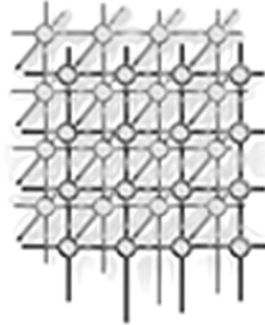


Exploring the Robustness of Unstructured Peer-to-Peer Content Distribution Systems *

Nikitas Liogkas, Robert Nelson,

Eddie Kohler, and Lixia Zhang[†]

University of California, Los Angeles - Los Angeles, CA, U.S.A.



SUMMARY

This paper assesses BitTorrent's robustness against selfish peers who try to download content faster than their fair share by abusing existing protocol mechanisms. We present three exploits that can deliver potential benefits to a selfish peer and evaluate their impact on both public and private download sessions. Our results show that BitTorrent is quite robust against these exploits. Although selfish peers can sometimes attain high download throughput and compliant peers' download rates suffer slightly in consequence, we observe no significant degradation of the overall system's quality of service. We identify scenarios where a selfish peer could attain significant benefits at the expense of compliant peers, and discuss the protocol characteristics that render these scenarios unlikely and hence lead to the system's robustness.

KEY WORDS: BitTorrent exploits free-riding robustness selfish

1. Introduction

The popular unstructured peer-to-peer (P2P) BitTorrent protocol, designed for scalable content distribution, strives to provide a form of fairness: clients who do not contribute data to others should not be able to attain high download throughput. However, although BitTorrent emphasizes fair interactions to increase performance and scalability [7], it does not strictly enforce fairness. In this paper, we study the behavior of *selfish* BitTorrent clients who abuse protocol mechanisms to maximize their download rate while minimizing their own contributions to the system. We identify *three exploits* that can deliver potential benefits to a selfish peer and may negatively impact the download rates of compliant peers. These exploits were derived after careful consideration of the core BitTorrent mechanisms and attack

*A shorter description of this work appeared at IPTPS'06 [13]

[†]E-mails: {nikitas, rnelson, kohler, lixia}@cs.ucla.edu



specific vulnerabilities in the protocol, such as the inability of peers that have the entire content to evaluate other peers' contributions.

Experiments with public torrents—BitTorrent download sessions—and with our own private torrents running on the PlanetLab infrastructure, show that, in practice, *a peer employing the exploits enjoys limited benefits and is unable to degrade the quality of service for compliant peers*. We discuss the BitTorrent characteristics that lead to this robustness and identify the enabling protocol design principles based on our results.

The rest of this paper is organized as follows. Section 2 provides a brief description of BitTorrent and presents related studies concerned with its robustness and sharing incentives. Section 3 describes the design and implementation of the exploits, while Section 4 evaluates their impact in private and public torrents. Lastly, Section 5 discusses the evaluation results and identifies robustness principles that may be also applicable to other unstructured P2P content distribution systems.

2. Background and Related Work

BitTorrent is an unstructured P2P protocol that enables scalable content distribution in the Internet. The basic BitTorrent idea is simple: a client should get better service the more it contributes to the network. This should speed up content distribution, since clients that retrieve data from the origin server will be eager to share that data with others in order to attain higher download throughput.

Prior to distribution, the content is divided into multiple *pieces* and each piece into multiple *blocks*. A *metainfo file* containing all information necessary for the download process is created by the content provider. This information includes the number of pieces, and the SHA-1 hashes for all the pieces that are used by peers to verify received data. In order to join a *torrent*—the collection of peers cooperating to download a particular content—a client retrieves the metainfo file out of band, usually from a well-known website. It then contacts the *tracker*, whose address can be found in the metainfo file, a centralized component that is not involved in data exchange, but rather keeps track of all peers participating in the download. When contacted, the tracker responds with a *peer set* of randomly-selected peers, which might include both *seeds*, peers that have the entire content and are offering it to others, and *leechers* that are still in the process of downloading. The new client then starts contacting peers in this set, requesting different pieces of the content. Most clients nowadays implement a *rarest-first* policy for piece requests: they first ask for the pieces that exist at the smallest number of peers in their peer set. Peers are able to determine which pieces others have based on a *bitfield* message exchanged upon the establishment of new connections, and also on *have* messages sent out by a peer when it downloads and verifies a new piece.

