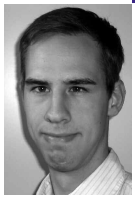
MICHAEL PIATEK, TOMAS ISDAL, TOM ANDERSON, ARVIND KRISHNAMURTHY, AND ARUN VENKATARAMANI

building BitTyrant, a (more) strategic BitTorrent client



Michael Piatek is a graduate student at the University of Washington. After spending his undergraduate years working on differential geometry, his research interests now include incentive design in distributed systems, network measurements, and large-scale systems building.

piatek@cs.washington.edu

Tomas Isdal graduated with a MSc in Computer Science and Engineering from the Royal Institute of Technology, Stockholm, Sweden, and is currently a graduate student in the Department of Computer Science and Engineering at the University of Washington. His interests include peer-to-peer and distributed systems, Internet measurements, and network security.

isdal@cs.washington.edu

Tom Anderson is a Professor in the Department of Computer Science and Engineering at the University of Washington. He is an ACM Fellow and a winner of the ACM SIGOPS Mark Weiser Award, but he is perhaps best known as the author of the Nachos operating system.

tom@cs.washington.edu

Arvind Krishnamurthy is an Assistant Research Professor at the University of Washington. His research interests are primarily at the boundary between the theory and practice of distributed systems. He has worked on automated mechanisms for managing overlay networks and distributed hash tables, network measurements, parallel computing, techniques to make low-latency RAID devices, and distributed storage systems that integrate the numerous ad hoc devices around the home.

arvind@cs.washington.edu

Arun Venkataramani has been an Assistant Professor at the University of Massachusetts Amherst since 2005, after receiving his Ph.D. from the University of Texas at Austin by way of the University of Washington. His research interests are in the practice and theory of networking and distributed systems.

arun@cs.umass.edu

PEER-TO-PEER SYSTEMS OFTEN APPEAL to scalability as a motivating feature. As more users request data, more users contribute resources. Scaling a service by relying on user contributions—the P2P approach—depends on providing incentives for users to make those contributions. Recently, the popular BitTorrent file distribution tool has emerged as the canonical example of an incentive-aware P2P design. Although BitTorrent has been in widespread use for years and has been studied extensively, we find that its incentive strategy is not foolproof. This article describes BitTyrant, a new, strategic BitTorrent client. For users interested in faster downloads, BitTyrant provides a median 70% performance improvement on live Internet swarms. However, BitTyrant also demonstrates that selfish users can improve performance even while reducing upload contribution, circumventing intended incentives.

Bandwidth demands on Internet data providers are increasing. Google Video, Amazon's Unbox, and Apple's iTunes music store are just a few well-known examples of bandwidth-hungry services now delivering entertainment content. In addition, application vendors regularly distribute large patches to thousands of customers. As demand for these services and applications grows, bandwidth costs increase in turn.

Peer-to-peer (P2P) systems offer a promising approach to deferring these costs while increasing scalability. They avoid the bottlenecks associated with typical one-sided data distribution, where servers send data to clients. P2P designs exploit the fact that once a client begins receiving data, it can function as an additional server by redistributing that data, shifting load from the server to clients.

Shifting load from servers to clients means relying on clients to contribute capacity. Early P2P systems builders quickly realized that when given a choice, most users wouldn't contribute their resources. Instead, they would "free-ride," a modern-day tragedy of the commons where users consume system resources without providing any in return [1].

To combat the free-riding problem, subsequent P2P designs included explicit contribution *incen-*

