Drawing Crowds and Bit Welfare

EYTAN ADAR
Hewlett-Packard Laboratories

Tit-for-tat style file sharing systems such as BitTorrent have proven to be remarkably effective in dealing with highly popular content. By explicitly addressing free-riding behavior, a "greedy" tit-for-tat approach encourages sharing and succeeds in providing a higher quality of service. However, in situations where a file is not as popular, or the rate of demand is not high, it is frequently difficult to obtain the file in a timely manner. In this paper we demonstrate how additionally greedy behavior on the part of some peers can counterintuitively address this problem. In particular we discuss two possible techniques by which peers, with complete file copies, strategically reduce their effort while improving total network performance by various metrics.

Categories and Subject Descriptors: C.4.2 [**Computer-Communication Networks**]: Distributed Systems – *Distributed applications*
General Terms: Design, Experimentation
Additional Key Words and Phrases: Peer-to-Peer, file distribution, free-riding

1. INTRODUCTION

Barter based file sharing systems, such as BitTorrent [7], work well due to two main factors: a) they enforce uploads, and b) they encourage *swarming* behavior. A tit-for-tat scheme, in which a client's download rate is proportional to their upload rate for a given file, reduces the amount of free-riding [1], and increases availability for currently traded files [9]. This approach works well when a large enough swarm of clients simultaneously attempts to download the file, but means that files that are less popular are typically more difficult to obtain. In part, this is due to low arrival rates for new, interested clients. Fewer clients mean fewer download options as it is unlikely that clients will overlap for a sufficient amount of time to share and replicate data.

If new clients, or *leeches* in BitTorrent terminology, obtain the file from the *seed* — a client with the complete file — there is a chance that they will depart immediately (or shortly thereafter). When the next leech arrives, the seed will have to upload the whole file yet again. Assuming that resources are limited, this scenario is clearly not in the interest of the seed which will have to re-upload a complete copy.

The solutions we consider in this paper attempt to maximize the effectiveness of uploaded data from seeds while not harming, and potentially helping, aggregate network performance. In other words, we would like to maximize the number of copies made of each bit uploaded by a seed. While it may be evident that a seed that limits outgoing data is helping itself, it may not be as clear that it will attain this goal and improve overall system performance. However, our experiments reveal that in various scenarios, the system can gain from selfish seeding behavior.

Author's address: Eytan Adar, HP Laboratories, 1501 Page Mill Road MS 1139, Palo Alto, CA 94304

We believe that whether by choice, or by convention of the application (e.g. a leech automatically becomes a seed), a seed is acting in an unselfish, or altruistic, manner. Perhaps more accurately, a seed switches its preference to file replication over file acquisition. Replication generally conflicts with any desire by the seed to minimize resource consumption. For example, in the current real-world "mainline" implementation of BitTorrent [5] the design favors replication over seed resources. When a leech becomes a seed it will begin to sacrifice resources by uploading as fast as possible to a list of preferred clients.

In this paper we describe a different way to think about the problem in which a "resources for copies" tradeoff is not necessary. We demonstrate this with two simple examples that allow seeds to improve system performance by becoming adaptive agents that more intelligently determine a) whom to upload to and b) at what rate. Specifically:

- Through *bit welfare* a seed will upload to those clients which are the poorest, ensuring that every uploaded piece remains in the system for the longest time possible. This method has been discussed [4], but does not appear to be implemented or analyzed.
- For files with a low rate of arrival of newly interested peers the *draw a crowd* method attempts to artificially build up a swarm of peers by slowing the rate at which leeches can obtain the file based on predictions of future arrivals.

2. RELATED WORK

A number of recent studies have attempted to address P2P performance issues including BitTorrent specifically. We briefly describe a representative sampling of this literature.

In recent work [3], Bharambe and Herley, propose two strategies for BitTorrent including a tit-for-tat scheme based on pieces (rather than rate) as well as an intelligent central tracker that returns a list of peers based on some optimization criteria (rather than at random). Other centralized solutions include [8], where a taxation scheme is suggested and a proximity based criteria described in [15]. While more significant modifications are certainly possible, and may yield better results, we restrict ourselves to changes that do not require third party or centralized intervention.