Each peer decides who to exchange data with via the so-called *choking algorithm*, which gives preference to those who upload data to the given peer at the highest rates. Once per *rechoke period*, typically every ten seconds, a leecher reconsiders the receiving data rates from all leechers in its peer set. It then selects the fastest—a fixed number of them—and uploads only to those for the duration of the period. In BitTorrent parlance, a peer unchokes the fastest uploaders and chokes the rest. Furthermore, an additional leecher is randomly unchoked once every third rechoke period, via a so-called *optimistic unchoke*. This allows peers to discover better partners and it enables new leechers that do not have any pieces yet to bootstrap into the system. Seeds, who do not need any download, follow



a different strategy. Most current implementations unchoke those leechers that *download* content from the seed at the highest rates. This aims to better utilize the available seed upload capacity.

Other researchers have also looked into the feasibility of circumventing BitTorrent mechanisms to gain unfair benefit. Shneidman *et al.* [18] were the first to demonstrate that BitTorrent exploits are feasible. They briefly describe an attack to the tracker and a simplified version of the third exploit in this paper. Jun *et al.* [11] argue that the choking algorithm is not sufficient to prevent free-riding and propose a new algorithm to enforce fairness in peers' data exchanges. Piatek *et al.* [16] also observe that high-capacity peers typically provide low-capacity ones with an unfair share of the data. They design a choking algorithm replacement that reallocates the superfluous upload bandwidth to other clients in order to maximize download throughput. Furthermore, since we completed this work, other researchers have extended it in different ways. Locher *et al.* [14] demonstrate that limited free-riding is feasible even in the absence of seeds. They also describe selfish behavior in BitTorrent sharing communities. In addition, Sirivianos *et al.* [19] evaluate an exploit very similar to the first exploit in this paper, and propose a way to prevent it by mandating that each peer maintains a complete view of every other peer.

There have also been some BitTorrent analytical and measurement studies that observed the feasibility of selfish behavior. Qiu *et al.* [17] provide a solution to a fluid model of BitTorrent, where they study the choking algorithm and its effect on performance. They observe that optimistic unchoking may provide a way for selfish peers to free-ride on the system. Massoulié *et al.* [15] introduce a probabilistic model of BitTorrent-like systems, and claim that overall system performance, i.e. the service experienced by compliant peers, does not depend critically on either altruistic peer behavior or the rarest-first piece selection strategy. On the other hand, Bharambe *et al.* [6] utilize a discrete event simulator to evaluate the impact of BitTorrent's core mechanisms, and also observe that the rate-based tit-for-tat strategy is ineffective in preventing unfairness. Izal *et al.* [10] measure several peer characteristics derived from the tracker log for the Redhat Linux 9 ISO image, including the number of active peers, the proportion of seeds and leechers, and the geographical spread of peers. They observe that while there is a correlation between upload and download rates, indicating that the tit-for-tat mechanism is working, the majority of content is contributed by only a few leechers and the seeds. Andrade *et al.* [5] study BitTorrent sharing communities. They find that sharing-ratio enforcement and the use of RSS feeds to advertise new content may improve peer contributions, yet torrents with a large number of seeds present ample opportunity for free-riding. Lastly, in a more recent measurement study [12] we verified the clustering of similar-capacity peers due to the choking algorithm and examined the operation of the sharing incentives mechanism in the presence of an underprovisioned initial seed.

3. Exploit Design and Implementation

The exploits we developed enable selfish clients to download more than their fair share in certain scenarios, even within the constraints of protocol rules. In order to examine their feasibility and impact on BitTorrent systems, we implemented the exploits by modifying an existing BitTorrent client. We chose CTorrent [2] version 1.3.4 for its simplicity and ease of extension. The modified client is available upon request.



