Improving BitTorrent: A Simple Approach

Alix L.H. Chow
Univ. of Southern California

Leana Golubchik
Univ. of Southern California

Vishal Misra
Columbia University

Abstract

Measurement studies have shown that real world BitTorrent (BT) systems exhibit high seed capacity. Thus, appropriate use of seed capacity can have a significant effect on BT performance. Moreover, seed capacity is also easily exploitable by free-riders, and such free-riding clients exist out in the wild today. In this paper, we propose a simple and scalable approach that makes more intelligent use of seed capacity by hurting free-riders, without their explicit identification, while improving the performance of contributing nodes. The effectiveness of our approach is studied via extensive simulations.

1 Introduction

An important thread in the design and evaluation of P2P systems has been the provision of incentives for nodes to contribute their resources to the P2P system. An interesting aspect of BitTorrent (BT) [4], that makes it stand out in that respect, is the "forcing" of peers to share their resource *while* attempting to complete their downloads, i.e., through the tit-for-tat [4] mechanism – this is done "locally" and without the need for centralized or distributed mechanisms for implementing an incentive system. Although this works nicely, there are still opportunities for *free-riders* (i.e., peers who do not contribute resources) to download files (e.g., [8, 13, 16]). These opportunities exist in two forms: (1) capacity provided by *leechers* (nodes who have not completed their download yet) through the optimistic unchoking mechanism (used by leechers to probe for new downloading opportunities), and (2) capacity provided by *seeds* (nodes that have completed their download but continue to contribute their uploading capacity to the system).

We believe that sufficient evidence exists to indicate that the following two statements are true: (1) opportunities for free-riding hurt the system's performance, and (2) a non-negligible number of nodes in BT stay around as seeds for some time after completing their downloads. That is, many users would only contribute resources if their performance was quite poor. And, the capacity provided by the seeds can significantly contribute to providing reasonable performance for the free-riders. Therefore, we believe that the behavior of seeds is an important factor in facilitating free-riding. And, to the best of our knowledge, there is no other study that explores this.

Moreover, as illustrated later in the paper, a leecher's progress through the file downloading process is not uniform. The progress is slower at the beginning, when having few chunks prevents a node from being selected for uploads through tit-for-tat. It can also be slower at the end, when it is more difficult to find peers with the few remaining pieces of needed data (although provisions exist to aid with that). In both cases, seeds can contribute significantly, as they are no longer playing the tit-for-tat game, and they have all the data pieces. Thus, intuitively, appropriate use of seeding capacity can lead to overall system performance improvements.

Given this, our main idea in this work is to explore alternative approaches for seeds to contribute their upload capacity to the system, with the goal of degrading free-rider performance while (if possible) improving other leechers' performance. The contributions of this paper are as follows:

- We propose several simple approaches to modifying the seeds' uploading algorithms, where the goal is to (a) discourage free-riding behavior and at the same time (b) improve (or at least not degrade) the performance of other leechers. The details of these schemes are given in Section 3.
- We show that simple schemes can provide significantly poorer performance to free-riders while at the same time improving (or not hurting) the other leechers' performance. We do this through a detailed simulation study in Section 4. The poor free-rider performance should serve as a deterrent, which will encourage free-riders to contribute resources.

2 Motivation

In BT, nodes join the system (after receiving "start-up" information from the tracker) and begin requesting chunks of data from their neighbors. Nodes that do not have a complete copy of the file are termed "leechers" and those that do are termed "seeds". Nodes that do not contribute their upload capacities are termed "free-riders". Each leecher i picks a number (typically 5) of nodes to whose requests it will respond with an upload of

an appropriate chunk, i.e., these nodes are "unchoked". A subset of these nodes (typically 4) are picked based on the tit-for-tat mechanism, i.e., those neighbors that have provided the best service (in terms of download rate) to node i recently. And a subset (typically 1) is picked randomly, i.e., they are "optimistically unchoked" (to explore better neighbors). Seeds also pick a subset of neighbors (typically 5), and upload data to them. In past versions of BT, seeds chose neighbors that could download data from them the fastest. In a more recent protocol (as described in [10]) the seeding capacity is distributed more uniformly to the neighboring peers. These choices are re-evaluated periodically.