Modifications to individual client behavior including reputation and incentive models [10] [11], or varying assumptions on cooperation [16] can improve performance by replacing the tit-for-tat strategy. However, these solutions do not explicitly address low-demand files or alleviate the resource demands on clients with complete file copies.

The BitTorrent community has implemented a *super-seed* mode in which seeds pretend to be leeches to encourage uploads of specific file pieces [6]. Finally, future implementations may include the ability to barter pieces across different files [4]. This may be useful, but not necessarily solve the problem at hand. The changes we propose seek to retain a decentralized design with "selfish" peers.

3. BIT WELFARE

To simulate the effects of bit welfare we construct a simplified model for a BitTorrent-like system. Although powerful, other models and simulations [3][14] add unnecessary complexity in answering a fairly straightforward question. Our simulator only makes use

of one seed serving a file composed of *n* pieces. We consider three seed strategies: RANDOM, in which the seed gives a file piece to a peer chosen at random, PREFERRED, in which a peer from a preferred subset is chosen to receive a piece (this is most similar to the current mechanisms), and the bit welfare strategy, POOREST-FIRST, in which the peer with the fewest pieces is chosen. In all cases the seed decides which piece to upload based on a global view of piece allocation. This is consistent with the block choosing protocol described in [3] where a "smartseed" makes a determination of which pieces to upload based on more global information.

The goal of our simulation is to understand the effect of various system properties on the effectiveness of the seed strategies. From our perspective the best strategy will a) minimize the number of times a seed needs to upload a piece while b) increasing the upload-to-download ratio for a peer, and preferably c) keep average download times stable. A system that becomes less dependent on the seed is good both from the seed's perspective, as network resources can be conserved, as well as for other clients that can then work independently of temporary seed failure.

Our simple simulator works by executing the following at each time interval step:

1. *Create a new peer with probability* $p_{arrival}$, *seeds have an upload capacity of one piece per time slice and a variable download capacity,* d_{max}.
2. *If no copy of a piece exists in the network (i.e. the seed is needed) execute the seed strategy*
3. *For each peer, enable optimistic unchoking (give a piece without expectation of return) with some probability,* $p_{optimistic}$.
4. *For each pair of peers, selected in random order, do:*
   a. *If one of the peers is in optimistic unchoking mode, upload the rarest piece needed by the second peer*
   b. *Else, if a trade is possible, copy over the rarest piece needed by each peer, update download counter*
5. *For each pair of peers, selected in random order, upload if an unreciprocated upload is possible*
6. *If all pieces downloaded, remove peer from network*
7. *Randomly remove peers from the network with probability* = p_{churn} + *% done* * (1 – p_{churn})

Note that we do not consider network latency, connection costs, or the effects or complex topologies where peers only have a limited local view of the network rather than a complete global perspective. These may be added in the future, but our goal here is simplicity. Our model for churn is based on the observation that clients are less likely to leave the network the more of a file they have (or are likely to return to complete the download). For each seeding strategy we fix the various probabilities and run the simulation for 100 * *n* iterations. Each simulation was run 50 times for each strategy and variable combination. A total of 162 variable combinations were tried by ranging *n* from 5-15, $p_{arrival}$ from .5-1, $p_{optimistic}$ from .9-1, p_{churn} from .75-1, and d_{max} from 1-2.

We find that in the bulk of the 162 experiments, POOREST-FIRST wins out over the other algorithms. Of the 143 experiments in which there is a significant difference (Kruskall-Wallis, $p < .05$) by POOREST-FIRST over RANDOM, all 143 show an improvement (i.e. decrease in the times a seed is needed to upload pieces during a given run). Similarly, of the 151 experiments with significant differences by POOREST-FIRST over PREFERRED, 151 showed improvement. This is encouraging as it means that

POOREST-FIRST is strictly better or no worse than the other seeding strategies. Figure 1 shows the result for one such experiment.