3.1. Downloading mainly from seeds

When a new peer joins a torrent, it receives a list of randomly-selected peers from the tracker. However, a peer can always request a new list at any time. There is no mechanism in place for trackers to punish peers that contact them too often. Thus, a selfish client can, upon joining, repeatedly ask for new lists. Such a client will soon acquire the contact information for most of the seeds in the torrent, which can be easily identified as they will be claiming to have all pieces of the content. The selfish leecher can then attempt to *download data mainly from the seeds that do not require any reciprocation*.

In addition, after acquiring a high total number of peers in its peer list, the selfish client has a higher probability of being optimistically unchoked. Thus, it can still benefit by accepting blocks from other leechers *without uploading in return*. Both the aforementioned strategies will help the selfish peer to eventually download the entire content without contributing any data to the system. We show in Section 4 that such a selfish client can sustain high download throughput despite the lack of contributions to others. This behavior directly violates BitTorrent's fairness model, and it also has the potential of harming compliant clients by monopolizing seeds' unchoke slots, thereby slowing down those who need rare pieces available only at the seeds.

3.2. Downloading from the fastest peers

The second exploit attempts to *maximize download throughput by only interacting 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 strives to do this anyway. However, since peers are periodically selected at random for optimistic unchokes, 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 have the chance to interact with faster peers. Consequently, *it is the lack of optimistic unchokes that qualifies this selective behavior as a potential exploit*.

In the absence of optimistic unchokes though, there is a need for another mechanism to discover the fastest peers in the torrent. The fact that peers are supposed to send out a 'have' message when they have downloaded and verified a new piece can be leveraged for that purpose. By observing the frequency of such messages, lower limits on peers' download capacities can be estimated. Note that these estimates may be incorrect, since the protocol in no way mandates that clients send these messages. However, our experiments on private torrents find this estimation method accurate enough to reliably guide the discovery of the fastest peers. Peers' upload capacities can then be inferred from these estimates, as upload and download capacities are usually correlated. Regarding seeds, there is no way to estimate their capacities, as they do not send out any 'have' messages, so the selfish client opts to always request and download pieces from seeds.

Note that an unmodified BitTorrent client might eventually arrive at the same selection of peers. The exploit attempts to avoid wasting time and resources due to convergence and optimistic unchokes. It is also worth noting that, the selfish peer in this case is contributing data to the system. However, the lack of optimistic unchokes can be especially harmful to joining compliant peers, when they can only obtain pieces via optimistic unchokes by others.



3.3. Advertising fake pieces

In order to attract a given leecher, a peer must offer rare pieces for download. A selfish peer can thus attempt to *advertise pieces it does not have*; when asked for a block of data, it can just send garbage. The compliant leechers will only detect the discrepancy after receiving all blocks of a piece and checking its hash against the one in the metainfo 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 peers are sending corrupt data. Instead of advertising all pieces at the same time, as previously proposed [18], a client employing this exploit *advertises new fake pieces at a constant rate*. Advertising all pieces at once is not advisable, since many current implementations will not send data to seeds. Instead, a selfish peer advertises fake pieces at a slow enough rate that most compliant peers never perceive it as a seed.

Although the exploit punishes the compliant peers who download garbage, this is not its primary objective. Once the selfish peer has a piece, it will gladly share it with the rest. Also note that, when making unchoke decisions, a leecher does not consult the pieces that are being advertised by the peers in its peer list. However, while such unchoke decisions are based solely on rates, the set of peers to consider for unchoking is determined in many implementations by the pieces others have. The same idea could be used to exploit any protocol that assigns different value to different pieces of data.

4. Evaluation

4.1. Experimental setup

For each exploit, we conducted two sets of experiments, one with our own private torrents on the PlanetLab experimental platform [4], and another with public torrents in the wide-area Internet. In public-torrent experiments we run two clients, a compliant and a selfish one employing a single exploit at a time, on similar machines with same upload and download capacities. They both join a given torrent at the same time, and the average download throughput the two clients achieve over their entire download lifetime is measured. These experiments reveal the behavior of the exploits in real settings where diverse protocol implementations participate in piece exchange.