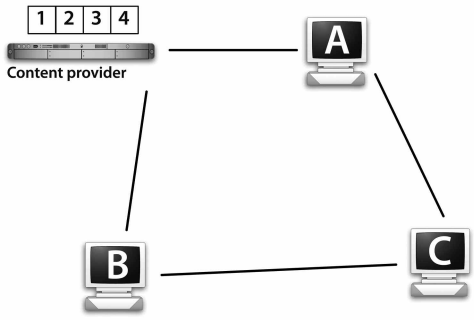


FIGURE 1A: A SAMPLE SWARM TOPOLOGY

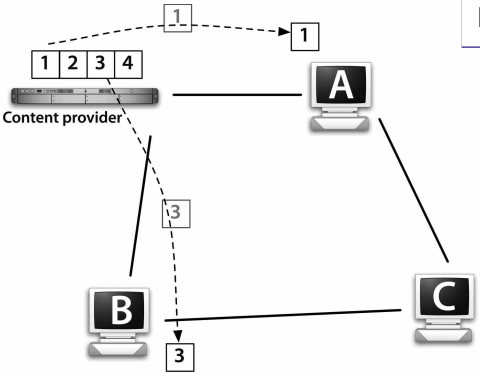


FIGURE 1B: THE PROVIDER SENDS RANDOM BLOCKS TO DIRECTLY CONNECTED PEERS A AND B.

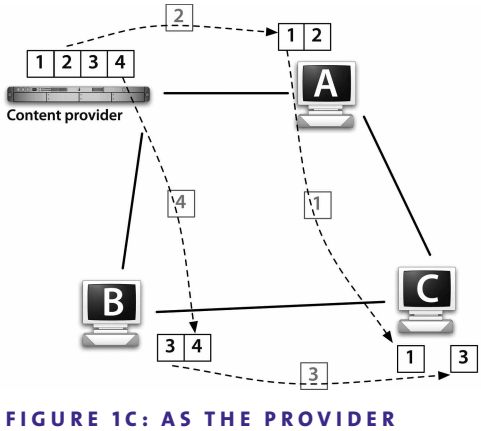


FIGURE 1C: AS THE PROVIDER SENDS MORE NEW BLOCKS, A AND B REDISTRIBUTE PREVIOUSLY RECEIVED DATA TO C, REDUCING LOAD ON THE PROVIDER.

tives. In these systems, increasing contribution improves performance and, as a result, free-riders receive poor service. Today, BitTorrent has become one of the most popular incentive-aware P2P systems and is used daily by millions of people worldwide. BitTorrent's policy is "tit-for-tat"; each individual user gives data to other peers that reciprocate, that is, send data in return.

Intuitively, BitTorrent's tit-for-tat policy makes sense. Each client acts in its local self-interest by rewarding peers that provide it with data. Further, this strategy can be carried out without the need for centralized enforcement, maintaining the decentralized nature of P2P networks. Our question is this: *Did BitTorrent get it right?*

In spite of its success, we find that BitTorrent's incentive strategy can be cheated. We've built a new BitTorrent client, *BitTyrant*, that circumvents intended incentives. Instead of improving performance by increasing contribution, BitTyrant improves performance by operating strategically, enabling its users to improve performance even while reducing their contribution.

How BitTorrent Works

Before describing BitTyrant, we'll first explore how BitTorrent works today. There are three pieces of relevant context: (1) how peers are organized, (2) how data is distributed, and (3) how peers prioritize requests. We examine these in the context of an example file distribution.

A BitTorrent user obtains a file by first joining a *swarm*, a set of peers already downloading the file. To join a swarm, clients contact a centralized coordinator, which returns a random subset of the existing peers to the new client. These peers form the new client's local neighborhood—the set of directly connected peers from which the client will send and receive file data. A sample swarm topology is shown in Figure 1a. In this example, three peers (A, B, and C) have just joined the swarm.

BitTorrent distributes a file by splitting it up into several fixed-size blocks. In Figure 1a, the content provider has a complete copy of the file, which it has split into four blocks. In practice, BitTorrent blocks are small, and a large file might be split into thousands of blocks, but we limit ourselves to four for simplicity. Content providers distribute data by sending randomly chosen blocks to directly connected peers. In Figure 1b, the content provider is directly connected to A and B, which receive blocks 1 and 3, respectively. After receiving these blocks, A and B can begin redistributing data to their directly connected peers—in this case, C. Figure 1c shows the next round in this process: the content provider continues to send new blocks to A and B while they concurrently redistribute previously received blocks 1 and 3 to C. This process continues until peers obtain all blocks and have a complete copy of the file.

