Exploiting BitTorrent For Fun (But Not Profit)

Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang
University of California, Los Angeles
{nikitas, rlnelson, kohler, lixia}@cs.ucla.edu

Abstract

This paper assesses BitTorrent's robustness against selfish peers, who try to download more than their fair share by abusing existing protocol mechanisms. We design and implement three selfish-peer exploits and evaluate their effectiveness on public and private torrents. In practice, BitTorrent appears quite robust against this kind of exploit: selfish peers can sometimes obtain more bandwidth, and honest peers' download rates suffer slightly in consequence, but we observe no considerable degradation of the system's quality of service. We identify private-torrent scenarios in which a selfish peer could benefit more significantly at the expense of honest peers, and discuss the BitTorrent protocol mechanisms that lead to robustness by rendering these scenarios infeasible.

1 Introduction

The popular BitTorrent protocol for large file distribution [4] strives to provide a form of fairness: clients who do not contribute data to the system should not achieve high download throughput. Although BitTorrent's design emphasizes fair interactions to increase performance and scalability [8], the protocol does not strictly enforce fairness. In this paper, we study the effects of *selfish* BitTorrent clients: implementations that attempt to download more than their fair share by abusing protocol mechanisms. We identify three exploits that can potentially deliver increased benefits for a selfish peer, and also cause damage to the honest peers in the community. Our exploits do not exhaust the wide range of possible selfish behavior, but they were derived after careful consideration of the core BitTorrent mechanisms, and we believe they make good representatives of the possible exploits in this space. Experiments with public torrents, and with our own private torrents running on the Planetlab infrastructure [5], show that, in practice, the benefit to a peer employing the exploits is limited, as is the damage to honest peers. We discuss the BitTorrent mechanisms that lead to this robustness, and derive guiding principles for future protocol design based on our results.

The rest of this paper is organized as follows. Section 2 provides a brief description of BitTorrent, as well as related studies concerned with its robustness and performance. Section 3 describes the design and implementation of our exploits, while section 4 evaluates their effectiveness. In section 5 we discuss the results.

2 Background and Related Work

BitTorrent is a peer-to-peer file distribution protocol whose main goal is to alleviate the load on a server hosting popular files. In BitTorrent, a file is divided into multiple pieces, and each piece into multiple sub-pieces. Different pieces of the file can be downloaded from different peers. A *metadata file* is associated with every download. This file contains information necessary for the download process, including the number of pieces and hashes for all the pieces; the hashes are used by peers to verify that a piece has been received correctly. This metadata file is typically created by the content provider, who must also launch at least one client offering the entire file for download. In order to join the download process, a client retrieves the metadata file out of band, usually either from a well-known website or by email. It then contacts the *tracker*, a centralized component that keeps track of all the peers participating in the download. The tracker's IP address and port are found in the metadata file. Nowadays, most torrents are hosted on public trackers that provide their services for free.

When contacted, the tracker responds with a list of randomly selected peers, which might include both *seeds*, who have the entire file already and are offering it to others, and *leechers*, who are still in the process of downloading. The newly arrived peer then starts contacting others on this list, requesting different pieces of the file. Most clients nowadays implement a rarest-first policy for piece requests: they look for the pieces that exist at the smallest number of other peers. This strategy effectively accomplishes the widest dissemination of the rarest pieces in the system, so that the probability of a missing piece is minimized. A peer is able to determine which pieces another peer has based on a bit-field message exchanged upon connecting.

The piece exchange strategy between peers is based on a trading model: peers prefer to send data to peers who reciprocate. In particular, preference is given to those peers that are uploading data at the highest rate. Once in a *choking period*, typically every ten seconds, each peer recalculates the receiving data rate from all the peers in its list and selects the fastest ones, typically three. It then uploads only to those peers for the duration of the period. We say that a peer *unchokes* the fastest uploaders, and *chokes* all the rest. Whenever a peer successfully downloads a new piece, it sends out an advertisement to all others in its list. Furthermore, everyone constantly keeps looking for better connections by randomly unchoking an additional peer once every third choking period, by means of an *optimistic unchoke*. Seeds, who do not need to download any pieces, choose to unchoke the fastest downloaders. Note that this algorithm is considered to be the main driving factor behind BitTorrent's fairness model: a free-rider will eventually get low download rates, since its lack of cooperation will result in being choked from most other peers.