Our private-torrent experiments serve to more accurately assess the exploits' impact in terms of both the selfish client's benefit and the effect on the compliant peers. Since we are in control of all the participants, we can modify protocol parameters and observe the resulting effects. This in turn helps us distinguish the protocol mechanisms that are responsible for the observed behavior. The results for the experiments that we present here are based on twenty runs of a setup where eight leechers download a single 113 MB file in the presence of a single initial seed. Our choice of a small peer population is partly motivated by a recent measurement study that found that most real torrents tend to be small [9]. Note, however, that we also evaluate the exploits in large public torrents. PlanetLab's available bandwidth is unusually high for typical BitTorrent clients; we enforce download and upload limits to model more realistic scenarios. Leechers who complete their download disconnect from the system right away, and peer download rates are measured only while the selfish peer is connected to the network to ensure fairness in comparisons. Leechers join the torrent according to a Poisson distribution with $\lambda = 0.1$. We also ran experiments with other peer arrival distributions, and our results do not seem

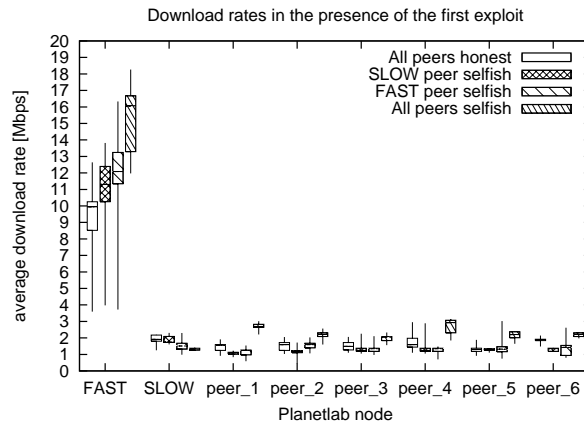


Figure 1. Downloading mainly from the seed

to be significantly affected by that parameter. In order to deploy and launch the BitTorrent client on PlanetLab nodes, we use the *pssh* package [3], which we modified to enable spawning remote processes according to different probability distributions.

4.2. Downloading mainly from seeds

For the first exploit's evaluation, we limit the bandwidth of six of the leechers to 3.3 Mbps for download and 1.1 Mbps for upload, and we impose no limit on the seed. We let two of the leechers operate without limits to examine the exploit's impact when employed by peers with unusually high or low capacity. By doing that we seek to discover the optimal operating conditions for the exploit. On the other hand, examining its applicability in real settings is achieved via the public-torrent experiments. We take measurements for a fast peer located on the same subnet as the seed (FAST) and a slow peer located overseas from the seed (SLOW). Figure 1 shows the observed download rates, represented by a candlestick, for four different scenarios: when everybody is compliant (honest), when only the slow unlimited peer employs the exploit, when only the fast unlimited peer employs the exploit, and when every leecher employs the exploit. The first scenario serves as the base case, while the last one seeks to determine the effect of widespread adoption. The top and bottom of the candlestick box for every leecher represent the 75th and 25th percentile download rates over all experiment runs. The horizontal line inside the box is the median, while the vertical lines extending above and below the boxes represent the maximum and minimum measured values respectively.

Clearly there is high variability, as also noted in [9], especially when bandwidth is unlimited. Note that Piatek *et al.* [16] have shown that this variability is due to the protocol itself rather than to PlanetLab's changing network conditions. The maximum benefit is seen by the selfish peer when it can maintain a fast connection to the single seed. In particular, its median download rate improves by 22%, *without expending any upload bandwidth*. This is because it is able to capture the seed early and

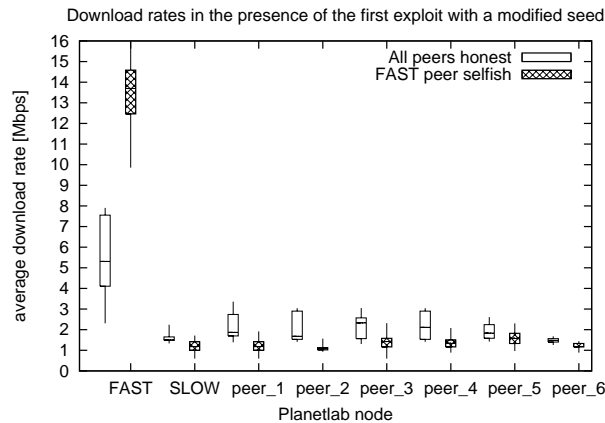


Figure 2. Downloading mainly from a seed with one unchoke slot

is never choked until it completes its download. On the other hand, the exploit is not effective when the selfish peer is slow. However, the impact on compliant peers is limited in both cases. Most of them perform only slightly worse, with drops in median download rates ranging from 3 to 46%.