A number of research studies (e.g., [8, 14, 11, 12, 16]), have focused on the fairness, robustness, and performance characteristics of BT, mainly resulting from the tit-for-tat mechanism. However, very few studies have considered the effects of seeding behavior. Given typical user behavior and the design of most BT clients, seeding is a typical behavior in the BT system. Many users may leave their clients running after completing downloads, possibly due to users not monitoring their clients during the download. Also, some sharing communities enforce a download/upload ratio to enable seeding. Real-world measurements such as (e.g. [1, 3, 6, 7, 10, 15]) also suggest that there exist a significant number of seeds in most torrents; they suggest that there is a large variety of seeding capacity in real-world torrents – some of the torrents measured have twice as many seeds as they do leechers.

Availability of seeding capacity can have a significant effect on BT, e.g., it can compensate for the asymmetric bandwidth scenarios in the Internet. At the same time, it can degrade the fairness and incentive properties of the system, as free-riders can finish their downloads with reasonable performance by relying on the seeds. (Not only do they not contribute to the systems' upload capacity, they also effectively reduce the performance gains that seeds provide.) Thus, intuitively, appropriate use of seeding capacity can have a significant effect on performance of both, contributing leechers as well as free-riders.

Existing studies on seeding behavior (e.g., [2, 9]) mainly concentrate on the initial seeding behavior, such as reducing sending of duplicate data. To the best of our knowledge, no studies focus on how to better utilize the service capacity provided by the seeds. In this paper, we focus on precisely that. Specifically, we ask a simple question – *how can the seeding capacity be better utilized to discourage free-riding behavior and at the same time improve performance of contributing leechers.* We now give a motivating example to illustrate opportunities for "closing a loop-hole" in BT which might encourage free-riding behavior.

Motivating Example

Consider the following example, where we have two classes of nodes and each class is defined by its upload capacity and download capacity. Class 1 nodes are contributing nodes, and Class 2 nodes are free-riders (with bandwidth settings given in Table 1). Figure 1 depicts the download times of each class as a function of average seeding time with 20% of users being free-riders and with no free-riders. These results are obtained by simulation (refer to Section 4) with settings given in Table 2. The 20% free-riders case illustrates that the average download time of free-riders is more sensitive to the seeding time than that of the contributing leechers. For example, with no seeding time, the average download time is 162.33 mins and 632.07 mins for contributing leechers and free-riders, respectively. When the average seeding time is increased to 60 mins, the average download time is reduced to 101.89 mins ($\approx 37\%$ speed up) for contributing leechers and 172.63 mins ($\approx 73\%$ speed up) for free-riders. This suggests that alternative approaches to distributing seeds' upload capacity might lead to significant effects on free-riders' performance while possibly improving (or at least not hurting) the performance of contributing leechers. Moreover, if we degrade free-riders' performance sufficiently, many of them might choose to become contributing leechers, i.e., in order to obtain reasonable performance. This would result in an improved overall system performance – e.g., with an average seeding time of 60 mins (in our example), the average download time can decrease from 101.89 mins to 69.84 mins ($\approx 31\%$ speed up), if free-riders contribute resources similarly to other users.

Table 1: Class Description (example)

Class	Fraction	Download Capacity	Upload Capacity
1 (Contributing)	80%	5000kbps	512kbps
2 (Free-Riders)	20%	5000kbps	0kbps

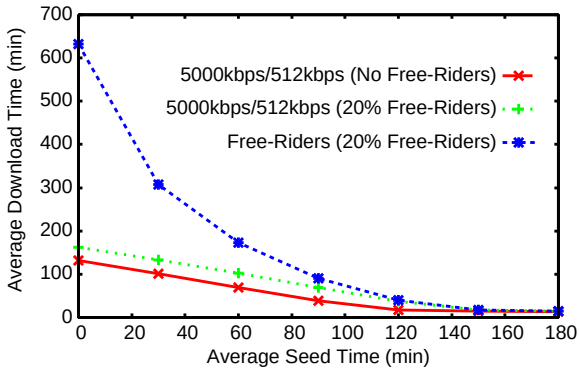

Figure 1: Download Time vs Seed Time (2 Classes)

3 Proposed Approach

We explore the following simple approach to modifying how seeds distribute their upload capacity to leechers.

Currently, seeds unchoke leechers regardless of how far along they are in their downloading process. Measurements and simulations indicate that the downloading rate of leechers is the slowest in two regimes: (i) right at the beginning when they don't have enough chunks to participate effectively in tit-for-tat with other peers, and (ii) right at the end when finding peers with the missing few chunks becomes difficult. (Possible causes of this are explored in Section 5.) In both situations seeds can help since they have no requirement for tit-for-tat and they have all the chunks to supply the last few pieces.