There has been a fair amount of work on the algorithms and performance of the protocol. Bram Cohen, BitTorrent's creator, has described BitTorrent's main mechanisms and their design rationales [8]. Qiu *et al.* derived model-based expressions for the average number of seeds and leechers, as well as the average download time [14]. Bharambe *et al.* use simulations to evaluate BitTorrent's basic operation, and find that the protocol scales very well and that the rarest-first policy outperforms alternative piece picking policies [7]. Several studies have measured BitTorrent traffic in detail. One examines several characteristics of the actual tracker log for the Redhat Linux 9 ISO image, including percentage of clients completing the download, load on the seeds, and geographical spread of clients [11]. Others present detailed measurements of actual torrent traffic, and observe that, although it can efficiently handle large flash crowds, the global tracker could potentially be a bottleneck [12, 13]. Lastly, a recent paper by Guo *et al.* [10] demonstrates that client performance fluctuates widely in small torrents, and that high-bandwidth peers tend to contribute less to the system. Inter-torrent collaboration is proposed as an alternative to providing extra incentives for seeds to stay longer in the torrent.

All the aforementioned studies assume peers conform to the proposed behavior. Shneidman *et al.* [16] briefly mention an exploit similar to the third exploit in this paper, and evaluate its effect on the selfish client's startup download throughput. To the best of our knowledge, no other study has examined the behavior of a BitTorrent system in the presence of peers who abuse protocol mechanisms to gain unfair benefit.

3 Exploit Design and Implementation

We developed three exploits that let selfish clients download more than their fair share, even within the constraints of protocol rules. The exploits focus on the peer interaction protocol, rather than the peer-tracker protocol. In particular, their design is based on selectively contacting other peers and on lying about already downloaded pieces. In order to verify their feasibility and performance implications, we implemented these exploits by modifying an existing BitTorrent client. We chose the latest version (1.3.4) of Ctorrent [2] for its simplicity and ease of extension, and ensured that our changes did not interfere with regular protocol operation.

3.1 Downloading only from seeds

When a new peer joins a torrent, it receives a list of randomly selected peers by the tracker. There is also the option of asking the tracker for a refreshed list at any time. Thus, a selfish client can, upon connecting, repeatedly ask for new lists. Since most trackers perform some form of load balancing, it is reasonable to assume that after a short period of time, such a client will have received the information for most of the seeds in the torrent; these can be easily identified, because they advertise having all pieces of the file. The selfish client can then completely ignore the leechers, and only attempt to connect and download pieces from the seeds. In addition, it can still benefit from optimistic unchoking by accepting pieces from other leechers, yet refuse to upload to them in return. Since seeds are typically high-bandwidth clients, we expect the selfish client to be able to sustain high download rates; this is experimentally confirmed in Section 4. Thus, a selfish client will sooner or later download the entire file, without contributing any data to the system.

This behavior violates BitTorrent's fairness model, according to which free-riding leechers should achieve low rates. It also has the potential of directly harming honest clients. When a fast selfish peer purposefully targets and downloads data from seeds, it occupies one of each seed's unchoking slots. Thus, other, low-bandwidth peers who need pieces available only at the seeds may starve, until either the selfish peer disconnects or the seed selects them though an optimistic unchoke. The damage can be even more significant if a Sybil attack [9] is employed, i.e. if the selfish client impersonates multiple identities and maintains multiple open connections to the same seed.

3.2 Downloading only from the fastest peers

This exploit attempts to maximize the download rate by peering with the fastest peers in the torrent—those who can reciprocate with high rates—*without* performing optimistic unchokes. Finding the fastest peers is not in itself an exploit; BitTorrent tries to do this anyway. However, BitTorrent periodically selects peers uniformly at random through optimistic unchoking. Thus, every client will eventually be given a chance to download from every other client, even if their rates are mismatched. All peers, and especially slow peers, benefit as a result; without optimistic unchoking, slower peers might starve, since they would never communicate with faster peers. Even without optimistic unchokes, the selfish peer stills needs a mechanism to select fast peers to interact with. The protocol dictates that every peer should send out an advertisement when it has finished downloading a new piece. Thus, by observing the frequency of advertisements sent by different peers, a selfish client can roughly infer their download rate. This estimated rate constitutes a lower limit on their download capacity, from which the upload capacity can usually be inferred; our experiments validate this estimation method on private torrents and find it accurate enough to reliably guide the discovery of the fastest peers. A selfish client then attempts to interact only with these peers. A default BitTorrent client might eventually arrive at the same selection; the selfish client attempts to avoid wasting time and resources during convergence. Regarding seeds, there is no way to estimate their capacity, since they do not send out any advertisements, so the exploit opts to always request pieces from seeds if possible.