Interestingly, it appears from the figure that when everybody is using the exploit, all peers benefit. This counterintuitive result is not generally true, but is rather an artifact of the imposed bandwidth limitations. Since all leechers only download from the seed, this scenario degenerates into a fast server providing the content to multiple clients who take turns in downloading data. In addition, the client population is low enough that the seed can easily handle the load. This is clearly more efficient than slow leechers exchanging pieces among themselves. However, when running the same experiment with unlimited selfish leechers we observed that the fast ones do slightly better, while the slow ones do much worse than in the base case. Fast selfish leechers share the seed and achieve high download rates, while slow ones starve with no seed available to serve them.

In order to examine the effect of multiple unchoke slots on the exploit's impact, we ran the same set of experiments *limiting the seed to upload only to one leecher at a time*, without performing any optimistic unchokes. Figure 2 shows the results for the all-compliant 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 compliant leechers suffer significantly, by at least 32%. This is because the selfish peer effectively monopolizes the seed until it completes its download. This leads us to believe that a similar, yet more sophisticated exploit could have severe impact on real torrents with unmodified seeds. The exploit could open multiple connections to seeds, thereby implementing a Sybil attack [8], and attempt to monopolize *all* their unchoke slots. Since most current implementations disallow multiple connections from the same IP address, a selfish peer employing this exploit would have to be reachable through multiple addresses, and keep track of which have been presented to each seed.

Experiments with public torrents also validate the limited success of the first exploit in real settings. Results from both small torrents with less than 20 peers and large ones with more than 150 peers show

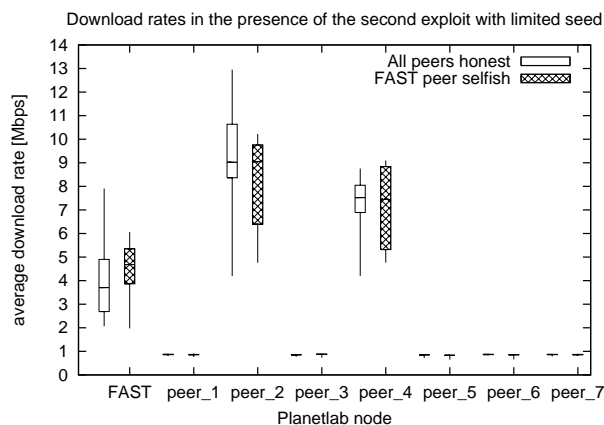


Figure 3. Downloading from the fastest peers

that the selfish leecher attains consistently higher download rates, with median improvements of 7–20%. The exploit does particularly well in torrents with a high number of seeds, since these provide a wider choice for the selfish client. In summary, we observe that a peer can sustain high download throughput without necessarily contributing data to others. This violates BitTorrent’s fairness model: downloading without uploading is indeed feasible to a certain extent in BitTorrent systems.

4.3. Downloading from the fastest peers

For the second exploit’s evaluation, we limit the upload bandwidth of the seed to 5.73 Mbps, and the download and upload bandwidth of five of the leechers to 1.1 Mbps and 273 Kbps respectively. Figure 3 shows the peer download rates for the all-compliant scenario and for a setup where an unlimited selfish peer interacts 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. It avoids wasting its bandwidth on slow peers and downloads most pieces from unlimited peers 2 and 4, and the seed. The trade-off is a slightly higher upload rate for maintaining its fast connections.