Our simple insight is to *prioritize the use of seeding capacity to only certain portions of a file's downloading process*. For example, we can unchoke only those leechers who are at the beginning (i.e., have a few chunks) and those at the end (i.e., have most of the chunks) of their download process. (Below we describe several approaches based on this idea.)

We can motivate this as follows. Consider a plot of the chunk rate (i.e., the rate at which a node acquires the next chunk) as a function of the number of chunks that node has – this is depicted in Figure 2 which is obtained through simulation (refer to Section 4). Here we plot the the chunk rate behavior of contributing leechers (class 1 in Section 2) and free-riders (class 2 in Section 2), using the original BT seeding scheme as well as our modified scheme in which we prioritize leechers that are towards the two ends of the process. Both, contributing leechers and free-riders experience an increase in seeding capacity at the end points (where we shifted this capacity) and a decrease in the middle portion. However, the free-riders have a far greater dependence on seeds than contributing leechers. Thus, the free-riders get hurt more by being "banished" from using the seeds. The contributing leechers on the other hand benefit by ramping up fast and being able to participate in tit-for-tat earlier than in the original BT scheme. In the example of Figure 2, the average download time of free-riders is degraded by $\approx 66\%$ while contributing leechers experience a small improvement. Armed with this intuition, we now give details of the specific schemes explored in the remainder of this paper. These are just samples of possible approaches. Our goal in this paper is to evaluate the potential of this general approach (refer to Section 4) rather than to exhaustively consider all possible similar schemes. We note first that our approach *does not* require identification of free-riders. And, it is a simple approach to implement and deploy in the current BT system.

Sort-based (N): In sort-based schemes, a seed sorts the requesting neighbors based on the number of chunks each has. It then unchokes N (we use the BT default of $N = 5$) of them based on the sorting order. Motivated by the need to help end points (as explained above), we choose the N that are furthest from the middle (i.e., having half of all the chunks).

Threshold-based (K, N): Threshold-based schemes are similar, except that we choose to unchoke N requesting neighbors that only have $K\%$ of the total number of chunks. Specially, we unchoke nodes that have either $[0..\frac{K}{2}]\%$ or $[(100 - \frac{K}{2})..100\%]$ of the chunks.

A number of variations on this schemes are possible; for clarity of exposition we only focus on the above mentioned one.

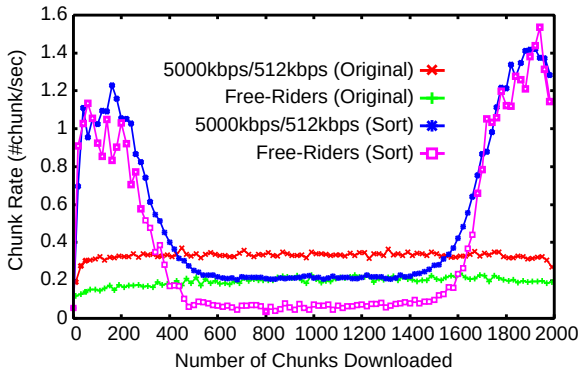

Figure 2: Example Chunk Rate (2 Classes)

4 Simulation-based Study

We use the BT simulator provided by [2], with modifications described below. This is an event-based simulator which simulates the chunk exchanging mechanism in the BT protocol. In [2], the authors use this simulator to study the existing BT protocol behavior, including *initial* seeding issues as well as download rate characteristics under flash crowd conditions (i.e., when all nodes arrive at the beginning of the simulation). They also assume that nodes do not stay around as seeds, once the download completes.

To explore the approach proposed in Section 3, we extended the simulator to support: (a) nodes staying around as seed with different seeding approaches as proposed in Section 3, (b) continuous node arrivals, and (c) nodes acting as free-riders (i.e., nodes that do not unchoke neighbors and leave the system immediately upon download completion). The seeding times and node inter-arrival times follow an exponential distribution. We also modified the original seeding scheme to be more uniform, inline with the current BT protocol (i.e., instead of uploading to the fastest peers as was done in an older version of BT).