We expect the benefit from employing this exploit to be less when sharing large files, where the convergence period is negligible compared to the entire download time. It is worth noting that the selfish client is indeed contributing data to the system. What makes this deviation from the rules an exploit is not the different method for estimating the fastest peers, but rather the lack of optimistic unchokes: the selfish peer is actively discriminating against slow peers by refusing to interact with them under any circumstances.

This behavior has the potential of being especially harmful during the startup phase, when new peers can only obtain pieces through optimistic unchokes by others. In addition, other researchers have observed that when all seeds disconnect from the system, fast peers tend to exchange pieces only among themselves, leaving the slow peers without a completed file [3]. This exploit aggravates this scenario by shutting out slow peers during this last phase of a torrent.

3.3 Advertising false pieces

Leechers prefer to upload pieces to those leechers who can reciprocate: those who will upload pieces with high rates in return. In order to attract a given leecher's download bandwidth, a selfish peer must thus offer rare pieces— but these pieces need not be actual pieces of the file. A selfish peer can advertise pieces it does not have; when asked for a sub-piece, it can just send garbage data. The honest receiving leecher will detect the garbage only after receiving all sub-pieces of a piece and checking its hash against the metadata file. Thus, since an entire piece is not necessarily downloaded from a single peer—and even if it is, the protocol does not mandate keeping state about its origin—there is normally no way for detecting which uploaders are lying.

Instead of advertising all pieces at the same time, as proposed in [16], a client employing this exploit advertises new fake pieces at a constant rate. We do not simply advertise all the pieces, since many implementations will not allow seeds to request pieces; instead, we advertise fake pieces at a slow enough rate that most honest peers do not see the selfish client as a seed. Note that even after all the pieces have been advertised, newly arriving leechers will still interact with the selfish peer. This is because the bit-field sent upon connection contains only the pieces that the selfish client actually has. Thus, at any point in time, some honest peers will see the selfish client as a seed, and some will not. Although this exploit punishes the leechers who download garbage, this disruption is not its primary objective; once the selfish peer has a piece, it will gladly share it with the rest.

When making unchoking decisions, a leecher does not consult the list of pieces that are being advertised by the other peers. However, while the unchoking decisions are based solely on rates, the set of peers to consider for unchoking is based on the pieces other peers have. In general, the same idea can be used to exploit any protocol that assigns different value to different pieces of data. By careful manipulation a selfish client could indirectly influence another client's decision-making process to its benefit.

4 EVALUATION

In this section, we describe our experimental setup and present our measurements and results. We evaluate the exploits in turn.

4.1 Experimental setup

For each exploit, we conducted two distinct sets of experiments, one with our own private torrents on the PlanetLab experimental platform [5], and another with public torrents on the wide-area Internet. The specifics of each set are provided in the respective subsections. PlanetLab's convenient tools for collecting measurements from geographically dispersed clients greatly facilitated our experiments. Our private-torrent experiments serve as a means of assessing the exploits' impact, in terms of the selfish client's benefit and the effect on the rest of the community. Since we are controlling all the peers in the torrent, we can record their behavior throughout their lifetime, and can change protocol parameters and observe the resulting effect on all the peers. This in turn helps us identify conditions that improve each exploit's effectiveness and distinguish which protocol mechanisms are responsible for observed behavior.

The results for these experiments are based on 20 runs at different times of day where eight leechers download a single 113MB file in the presence of single seed. Our choice of a small peer population is partly motivated by measurement studies that find that most real torrents tend to be small [10]. PlanetLab's available bandwidth is unusually high for typical torrents; we enforce download and upload limits on the peers by suspending all requests and sub-piece transmissions when a limit is reached, and resuming when the rates fall below the limits. Leechers who complete the download disconnect from the system right away; when employing the exploit, we only measure the download and upload rates during the period that the selfish peer is connected. Leechers join the torrent according to a Poisson distribution with $\lambda = 0.1$. We also ran experiments using other distributions, and our results do not seem to be significantly affected by that parameter. In order to copy and launch the BitTorrent client on the different PlanetLab nodes, we use the *pssh* package [6], which we have modified to allow for executing remote processes at different intervals.