In our example, each client receives only a few blocks from a few peers. In practice, clients are connected to dozens of peers that compete for scarce upload bandwidth. BitTorrent clients are faced with a decision: Given many competing requests and limited resources, which should be serviced? BitTorrent adopts a tit-for-tat strategy. First, a client ranks peers according to the rate at which they have been sending data in the recent past. Then the client provides the top k of these peers with an equal split of its upload capacity. The value of k is fixed and determined by a peer's upload capacity. In our example, suppose peer C has total upload capacity 20 with $k = 1$ and receives data from A and B at rates 15 and 10, respectively. In this case, C would reciprocate with A, providing it with data at rate 20.

Decisions about which peers receive data are reevaluated every 10 seconds, a tit-for-tat round. Each round, clients send data to a few random peers that have not "earned" it in a tit-for-tat sense, to explore their local neighbors for better pairings and to bootstrap new users into the tit-for-tat process. Once a new peer has received a few blocks, it can begin trading to induce reciprocation.

Building BitTyrant

Although BitTorrent's tit-for-tat strategy rewards contribution, the reward is not exact. A client that contributes quickly *tends* to receive quickly—with some variability. For instance, a DSL user might be directly connected to a peer behind a university's high-capacity link. Although the DSL user might send at rate 10, the university peer might reciprocate at rate 100. Mismatches like this arise because peers make decisions with limited information.

Ideally, tit-for-tat would match high-capacity peers with mostly high-capacity peers and low-capacity peers with mostly low-capacity peers, avoiding unfair mismatches. In practice, achieving this grouping is slow. Recall that BitTorrent clients search for better matches randomly. Because the bulk of BitTorrent users are low-capacity, high-capacity peers encounter one another comparatively infrequently, through random exploration. Furthermore, BitTorrent swarms are highly dynamic, with users arriving and departing rapidly. Even if a stable pairing arises, it may be short-lived.

Capacity mismatches among peers suggest a potential strategy for improving performance. If a client could quickly identify high-capacity peers, it might induce reciprocation even with small contributions, relying on the slow convergence of tit-for-tat to inhibit competition. Predicting the effectiveness of this strategy depends on how often mismatched pairings occur and on the extent of the imbalance in mismatches.

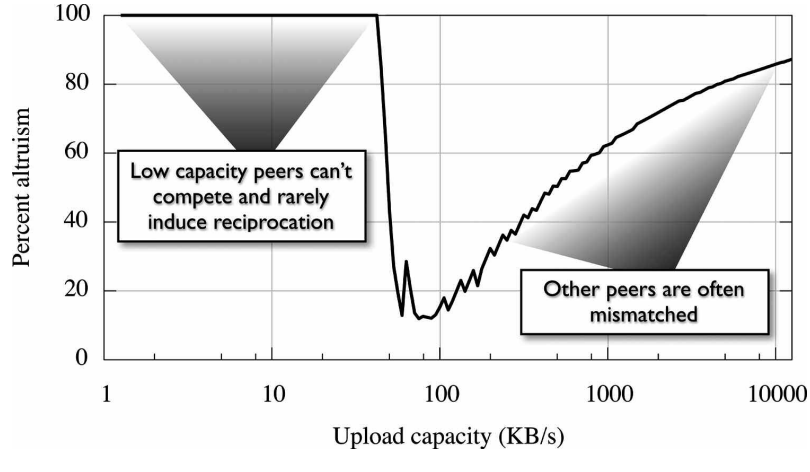


FIGURE 2: PERCENTAGE OF ALTRUISTIC UPLOAD CONTRIBUTION AS A FUNCTION OF CAPACITY

We examine the impact of mismatches through the lens of *altruism*, the contributions of a user that, if withdrawn, would not impact performance. For instance, if a high-capacity user sends data to a peer at rate 100 when rate 10 would suffice to induce reciprocation, we say that 90% of that contribution is altruistic. The altruistic proportion of a random BitTorrent connection can be computed statistically, and Figure 2 shows percent altruism across all connections as a function of upload capacity. Altruism is highest at the ends of the capacity spectrum. The lowest-capacity peers rarely induce reciproca-

tion, because their contribution rates are not competitive. These peers rely on the random exploration of others for all the data they receive. As a result, virtually all of their contributions could be withdrawn. At the high end, mismatches are frequent, with altruistic contributions increasing with capacity. For those peers in the middle, altruism varies, but it never reaches zero.