We experimented with a number of seeding approaches. The results presented below use the following ones. *Original* refers to the current BT uploading scheme. *Ideal* refers to the scheme where the seeds can

identify which of their neighbors are contributing leechers and which are free-riders, and they only upload to contributing neighbors. We only implement this scheme in the simulator as a baseline for evaluating other approaches (i.e., this gives us an idea of a possible bound on the improvements we can offer). *Sort* refers to the sort-based scheme (see Section 3) where the 5 neighbors furthest away (in either direction) from having half of the chunks are unchoked. *T 20%* (10% to each end) refers to the threshold-based scheme (see Section 3) where the 5 neighbors are chosen for unchoking, using the same algorithm as "original", except that the set of neighbors that quality for an unchoking is restricted to those that have at most the 10% of the chunks or at least 90% of the chunks. *T 40%* (20% to each end) is similar to *T 20%* but with the threshold size being doubled.

Unless specified otherwise, the following results correspond to the settings in Table 2. The system starts with 1 origin seed with 1000kbps upload capacity, which stays in the system for the duration of the simulation. Nodes arrive to the system from a Poisson process with a rate λ, and are assigned to a particular class according to a given distribution. The classes differ in their upload and download capacities. To have a fair comparison between approaches, we use the same node arrival sequence for each simulation with a given arrival rate and class distribution. We look at the steady state behavior of the system. Each simulation run corresponds to 30 hours, and we only compute our results over the last 24 hours. (We check the results to make sure the system passes the "ramp up" stage during the first 6 hours.) We performed a comprehensive study, exploring a variety of parameters. Due to lack of space, we only give representative results; other results are qualitatively similar.

Table 2: Simulation Settings

Filesize	500 MB (2000 Chunks, 256KB each)
Simulation Time	24 hours (+ 6 hours warmup time)
Avg node inter-arrival time ($\frac{1}{\lambda}$)	1 min
Peer Set Size	80
# Leecher Unchokings	4 Regular + 1 Optimistic
# Seed Unchokings	5

Table 3: Class Description (simulation)

Class	Fraction	Download Capacity	Upload Capacity
Slow	40%	1500kbps	128kbps
Fast	40%	5000kbps	512kbps
Free-Riders	20%	1500kbps	0kbps

4.1 Effect of seeding approaches

We consider the effects of different seeding approaches (described above) with class settings given in Table 3. These settings are chosen to be representative of the services provided by commercial broadband ISPs, e.g., they are similar to those provided by local ISPs in Southern California. Figure 3 depicts the download times of different node classes, with several average seeding times, under different seeding schemes. From these results, we observe the following:

- Under "medium" seeding capacities, sort-based scheme do a good job of slowing down free-riders while having a relatively smaller effect on contributing leechers. For instance, when the average seed time is 120 mins, free-riders have an average download time of 263 mins under the original scheme and it is increased to 433 mins ($\approx 65\%$ slow down) by using the Sort scheme. Under the same settings, the download time for the slow class is reduced from 215 mins under the original scheme to 209 mins ($\approx 3\%$ speed up) in the Sort scheme (the fast class is less sensitive to our choice of strategies since it relies on tit-for-tat more than seeds).
- Threshold-based approaches can slow down the free-riders more than sort-based schemes. However, they also have more of a detrimental effect on the contributing leechers – this becomes more critical as the seeding capacity increases since the threshold ends up wasting the seeds capacity. For instance, when the average seed time is 120 mins, the average download time for the free-riders is increased to 683 mins ($\approx 160\%$ slow down) by using the *T 20%* scheme. Under the same settings, the download time for the slow class and the fast class are slowed down by $\approx 19\%$ and $\approx 43\%$, respectively.

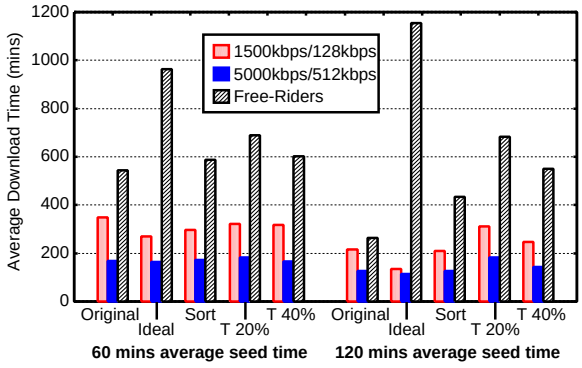

Figure 3: Average Download Time (3 Classes)

4.2 Increasing peer set size (large view exploit)

Interesting studies of free-rider behavior, in the context of the "large view exploit", are given in [12, 16]. The basic idea is for free-riders to increase their peer set size, to increase the probability of being optimistically unchoked by a leecher or picked by a seed's unchoking mechanism. Theoretically, a free-rider can increase the download rate linearly with the increase in the peer set size.

