known as a *Sybil attack* [5]. However, this attack involves opening two or more connections to a remote peer. Both the official client and Azureus prevent such behavior. If multiple IP addresses are available, it would be an easy task to extend the client in a way to fake two entities and trick remote peers. The peers would gladly open a connection to both external addresses and thus our download rate might increase up to twofold.

5 Related Work

In 2000, Adar and Huberman [1] noticed the existence of a large fraction of free riders in the file sharing network *Gnutella*. The problem of selfish behavior in peer-to-peer systems has been a hot topic in p2p research ever since, e.g. [8, 12], and many mechanisms to encourage cooperation have been proposed, for example in [6, 7, 11, 13, 14].

BitTorrent [4] has incorporated a fairness mechanism from the beginning. Although this mechanism has similarities to the well known *tit-for-tat mechanism* [3], the mechanism employed in BitTorrent distinguishes itself from the classic tit-for-tat mechanism in many respects [9]. This fairness mechanism has also been the subject of active research recently. Based on PlanetLab tests, [9] has argued that BitTorrent lacks appropriate rewards and punishments and therefore peers might be tempted to freeload. The authors further propose a tit-for-tat-oriented mechanism based on the iterated prisoner's dilemma [3] in order to deter peers from freeloading. However, in their work, a peer is already considered a free rider if it contributes considerably less than other peers. We, on the other hand, aim at attaining fast downloads strictly without uploading any data. This is often desirable, since in many countries downloading certain media content is legal whereas uploading is not.

The paper closest to our work is by Liogkas et al. [10]. The authors implement three selfish BitTorrent exploits and evaluate their effectiveness. They come to the conclusion that while peers can sometimes benefit slightly from being selfish, BitTorrent is fairly robust. Our work extends [10] in that, rather than concentrating on individual attacks, we have implemented a client that combines several attacks (an open question in [10]). In contrast to our work, the authors examine the effect of free riders on the *overall system* and argue that the quality of service is not severely affected by the presence of some peers that contribute only marginally. We focus strictly on maximizing the download rate of a *single, selfish peer*, regardless of what effect this peer has on the system.

Finally, [2] has studied the cooperation in *BitTorrent communities*. It has been shown that community-specific policies can boost cooperation. In our work, we have demonstrated that cheating is often easy in communities and selfish behavior even more rewarding.

6 Outlook

In a first thread of future research, we aim at incorporating further selfish attacks such as *collusion* into BitThief. Moreover, current trends such as *ISP caching*[10] could also introduce new potential exploits.

In a second thread of research, we extend our BitThief client such that it truly enforces cooperation among peers. For this purpose, the *Fast Extension*[11] might serve as a promising starting point. A challenging problem which has to be addressed is to find a mechanism that applies some kind of tit-for-tat algorithm for older peers in the system, while at the same time efficiently solving the *bootstrap problem* of newly joining peers: As these new peers inherently do not have any data to share, they must be provided with some "venture capital".

References

[1] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[2] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on Cooperation in BitTorrent Communities. In *Proc. 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2005.

[3] R. Axelrod. The Evolution of Cooperation. *Science*, 211(4489):1390-6, 1981.

[4] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. 1st Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2003.

[5] J. R. Douceur. The Sybil Attack. In *1st International Workshop on Peer-to-Peer Systems (IPTPS), Cambridge, MA, USA*, pages 251–260, 2002.

[6] M. Feldman and J. Chuang. Overcoming Free-Riding Behavior in Peer-to-Peer Systems. *ACM Sigecom Exchanges*, 6, 2005.

[7] D. Grolimund, L. Meisser, S. Schmid, and R. Wattenhofer. Havelaar: A Robust and Efficient Reputation System for Active Peer-to-Peer Systems. In *1st Workshop on the Economics of Networked Systems (NetEcon), Ann Arbor, Michigan, USA*, June 2006.

[8] D. Hughes, G. Coulson, and J. Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, 6(6), 2005.

[9] S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding. In *Proc. 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2005.

[10] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting BitTorrent For Fun (But Not Profit). In *Proc. 5th Itl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[11] S. Sanghavi and B. Hajek. A New Mechanism for the Free-rider Problem. In *Proc. 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2005.

[12] J. Shneidman and D. C. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In *Proc. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[13] K. Tamilmani, V. Pai, and A. Mohr. SWIFT: A System with Incentives for Trading. In *Proc. 2nd Workshop on Economics of Peer-to-Peer Systems*, 2004.

[14] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In *Proc. Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2003.

[10] See CacheLogic Press Release http://www.cachelogic.com /home/pages/news/pr070806.php.

[11] See http://bittorrent.org/fast_extensions.html.