In public torrent experiments, we run two clients, an honest one and a selfish one employing one of the exploits. They both join a given torrent at the same time, and the average download rate the two clients achieve over their entire download lifetime is measured. We ran experiments for torrents with both small and large peer populations, and also at different times of day. These experiments reveal the behavior of the exploits in real settings where diverse protocol implementations participate in piece exchange, and where not all torrents are small.

4.2 Downloading only from seeds

To evaluate the first exploit, we limit the bandwidth of six of the leechers to 3.3Mbps for download and 1.1Mbps for upload. These limits were chosen heuristically in order to showcase the exploit's impact, both to selfish and honest peers. Two of the leechers operate without limits, in order to examine the effect of the exploit when employed by peers with high or low bandwidth. In particular, we evaluate the exploit both for a fast peer located on the same subnet as the seed (FAST), and for a slow peer located overseas from the seed (SLOW). We also impose no limit on the seed.

Figure 1 shows the achieved download rates for four different scenarios: when everybody is honest, when only the slow unlimited peer employs the exploit, when only the fast unlimited peer employs the exploit, and when every leecher

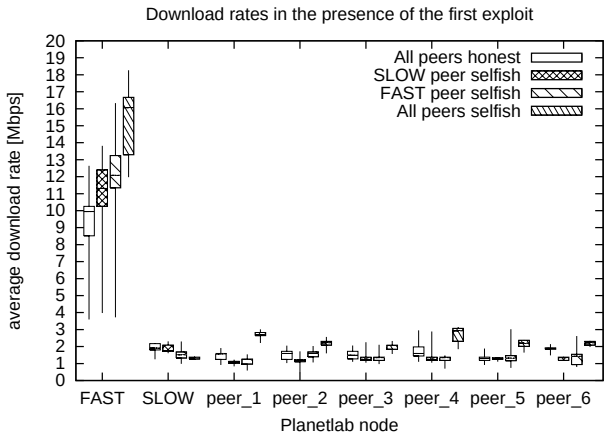

Figure 1: Effect of downloading only from the seed

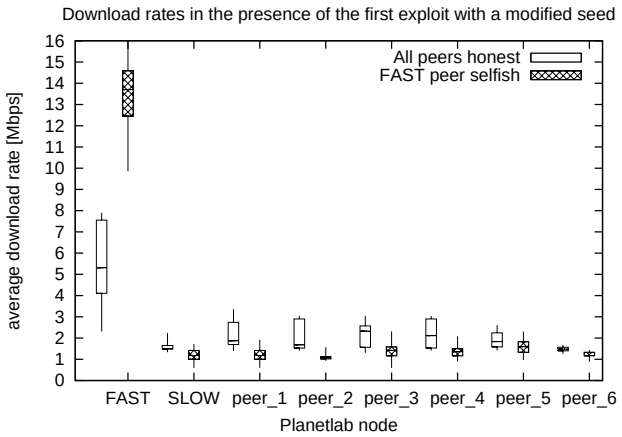

Figure 2: Effect of downloading only from a seed with one unchoking slot

employs the exploit. The first scenario serves as the base case, while the last one seeks to determine the effect of a widespread exploit. The top and bottom of the box for every leecher represent the 75th and 25th percentile download rates over the 20 experiments. The horizontal line inside the box is the median, while the vertical lines extending above and below the boxes represent the maximum and minimum values respectively. Clearly there is high variability, as noted in [10], especially when bandwidth is unlimited.

The maximum benefit is achieved when the selfish peer can maintain a fast connection to the seed. In particular, a selfish fast peer's median download rate improves by 22%, without that peer expending any upload bandwidth. This is because it is able to capture the seed early and is never choked until it completes the download. On the other hand, the exploit is not effective when the selfish peer is slow. The effect on honest peers in both cases is limited, however. Most of them perform only slightly worse, with drops in median download rate ranging from 3 to 46%. Further experiments with lower bandwidth limits validate this claim. In particular, when limiting all leechers to a download bandwidth of 240Kbps, an average speed consistent with earlier findings [13, 10], the selfish peer is not able to sustain high download rates from the seed, and is thus choked, resulting in slightly worse download rates. Therefore, BitTorrent proves to be quite robust against this kind of exploit. We provide an explanation of this later in this section and in Section 5.