We experiment with adding the large view exploit to our simulations in order to illustrate its effect on our

schemes. For ease of illustration, we present our results using the settings in Table 1 with an average seeding time of 60 mins. We increase the average arrival rate to 2 per minute, thus sufficiently increasing the average number of nodes in the system for the large view exploit to work. Contributing leechers and seeds strive for a peer set of size 80 in all cases, and we vary the peer set size of free-riders. Figure 4 depicts the download rates of contributing leechers and free-riders, as a function of the free-riders "desired" peer set size, and illustrates that:

- The large view exploit does indeed have a significant effect on the original scheme – free-riders can increase their download rate linearly as a function of increasing peer set size. Note that free-riders can even achieve faster downloads than contributing leechers when the peer set size is high enough (in this case 160, 200 or 240).
- Our schemes can still degrade the performance of free-riders and essentially prevent free-riders from increasing their download rates through the large view exploit. We reiterate, the exploit is prevented by our schemes in a completely scalable way since no explicit identification or tracking of free-riders is required.

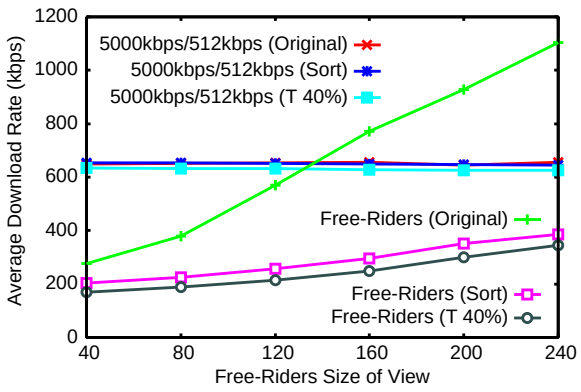

Figure 4: Large View Exploit: Download Rate

5 Discussion

The focus of our work is on better utilization of seeding capacity. The significance of our approache's effects depends on the amount of seeding capacity available. Specifically, with a "medium" amount of seeding capacity (relative to the download capacity), our schemes result in a significant difference, i.e., they degrade the performance of free-riders significantly while either improving or having little effect on the performance of contributing leechers. When the system's seeding capacity is small relative to download capacity, our schemes do not have a significant effect, which makes sense, as seeding behavior is all that is being modified here. In this case, the performance is mainly affected by the utilization of leecher capacity, which is outside the scope of this paper. At the other extreme, when the system's seeding capacity is very high relative to download capacity (either due to long seeding times or to having a large fraction of seeds with high upload capacities), our schemes do not have a significant effect. This also makes sense, as in that case resources are plentiful.

Additionally, our scheme can easily work with other schemes that have been proposed to handle free-riders (e.g., [13, 16]). As these schemes do not focus on approaches to distributing seeding capacity, our schemes can serve as a good complement.

Overall Performance Improvement

Our simulations indicate (refer to Figure 3) that redirecting seeding capacity to end points improves the performance of contributing nodes, especially the slower ones. This confirms our intuition that appropriate prioritization of seeding capacity is useful. We speculate that in a real system our approaches will result in an even greater performance improvement. Consider Figure 2, where nodes using the original protocol have a slower downloading rate at the end points of the downloading process. In a real system, the slow start of newly joined nodes is even worse because it takes awhile to gather the peer list and successfully connect to enough peers, which leads to wasted upload resources of new nodes. Using our schemes the seeds can "kick start" a new node faster (as it rises to the top of their list for unchoking) so that it can start contributing its upload capacity quickly. Since simulation abstractions hide many of the delays, the performance gains of our technique are not visible to the fullest extent; we are exploring such effects through experiments on PlanetLab.

Possible Consequence

Even if our schemes do not improve the performance of the contributing leechers much in some scenarios, they can slow down free-riders significantly in many cases. Consequently, in order to have reasonable performance, free-riders may choose to contribute their resources, thereby increasing the system capacity and improving overall system performance (refer to Section 2).

Malicious Behavior

One way to circumvent our scheme by free riders is for them to lie about the chunks they have so that they would fall in the "right" region from a seeds' perspective. Using current BT clients can act as free-riders easily by limiting their upload rate – most available BT clients allow this. On the other hand, cheating in our scheme requires source code modification, which is harder. Also, modifying source code can result in exploits in the original BT

protocol as well, e.g., as in [11, 14]. Of course, contributing leechers can cheat in a similar manner.