It is worth briefly considering in which situations POOREST-FIRST does not appear to help. The approach tends to do no better than PREFERRED primarily in runs with larger file sizes, moderate to high churn, and a lower download capacity. What we see is that peers are unable to download enough pieces fast enough. Since churn is proportional to the percent downloaded the seed is "wasting" energy on clients that have a high probability of departing. Interestingly, with a higher download capacity this problem is not nearly as noticeable as clients are able to obtain file pieces from other leeches and are therefore unlikely to depart. In situations where RANDOM performs the same as POOREST-FIRST, we find similar issues (high churn on large files). Additionally, where peers are arriving slowly and have a high download speed the notion of a "poor" peer is rare. Because there are not as many peers in the network and pieces can be downloaded quickly from other peers, the POOREST-FIRST strategy does no better.

In considering average download times, for the 162 experiments the POOREST-FIRST strategy does slightly better than RANDOM (1-2%) and slightly worse in comparison to PREFERRED (again 1-2%). This, again, is encouraging as it implies that we are not causing significant peer delays, making the adoption of this technique more attractive.

Finally, where a statistically significant difference exists between upload to download ratios we again find that POOREST-FIRST does overwhelmingly better. In the 110 of the 114 significant cases between POOREST-FIRST and RANDOM we see this improvement (14% on average). For 124 of the 127 significant cases between POOREST-FIRST and RANDOM we find an average improvement of 13%.

Clearly, there are situations in which POOREST-FIRST does not affect performance in any way. It is also possible that with certain rates of churn and other variables, the other strategies may perform better. For example, some clients are behind firewalls and may always be the "poor." However, these clients will be unable to upload data, so sending them data may not be an effective use of bandwidth. In this case we may want to make the added requirement that seeds should upload to those clients they know are capable of uploading themselves (e.g. biasing themselves to those with the fastest upload rates). This could be called the POOREST-AND-FASTEST-FIRST strategy, but the exact balance of poverty and speed needs to be explored. Our experiment also restricted seed uploads to the times when a piece was completely missing from the network. However, this need not be the final strategy. For an under-loaded seed, with bandwidth to spare, it would help all strategies to upload more than needed. Despite potential issues, we believe that there are many examples in which this seed strategy is useful and demonstrates how a strategy beneficial to the seed is also beneficial to the system at large.

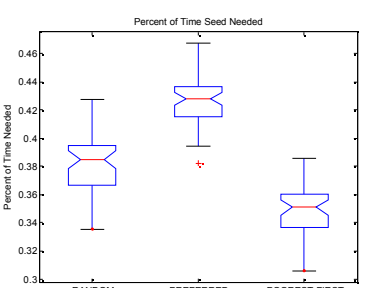

Figure 1: How often a seed is needed based on the three algorithms where n=15, $p_{arrival}$ =.75, $p_{optimistic}$ =.9, p_{churn} =.95, and d_{max} =2.

4. DRAWING A CROWD

After making a "local" decision of who should receive seed-uploaded pieces, we can begin to consider longer-term strategies. Our concern here is the situation when the download swarm is small or slow to arrive and peers are not present simulatneously. One solution is to force peers to overlap by controlling the rate at which they can retrieve data. The effect of this strategy is that we have manipulated the network to boost demand.

To begin to understand the impact of overlapping peers we construct a simple model composed of three peers: one seed, hosting a file of size f, and two downloaders. The seed is always available while the downloaders operate between a start and end range, (t_{s1},t_{e1}) and (t_{s2},t_{e2}) for downloader 1 (D1) and 2 (D2) constrained by $t_{s1} \leq t_{s2}$. Additionally, we define the upload rate of the seed to be to r_s and the (symmetric) upload/download speeds for the two downloaders to be r_{cc} with $r_s \leq r_{cc}$.

In reality, a BitTorrent peer must obtain a complete file piece (e.g. 16 KB to 1 MB) before providing that piece to a neighbor. To simplify the model, our system allows any data, regardless of size, to be immediately available for transfer to a neighbor. In the two downloader case this means that D1 will always receive data at a rate of r_s. This is obviously true before D2 arrives as D1 can make use of the full upload rate of the seed. When D2 arrives, the seed will divide its attention between D1 and D2, interleaving sent data, and uploading to each at $r_s/2$. Recalling that $r_s \leq r_{cc}$ and the data received by the two downloaders is non-overlapping, D2 will also transfer to D1 at a rate of $r_s / 2$. As we will later see, this simplification does not greatly impact the accuracy of the model. The combined download speed to D1 remains r_s, allowing us to conclude that:

$$t_{e1} = t_{s1} + \frac{f}{r_s} \tag{1}$$

Clearly, the optimal scenario for a bit conserving seed is that it should only send out one copy of the file for both peers. Most simply this happens when D1 and D2 start simultaneously (i.e. $t_{s1} = t_{s2}$) and are each given half of the file. The worst case arises when there is no overlap between the clients (i.e. $t_{s2} > t_{e1}$). In this case the seed must give a full copy to each of the downloaders, or transferring $2 * f$ bytes. To understand the situations in which there is partial overlap we fix $t_{s1} = 0$ and set t_{s2} to a constant where $0 \leq t_{s2} < t_{e2}$. Substituting into (1) we find: $t_{e1} = f / r_s$.

D1 will be able to download some amount of data from the seed before the arrival of D2, specifically $(t_{s2} - t_{s1}) * r_s = t_{s2} * r_s$. Again, since $r_s \leq r_{cc}$ some of this data can be sent to D2 after t_{s2}. Exactly how much is constrained by how long D1 will remain online and the amount of "spare" bandwidth between D1 and D2. The spare bandwidth is the quantity of r_{cc} that is not being used to rebroadcast the incoming seed data or $r_{spare} = r_{cc} - r_s / 2$. The time remaining for D1 is $t_{e1} - t_{s2}$. However, since the pre-buffered content at D1 is finite, the time necessary to transfer it may be less than t_{e1}, as we may not be able to make use of r_{spare} for the entire remaining period. We make use of r_{spare} for the minimum of these two time periods:

$$t_{spare} = \min\left(\frac{f}{r_s} - t_{s2}, \frac{r_s * t_{s2}}{r_{cc} - r_s / 2}\right) \tag{2}$$

Since f = t_{spare*} r_{spare} + (t_{s2}- t_{e2})* r_s, we find that t_{e2} :

(3)
$$t_{e2} = t_{s2} + \frac{1}{r_s} * (f - \min(\frac{f}{r_s} - t_{s2}, \frac{r_s * t_{s2}}{r_{cc} - r_s / 2}) * (r_{cc} - \frac{r_2}{2}))$$

Finally, we are able to calculate for the amount of data transferred from the seed, d_{seed} = amount transferred to D1 before t_{s2} + amount transferred to D1 while both D1 and D2 were running + amount transferred to D2 after t_{e1}. Alternatively, and more simply:

(4)
$$\begin{aligned} d_{seed} &= t_{s2} * r_s + (t_{e1} - t_{s2}) * r_s + (t_{e2} - t_{e1}) * r_s \\ &= r_s * t_{e2} \end{aligned}$$

We verify the quality of the model by actually running a real three node BitTorrent network on a fixed length file and varying the starting time for D2. The real file used was approximately 23.84MBs. Using the original Python client [4] we were able to run a seed with r_s set to 130Kbs and r_{cc} set to 500Kbs. The model tracks the actual data very well up (see Figure 2) until approximately 120 seconds. After 190 seconds we experience the effects of larger piece sizes, enforced by the real BitTorrent client, as the peer D1 is unable to transfer the full piece before quitting. However, this is minor linear discrepancy and a simple 20 second adjustment in this region results in a nearly perfect fit (r^2 = .958). We note a similar property to the modeled versus actual runtime for D2 where certain start up and piece packaging costs introduce a minor discrepancy which is corrected by a linear shift (see Figure 3).

Given the current situation our seed has a number of options. The first requires having the two downloaders begin at the same time. A seed can make an announcement that all peers should arrive at a pre-defined time during which it will upload. This is similar to the function of torrent announcements websites. However, to function effectively this strategy would require a change in the protocol, or at the very least in the way announcement sites are used. Such changes introduce a number of problems as they require larger scale coordination and are prone to failure and attack. An alternative

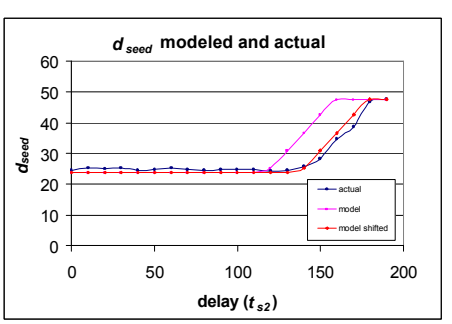

Figure 2: Data transferred from the seed given different starting delays for the second peer. The modeled and actual data are very similar.