Upload rates increase slightly in the presence of the exploit; there is now one less peer contributing data, so there is higher contention for the unchoking slots. Interestingly, it appears that when everybody is cheating, everybody seems to benefit. This counterintuitive result is an artifact of the imposed bandwidth limitations. Since every leecher only downloads from the seed, and since the same limit is imposed on most of the leechers, file sharing degenerates into a fast seed effectively serving the file to multiple equal-bandwidth clients, who take turns in the unchoking slots. In addition, the torrent population is low enough that the seed can easily handle incoming requests. Since this is more efficient than slow leechers exchanging pieces with each other, the download rates of all limited leechers increase. When running the same experiment with all unlimited leechers, we observed that fast peers do slightly better, while slow ones do much worse. Fast selfish leechers will capture the seed, and thus achieve higher rates than they would under normal circumstances, while slow selfish leechers starve with no seeds available to serve them. An improved exploit might download from leechers when it cannot download from the seed and thus prevent starvation for slow selfish leechers.

In order to examine the effect of multiple unchoking slots to the success of the exploit, we ran the same set of experiments, but we limited the number of unchoking slots at the seed to one from the default of three. That is, the seed only sends data to one leecher at a time. We also removed optimistic unchoking at the seed. The new download rates can be seen in Figure 2 for the all-honest and fast-selfish scenarios. Clearly, the exploit is much more effective in this case. The selfish peer's median download rate increases by 155%, while the honest leechers suffer significantly, by at least 32%. This is because the selfish peer effectively monopolizes the seed until it completes the download. Consequently, we claim that file-sharing peer-to-peer protocols that aspire to be robust against such monopolizing exploits should incorporate a mechanism for parallel downloading. We further explore this idea in Section 5.

Experiments with public torrents validate the limited success of this exploit in real settings. Measurements on small torrents with less than 20 peers, and also on large ones with more than 150 peers, show that the selfish leecher gets consistently higher download rates, with median improvements of 7–20%. However, the variability in such torrents is sometimes high, depending on the torrent and the time of day. The exploit does particularly well in torrents with a high number of seeds, since they provide a wider choice of peers for the selfish client. Thus, peers have a greater incentive to be selfish in popular torrents.

In conclusion, the most noteworthy observation about this exploit is not the slight increase in download rate, but rather the fact that a leecher can sustain high rates without necessarily contributing data to the system. This violates BitTorrent's model of ensuring low rates for free-riding leechers.