Interestingly, *experiments in public torrents do not confirm the success of the exploit*. The selfish leecher’s download rates are consistently lower, by 1–30%. While we cannot draw a definitive conclusion, we believe that the rate estimation algorithm, which works well in PlanetLab’s relatively stable environment, is outperformed in the more dynamic Internet by BitTorrent’s short-term rate measurements. In addition, fast peers close to download completion may have a hard time finding their last remaining pieces, and thus will not send out ‘have’ messages often, thereby appearing to be slow to our selfish peer. It seems that an adaptive rate estimation algorithm is critical to performance in the face of changing network conditions.

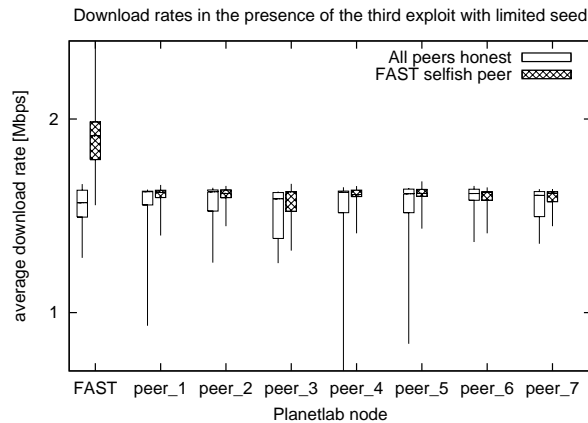


Figure 4. Advertising fake pieces

4.4. Advertising fake pieces

A selfish peer employing this exploit advertises fake pieces at a fixed rate of 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 1.6 Mbps. We do not impose any limits to the selfish client. Figure 4 shows the download rates for the all-compliant and fast-selfish scenarios. The selfish peer achieves 22% better download rates, as measured by the median. In addition, some of the compliant leechers also slightly improve their download rates. This is because, once the selfish client obtains a piece, it will exchange that piece honestly with the slow leechers at a faster rate than they could among themselves.

When describing the design of this exploit, we mentioned that the protocol does not mandate keeping state about the origin of received blocks. However, current implementations increasingly incorporate other defenses. For instance, the popular Azureus [1] first determines the peer from which it has received the most blocks for a corrupt piece. It then attempts to download all blocks of that piece from that peer and does not request any more pieces from it until the particular corrupt piece has been obtained. If the peer in question fails to supply the requested piece soon, it is banned. In that manner, compliant peers can detect and blacklist clients who consistently send out garbage. As a result, *this exploit provides little benefit in public torrents*, and is in fact harmful for the peer employing it when interacting with stateful implementations. When run with an Azureus client, after only four garbage pieces our selfish client was blacklisted for the remaining download. It seems that keeping some state about the origin of received data contributes greatly to robustness, since it enables the punishment of peers who deviate from expected behavior.



5. Discussion

Based on our observations we now identify the system that we believe are the main contributors to BitTorrent's robustness. We argue that the principles we present here are also applicable to other unstructured P2P content distribution protocols that employ multiple-source downloads. This is because they were derived from three representative exploits that attack specific vulnerabilities common to such protocols, such as the inability of seeds to detect free-riders.

First of all, the ability to maintain **many parallel interactions** greatly facilitates robustness. For example, the first exploit's impact is greatly reduced because seeds have multiple unchoke slots, and because they freely invoke optimistic unchoking. Some solutions exist apart from this principle; for instance, seeds employing the so-called *super seeding* policy—by masquerading as leechers and gradually advertising available pieces—could thwart the first exploit. However, maintaining an any-to-any topology enables the protocol to remain resilient in the presence of misbehaving peers. Compliant clients in public torrents just ignore the selfish peer employing the third exploit and continue their download from others.

Detecting and isolating misbehaving clients requires some **memory of past interactions**. BitTorrent clients that remember the origins of piece downloads are able to detect and punish lying peers. Of course, keeping all history of interactions might severely impact performance, since all of it would have to be checked on every interaction. Fortunately, it seems that only a recent subset of history is sufficient to determine the trustworthiness of a peer. Evaluating exactly how much state to maintain is a subject of future work.