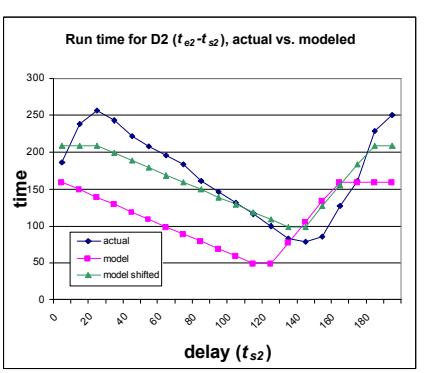

Figure 3: The runtime for the second arrival in the two peer network. Both the model and actual data is displayed.

strategy for the seed is to somehow stretch out the time for D1 to guarantee that D1 will overlap and rebroadcast to D2. Rather than having both clients start at the same time, we are seeking to have them end at the same time.

4.1 Reactive Dampening to Build Demand

One simple way to achieve synchronous end time is to upload to the first client and withhold the last piece of the file. We are in fact aware of this strategy being employed in a streaming video application using a BitTorrent like protocol. However, this assumes that there is only one controlled seed in the system. If multiple seeds exist, a peer may connect to both and lie about which pieces of the file it has.

We prefer a slightly different strategy where we give the seed the ability to vary the bandwidth it will allocate for the transmission of the file. To do this we introduce a dampening factor, d, where $0 < d \leq 1$. We then stipulate that the seed can transmit at either full speed, r_s, or dampened at $d * r_s$. Intuitively, our newly empowered seed will now dampen the connection while there is only one downloader online and switch to full speed when both are online. We can now proceed with an analysis similar to the one above. The trivial, and uninteresting case, is when D2 does not appear in time to overlap with D1. In this scenario we find that $t_{e2} = t_{s2} + t_{e1}$ and $d_{seed} = 2 * f$ where:

$$t_{e1} = t_{s1} + \frac{f}{d * r_s} \tag{5}$$

When an overlap between D1 and D2 exists, we now find that up until t_{s2} D1 will have received $t_{s2} * d * r_s$ (assuming again that $t_{s1} = 0$). The time for D1 to receive the rest of the file at that point will be:

$$\frac{f}{r_s} - t_{s2} * d \tag{6}$$

And the time necessary to send the buffered content given r_{spare} would be:

$$\frac{d * r_s * t_{s2}}{r_{cc} - r_s / 2} \tag{7}$$

Solving for the various variables of interest we find:

$$t_{spare} = \min\left(\frac{f}{r_s} - t_{s2} * d, \frac{d * r_s * t_{s2}}{r_{cc} - r_s / 2}\right) \tag{8}$$

$$\begin{aligned} d_{seed} &= dr_s t_{s2} + (t_{e1} - t_{s2}) r_{rs} + (t_{e2} - t_{e1}) dr_s \\ &= r_s (t_{s2}(d-1) + t_{e1}(1-d) + dt_{e2}) \end{aligned} \tag{9}$$

$$t_{e2} = \frac{f - t_{spare} * (r_{cc} + r_2 / 2) - (t_{e1} - t_{spare} - t_{s2}) * r_s}{d * r_s} + t_{e1} \tag{10}$$

Our seed would like to make sure that the time D1 will remain online after the arrival of D2 is larger than the amount of time necessary to transfer the buffered data and select *d* accordingly. Returning to equation 8, we solve for *d*, s.t.

$$\frac{dr_s t_{s2}}{r_{cc} - r_s/2} \leq \frac{f}{r_s} - dt_{s2} \tag{11}$$

$$d \leq \frac{f * (r_{cc} - r_s/2)}{t_{s2} * r_s * (r_{cc} + r_s/2)} \tag{12}$$

It is important to note that while convenient for the seed, this mechanism may not be practical if *d* is too small. In addition to an increased chance of client failure, a client is driven by a human who may simply tire of waiting. Therefore, a more reasonable approach would be to bound *d* in some way to reduce the chance that the D1 client disconnects from the network. Determining this lower bound requires some experimentation with real users. Another feature lacking in the present model is that downloaders do not always quit as soon as they receive a copy of the file. Rather, these downloaders become seeds themselves for some additional time period. Based on prior experience, a seed may generate a probability distribution on this time and vary *d* accordingly. That is, knowing that the average downloader sticks around, would allow D1 to upload at r_{cc} during this time while downloading nothing and giving D2 an available download bandwidth of $r_s + r_{cc}$.