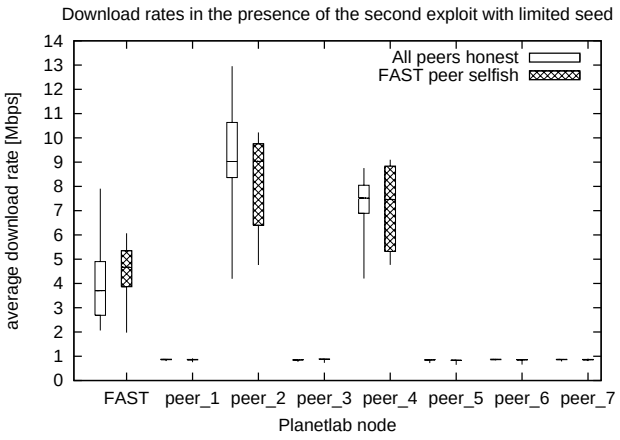

Figure 3: Effect of downloading only from the fastest peers

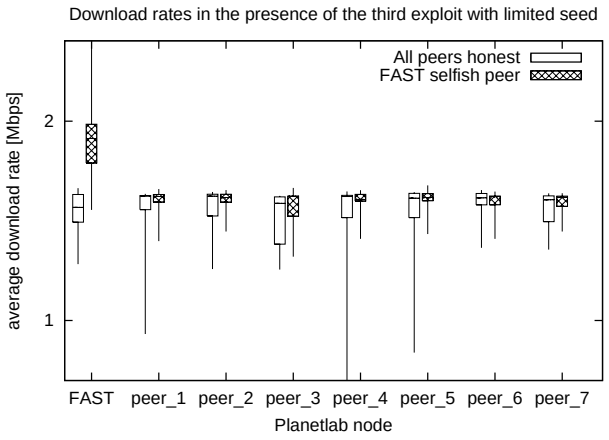

Figure 4: Effect of advertising false pieces

4.3 Downloading only from the fastest peers

For this exploit's evaluation, we limit the upload bandwidth of the seed to 5.73Mbps, and the download and upload bandwidth of five of the leechers to 1.1Mbps and 273Kbps respectively. The seed limit helps us assess the impact in torrents with few seeds; it also makes the potential effects of the exploit more apparent. Figure 3 shows the download rates for two different scenarios: when everybody is honest, and when one fast unlimited leecher employs the exploit by only interacting with the two fastest leechers in its peer list. We observe that the selfish peer achieves 29% better download rates, as measured by the median. This is because it avoids wasting its bandwidth on slow peers and only downloads pieces from unlimited peers 2 and 4, and the seed. The trade-off is a slightly higher upload rate for the selfish leecher, who has to maintain its fast connections to the unlimited peers.

Interestingly, experiments with public torrents do not confirm this success. According to results for torrents with more than 100 peers, the selfish leecher gets consistently lower download rates, by 1–30%. While we cannot draw a definitive conclusion, we believe that the selfish peers' rate estimation algorithm, which works well in PlanetLab's relatively stable environment, is outperformed in the more dynamic global Internet by BitTorrent's short-term rate measurements of peers. Thus, a more adaptive rate estimation algorithm might make this exploit more effective.

4.4 Advertising false pieces

For this exploit's evaluation, the selfish peer advertises 5% of the total number of pieces every five seconds. We limit the upload bandwidth of the seed, as well as the download and upload bandwidth for seven of the leechers, to 1.6Mbps. Our experiments show that higher seed limits reduce the effectiveness of this exploit. Furthermore, we do not limit the selfish client, so that we maximize the probability of honest leechers being willing to exchange data with it. Figure 4 shows the download rates for two different scenarios: when everybody is honest, and when the unlimited leecher employs the exploit. The selfish client achieves 22% better download rates, as measured by the median. The selfish client achieves this improved download rate by remaining unchoked and exchanging more data with the honest leechers. In addition, some of the honest peers also slightly improve their download rates. That is because, once the selfish client gets a piece, it will exchange that piece honestly with the slow peers at a faster rate than other slow peers could.

When describing the design of this exploit, we mentioned that the protocol does not mandate keeping state about the origin of sub-pieces. Increasingly, client implementations, such as the popular Azureus [1], download all sub-pieces from the same peer, and record the information of peers who send out corrupt data in an internal data structure. Thus, they can easily blacklist clients who consistently send out garbage. Thus, this exploit provides little benefit, and is in fact harmful, when interacting with stateful client implementations. When run with an Azureus client, after only four garbage pieces our selfish client was blacklisted for the entire download duration.

5 Discussion

This section discusses a handful of patterns that we believe contribute to BitTorrent's robustness. First of all, the ability to maintain *parallel interactions with diverse peers*, especially when there is *freedom of choice* among peers, greatly facilitates robustness. For example, the first exploit's impact is reduced because seeds have multiple unchoking slots, and because seeds freely invoke optimistic unchoking. Although some exploits have solutions apart from this principle—for instance, seeds employing the so-called "super seeding" policy (by masquerading as leechers and gradually advertising available pieces) could easily thwart the first exploit—maintaining an any-to-any topology enables the protocol to remain resilient in the presence of misbehaving peers. This is evident in the behavior of public torrents in the presence of the third exploit, where honest clients just ignore our selfish peer and continue their download interacting with others.

Detecting and isolating selfish clients requires some *memory of past interactions*: BitTorrent clients that remember the origins of piece downloads are able to detect and punish false piece advertisements. Of course, trade-offs between protocol efficiency and robustness must be considered. Keeping all history of past interactions might severely impact perfor-

mance, since all history would have to be stored and checked on every interaction. Luckily, only a recent subset of history is necessary to determine the trustworthiness of a given peer.