Specifically, one malicious exploit would be for a free-rider to lie about the chunks it has by using a modified client. We believe that it would be tough to get a huge performance gain that way. That is, to prevent a node from always claiming to be within the "right" region, a seed can keep a counter of how many chunks it uploaded to each peer such that they are eventually forced to exit the right region. In this simple scheme, the amount of data a liar can download from a particular seed is no more than the size of the right region.

Of course, a node can lie differently to each seed in order to collect more chunks. However, this requires more sophisticated client modifications. Moreover, a seed can estimate the seeding capacity of a peer by looking at its own peer distribution. A seed could take the view that (roughly and conservatively) it needs to upload at most $\frac{\text{own upload capacity}}{\text{total seed peers capacity}}$ of the "right region" to a peer; this can be used as a reference for the limit on the amount of data it uploads to a peer. Then, ideally, a liar (on average) would download the amount of data corresponding to the "right region" from all its neighboring seeds, which is essentially equivalent to the non-lying case.

We could also consider a simple system that globally keeps track of the total number of chunks uploaded to a peer. Light weight reporting and checking mechanisms can be used, as only loosely synchronized information is enough to serve our purpose.

Lastly, a free-rider can lie by assuming multiple identities in the system, which is known as Sybil attack [5]. In a BT system, a node's IP address can be used as its identity. Thus, a node requires access to multiple IP addresses in order to achieve this exploit. This is fairly difficult to achieve for common users.

Due to space limitations we do not fully explore mechanisms to prevent the exploit by free-riders; however even without any prevention mechanism a cheating node will not be able to prevent the performance improvements to contributing leechers.

Practical Implementation

Our proposed approach can be easily adopted in the current BT system. No protocol changes (in terms of messages) are required as all the information is already available in the current BT protocol, and it also allows for incremental deployment.

6 Conclusions

We considered an important problem of how to appropriately distribute seeding capacity in BT. Our study indicates that, under a number of scenarios, our schemes are able to: (1) significantly degrade free-riders' performance while (2) improving (or not affecting) the performance of contributing leechers. Thus, we believe that this is a promising direction for improving the overall performance of BT while discouraging free-riding behavior. Our ongoing efforts include (a) further improvements to our schemes (as the ideal scheme we used as a baseline indicates that there is still room for improvement under some scenarios), (b) PlanetLab experiments, (c) development of analytical models for studying the effects of free-riding and seeding in BT, and (d) exploration of malicious behavior prevention schemes.

Acknowledgements: This research was funded in part by the USC Annenberg Graduate Fellowship, the NSF 0627590, 0238299, 0615126, 0091474, 0417274, 0540420 grants and by IMSC, an NSF ERC, Cooperative Agreement No. EEC-9529152.

References

[1] A. Bellissimo, B.N. Levine, and P. Shenoy. Exploring the use of bittorrent as the basis for a large trace repository. Technical Report 04-41, CS, UMASS, 2004.

[2] A.R. Bharambe, C. Herley, and V.N. Padmanabhan. Analyzing and improving bittorrent performance. In *INFOCOM'06*.

[3] J. Bieber, M. Kenney, N. Torre, and L.P. Cox. An empirical study of seeders in bittorrent. Technical Report CS-2006-08, Duke University.

[4] B. Cohen. Incentives build robustness in bittorrent. In *P2PEcon'03*.

[5] J.R. Douceur. The sybil attack. In *IPTPS'02*.

[6] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis and modeling of bittorrent-like systems. In *IMC'05*.

[7] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garc'es-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *PAM'04*.

[8] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *P2PEcon'05*.

[9] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *SIGMETRICS'07*.

[10] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *IMC'06*.

[11] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting bittorrent for fun (but not profit). In *IPTPS'06*.

[12] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in bittorrent is cheap. In *HotNets'06*.

[13] T. Locher, S. Schmid, and R. Wattenhofer. Rescuing tit-for-tat with source coding. In *P2P'07*.

[14] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent? In *NSDI'07*.

[15] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *IPTPS'05*.

[16] M. Sirivianos, J.H. Park, R. Chen, and X. Yang. Free-riding in bittorrent networks with the large view exploit. In *IPTPS'07*.