Though we have constructed this model for two downloaders, it should be apparent that it can be extended to any number of peers. The most significant change now is that D1's upload bandwidth must be used to relay the data from the seed to two different downloaders and share the remaining between D2 and D3. Realistically, there are many additional variables introduced by the larger models as download capacities become more of an issue. Practically, the relaying costs to each downloader will begin to dominate available bandwidth (since each byte received from the seed must be sent to all neighbors). In these larger networks, and in situations where more than one seed is present, it becomes very difficult to model the system analytically.

An alternative strategy is one in which *d* is not calculated, but rather adaptively determined. A seed that has a sense of the arrival rate of new clients can begin to vary *d* as it witnesses the arrival, or lack thereof, of new peers. A seed can be aware of how close a peer is to completion and potentially the rate at which they are getting there. Given this information the entire population of seeds can begin to restrict the downloads if new clients are "expected."

4.2 Estimating Arrival Times

Estimating future arrival times for peers depends largely on the process by which new clients arrive. Assuming, for example, a Poisson process [14] we would only need to look at the average arrival time to calculate λ. Since the arrival rate may change over time, a running window may be used for this estimate. We can also refine the estimate

by calculating lower and upper bounds given some α and calculating the inverse of the χ^2 CDF or normal CDF. Our estimate can be refined over time as new peers arrive.

Recent work [13], however, suggests that the arrival of clients may not be Poisson and different estimation techniques may be appropriate. Further analysis of real world BitTorrent traces [2][12][17] may reveal alternative processes.

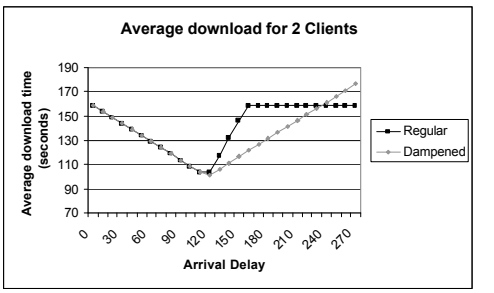

Figure 4: Average download times for 2 downloaders given the regular and dampened strategies for a 23MB file.

4.3 Fairness and the Global Average

Though the dampening approach may seem unfair to the first arriving peer we should consider the effects on the download times in the global average. As an example, we can imagine a seed in North America and two peers from the same network somewhere in Europe. If a file takes 10 minutes to upload from the seed to a European client and the two peers arrive with more that 10 minutes between them, the average download time for both may be 10 minutes. However, imagine that the time to transfer the file between the two peers is 2 minutes. If the seed were to know that the second arrival will be 10 minutes after the first (t_{s1}+10), the seed can dampen the connection to cause the first client to wait 12 minutes for the file. In the 2 minutes of overlap the first peer can upload everything it has received to the second and both will complete at t_{s1}+12 with an average download time of 7 minutes.

In Figure 4 we graphically depict the average download times for the 23.84MB file. The figure was generated by varying the arrival time of the second downloader and solving for d (equation 12). Note that up to 120 seconds the regular (d = 1) and dampened cases generate the same download times. However, between 120 and 230 seconds the average download time is notably different. Beyond this point the cost to the first downloader begins to dominate, causing the average time to exceed the regular curve. Notably, the seed has only uploaded one copy of the file.