The principle of *problem partitioning* [15] should be strictly enforced. According to that, in a multi-party algorithm, a client should never be able to negatively influence another client's decision process by declaring false information. Right now, BitTorrent does allow a peer to indirectly manipulate another peer's behavior by advertising false pieces. If the data a peer possesses was not used at all when making unchoking decisions—thus decoupling the data needs of a client from the service provided by that client—the third exploit would not be feasible. This could, however, harm performance: in the common case of honest peers, it is indeed advantageous to choose piece-appropriate leechers to interact with.

Selfish peers using our exploits take advantage of information provided by the BitTorrent protocol. It might be possible to enhance robustness by *exporting minimal information*—or, for example, by allowing nodes to *hide their properties* (such as whether they are seeds). Nevertheless, we find, in the failure of the fastest-peer exploit on public torrents, that the network's dynamic properties can make existing information difficult to exploit.

Lastly, BitTorrent's optimistic unchoking policy aids robustness by *preventing monopolization* and *preserving a fully-connected graph*: due to the randomness inherent in optimistic unchoking, every leecher, even the slowest, has a nonzero chance of interacting with a fast leecher or seed. The value of optimistic unchoking is evident in the failure of the fastest-peer exploit on public torrents, as well as the significant success of the first exploit when removing optimistic unchoking at the seed. A more optimized protocol that relied only on rate estimations, though perhaps faster when all peers were honest, would probably be less robust.

6 Conclusion

We have presented three BitTorrent exploits that attempt to abuse existing protocol mechanisms in order to achieve higher download rates. Although in some cases the exploits indeed delivered significant benefits, BitTorrent proved to be quite robust against them. We examined the protocol mechanisms that provide robustness and proposed design guidelines for future peer-to-peer file-sharing protocols.

It would be interesting to investigate combinations of these exploits. For example, the second and third exploits could work well together to better convince the fastest peers to interact with a selfish leecher. In addition, it is not clear what the exploits' effect would be in a multi-torrent system [10] where exchange of pieces belonging to different torrents is possible. The dynamics of cross-torrent communication might also present new opportunities for selfish behavior.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant No. 0230921. We wish to thank RuGang Xu, whose unparalleled spirit provided the initial motivation for this project. We thank the anonymous reviewers, as well as Jeff Shneidman and Lei Guo, who gladly shared with us their insights on BitTorrent systems. We are also grateful to the members of the TERTL research lab for their constructive criticism of our work. Lastly, the first author is indebted to Pino, who provided constant support and encouragement for the entire duration of this project.

References

[1] Azureus homepage. `http://azureus.sourceforge.net`.

[2] Ctorrent homepage. `http://sourceforge.net/projects/ctorrent`.

[3] Jeff Shneidman. Personal Communication.

[4] Official BitTorrent homepage. `http://www.bittorrent.com`.

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

[6] pssh homepage. `http://www.theether.org/pssh/`.

[7] A.R. Bharambe, C. Herley, and V.N. Padmanabhan. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, Redmond, WA, February 2005.

[8] B. Cohen. Incentives build robustness in BitTorrent. In *First Workshop on Economics of Peer-to-Peer Systems*, pages 251–260, Berkeley, CA, May 2003.

[9] J. Douceur. The Sybil attack. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, Boston, MA, March 2002.

[10] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *Internet Measurement Conference (IMC)*, Berkeley, CA, October 2005.

[11] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: five months in a torrent's lifetime. In *Passive and Active Measurements*, Antibes Juan-les-Pins, France, April 2004.

[12] J.A. Pouwelse, P.Garbacki, D.H.J. Epema, and H.J. Sips. A measurement study of the BitTorrent Peer-to-Peer File-Sharing System. Technical Report PDS-2004-003, Delft University of Technology, The Netherlands, April 2004.

[13] J.A. Pouwelse, P.Garbacki, D.H.J. Epema, and H.J. Sips. The BitTorrent P2P file-sharing system: Measurements and Analysis. In *Fourth International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2005.

[14] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM*, September 2004.

[15] J. Shneidman and D.C. Parkes. Specification Faithfulness in Networks with Rational Nodes. In *23rd ACM Symposium on Principles of Distributed Computing (PODC)*, St. John's, Canada, July 2004.

[16] J. Shneidman, D.C. Parkes, and L. Massoulie. Faithfulness in Internet Algorithms. In *ACM SIGCOMM 2004 Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems*, Portland, OR, September 2004.