The principle of *problem partitioning*, as proposed by Shneidman, should be strictly enforced. That is, **a peer should never be able to influence another peer's decision process by declaring false information**. If the pieces a peer has were not used at all when making unchoke decisions—thus decoupling the data needs of a client from the service provided by that client—the third exploit would not be feasible even in private torrents. This could, however, harm performance: in the common case of compliant peers, it is indeed advantageous to choose piece-appropriate leechers to interact with.

BitTorrent's optimistic unchokes aid 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 this is evident in the failure of the second exploit in public torrents and the significant impact on compliant peers of the first exploit when the seed performs no optimistic unchokes. Consequently, it is important to incorporate a mechanism of **continuous resource discovery** to help peers find the best partners.

Lastly, it should be possible to **hide the properties of a node**, if desired. For example, if a peer could not easily locate the seeds in the torrent, the first exploit would not be possible. A mechanism could be provided to enable a seed to withhold advertising its status to a given leecher unless it has deemed it trustworthy. This would arguably require a reputation system, which could be hosted by the centralized tracker.

6. Conclusion

We have presented three BitTorrent exploits that abuse existing protocol mechanisms in an attempt to download faster than their fair share. Although in some cases the exploits indeed deliver significant



benefits, BitTorrent appears generally robust against them. We examined the protocol characteristics that enable this robustness and identified design principles for unstructured P2P protocols.

Clearly, these exploits do not constitute an exhaustive list in the wide spectrum of selfish peer behavior. However, they were derived after careful consideration of the core protocol mechanisms, and we believe that they constitute good representatives of the space. It would be interesting to formulate a comprehensive methodology that would facilitate the discovery of *all possible selfish exploits* in BitTorrent. In addition, we would like to further investigate *combinations of existing exploits*, as well as the impact of increasing numbers of selfish peers in the system. Lastly, our experiments with a single unchoke slot at the seeds lead us to believe that a *Sybil attack*, where a selfish client maintains multiple open connections to each seed, may be highly destructive.

ACKNOWLEDGEMENTS

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 work. We also thank Patrick Moor, Michael Sirivianos, and Jeff Shneidman, who shared their insights on BitTorrent systems.

REFERENCES

1. Azureus: Java BitTorrent Client. <http://azureus.sourceforge.net>.
2. CTorrent. <http://sourceforge.net/projects/ctorrent>.
3. Parallel versions of openssh tools. <http://www.theether.org/pssh/>.
4. PlanetLab open platform. <http://www.planet-lab.org>.
5. N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on Cooperation in BitTorrent Communities. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'05)*, Philadelphia, PA, August 2005.
6. A. R. Barambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Network's Performance Mechanisms. In *Proc. of Infocom'06*, Barcelona, Spain, April 2006.
7. B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'03)*, Berkeley, CA, June 2003.
8. J. R. Douceur. The Sybil Attack. In *Proc. of IPTPS'02*, Cambridge, MA, March 2002.
9. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *Proc. of IMC'05*, Berkeley, CA, October 2005.
10. M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *Proc. of PAM'04*, Antibes Juan-les-Pins, France, April 2004.
11. S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems (P2PEcon'05)*, Philadelphia, PA, August 2005.
12. A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and Sharing Incentives in BitTorrent Systems. In *Proc. of SIGMETRICS'07*, San Diego, CA, June 2007. To appear.
13. N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting Bittorrent For Fun (But Not Profit). In *Proc. of IPTPS'06*, Santa Barbara, CA, February 2006.
14. T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. of HotNets-V*, Irvine, CA, November 2006.
15. L. Massoulié and M. Vojnovic. Coupon Replication Systems. In *Proc. of SIGMETRICS'05*, Banff, Canada, June 2005.
16. M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proc. of NSDI'07*, Cambridge, MA, April 2007. To appear.
17. D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proc. of SIGCOMM'04*, Portland, Oregon, August 30–September 3, 2004.
18. J. Shneidman, D. Parkes, and L. Massoulié. Faithfulness in Internet Algorithms. In *Proc. of the Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04)*, Portland, OR, September 2004.
19. M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent Networks with the Large View Exploit. In *Proc. of IPTPS'07*, Bellevue, WA, February 2007. To appear.