Despite the benefit in the region before 230 seconds it is worth considering the cost to the first downloader. When the second client is delayed by under 120 seconds, we see no cost to the first downloader. With delays greater than 120 the first download sees a linear increase, doubling download time as the delay reaches 270. This cost is perhaps not an issue when the downloader order is random as the average time is still largely favorable. However, there is a clear problem if the peers arrive in the same order every time. There are also potential issues if peers begin to game the system to not be the first to arrive. This, however, is an issue even for the system as it exists in BitTorrent networks today. Even an argument of "better overall" may not make a difference with humans involved. As behavioral studies have repeatedly illustrated economic sense does not necessarily translate to human notions of fairness [16] [18]. The reality is that real

world experiments are necessary to test these issues. Most likely some economic "optimality" will need to be tempered with reality.

5. DISCUSSION AND FUTURE WORK

While intuition (and queuing theory) may tell us that that the best way to service demand is to get data out as fast as possible, we have attempted to demonstrate the possibility of alternative techniques based on restricted and controlled data flow. We have shown that in various situations, by counterintuitively addressing the "selfishness" demands of those peers that have complete file copies, it is possible to actually help the network at large.

BitTorrent has grown to be an extremely popular protocol for obtaining content. It is likely to remain this way for a while but despite its popularity there are many areas where it can be improved without drastic changes. We have begun early testing of our methods in an actual BitTorrent client. Our hope is that we may be able to test the utility of these mechanisms in the real world and with more data begin to refine them.

ACKNOWLEDGEMENTS

Thanks to Kevin Lai, Boaz Patt-Shamir, Li Zhang, Fang Wu, Mark Tuttle, Lada Adamic, Mema Roussopoulos, and Bernardo Huberman for their help in fleshing out these ideas.

REFERENCES

[1]. Adar, E. and B. A. Huberman, "Free Riding on Gnutella," First Monday, 5(10), 2000.
[2]. Bellissimo, A., B. N. Levine, and P. Shenoy, "Exploring the Use of BitTorrent as the Basis for a Large Trace Repository," University of Massachusetts Technical Report 04-41, June 2004.
[3]. Bharambe, A.R., and C. Herley, "Analyzing and Improving BitTorrent Performance," Microsoft Technical Report MSR-TR-2005-03, Feb. 2005.
[4]. BitTorrent Mailing List, http://groups.yahoo.com/group/BitTorrent/.
[5]. BitTorrent, The Official Homepage, http://www.bittorrent.com.
[6]. The BitTorrent Specification, available at: http://wiki.theory.org/BitTorrentSpecification.
[7]. Cohen, B., "Incentives Build Robustness in BitTorrent," 1st Workshop on Economics of Peer-to-Peer Systems, Jun. 2003.
[8]. Chu, Y.H., J. Chuang, and H. Zhang, "A Case for Taxation in Peer-to-Peer Streaming Broadcast," ACM SIGCOMM'04 Workshop on Practice and Theory of Incentives in Networked Systems, August 2004.
[9]. Ganesan, P., and M. Seshadri, "On Cooperative Content Distribution and the Price of Barter," Stanford Technical Report 2005-4, Feb. 2005.
[10]. de Veciana, G., and X. Yang, "Fairness, incentives, and performance in peer-to-peer networks," 41st Annual Allerton Conference on Communication, Control and Computing, Monticello, October 2003.
[11]. Feldman, M., K. Lai, I. Stoica, and J. Chuang, "Robust Incentive Techniques for Peer-to-Peer Networks," ACM E-Commerce Conference, May 2004.
[12]. Izal, M., G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime," Passive & Active Measurement Workshop, Aug. 2004.
[13]. Pouwelse, J.A., P. Garbacki, D.H.J. Epema, and H.J. Sips, "The BitTorrent P2P File-Sharing System: Measurements and Analysis" 4th International Workshop on Peer-to-Peer Systems, February 2005.
[14]. Qiu, D., R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," SIGCOMM'04, Aug. 2004.
[15]. Qureshi, A., "Exploring Proximity Based Peer Selection in BitTorrent-like Protocol," MIT 6.824 student project, 2004.
[16]. Raiffa, H., *The Art and Science of Negotiation*, Belknap Press, 1985.
[17]. Sherwood, R., R. Braud, B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," IEEE Infocom, Mar. 2004.
[18]. Thayler, R. H., "Anomalies: The Ultimatum Game," *Journal of Economic Perspectives,* 2(4), pp. 195-206.