All peers make altruistic contributions, suggesting that a strategic client could improve performance by identifying those contributions and reallocating them intelligently. This is the approach taken by BitTyrant, a more strategic BitTorrent client that exploits altruism. BitTyrant deviates from the behavior of most existing BitTorrent clients in two ways. First, rather than exploring peer pairings randomly, BitTyrant infers which peers have high capacity and preferentially explores them. Second, BitTyrant does not split its upload capacity equally; instead, it dynamically varies the send rate to each peer to maximize return on investment.

To infer which peers have high upload capacity, BitTyrant relies on control traffic broadcast by all BitTorrent peers about which data blocks they have received so far. Measuring the rate of these block announcements provides an estimate of the download rate of a peer. Recall that tit-for-tat, although inexact, tends to reward higher contribution with a higher download rate, allowing a BitTyrant peer to infer that a peer downloading quickly may also upload quickly.

Although the download rate heuristic provides a good first approximation, it might not be accurate, and network conditions change over time. Further, simply picking *which* peers will receive data is only half the problem; BitTyrant also needs to choose *how quickly* to send to each of those peers. BitTyrant copes with both of these issues by dynamically selecting peers and rates at the same time. For each peer, BitTyrant maintains a benefit/cost ratio, where cost is the upload rate required to induce reciprocation, and benefit is the download rate resulting from that reciprocation.

BitTyrant sorts peers by their benefit/cost ratios, sending data to each in descending order until upload capacity is exhausted. After every tit-for-tat round, BitTyrant updates its estimates of cost and benefit according to peer behavior. A peer that reciprocates has its download rate (benefit) updated with the directly observed download rate. If after receiving data a peer does not reciprocate, BitTyrant increases the cost estimate (required upload rate). Finally, BitTyrant interprets continued reciprocation over many tit-for-tat rounds as a signal that its cost estimate is too generous and scales down the upload rate provided to the continually reciprocating peer.

These rules reflect the strategic nature of BitTyrant's approach. Motivated by the heavy skew of bandwidth capacity, BitTyrant actively seeks out the minority of high-capacity peers that provide the bulk of download throughput. To ensure continued reciprocation with these peers and maximize overall return on bandwidth investment, BitTyrant dynamically adjusts which peers receive data and sends rates to those peers.

Performance in the Wild

To evaluate BitTyrant, we measure its download performance in live Internet swarms. To provide an apples-to-apples comparison, we compare BitTyrant to Azureus, currently the most popular BitTorrent client implementation and the distribution on which BitTyrant is based. We crawled popular BitTorrent swarm aggregation Web sites, obtaining a set of 114 swarms, which we then downloaded concurrently with both BitTyrant and Azureus from two machines at the University of Washington. Both clients were given an

upload capacity limit of 128 kilobytes per second, to avoid interference from network cross-talk and to provide an evaluation of BitTyrant's effectiveness for modestly provisioned hosts.

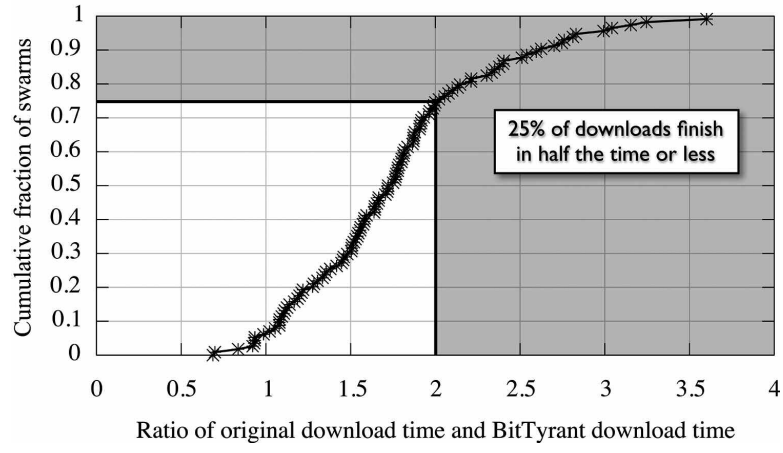


FIGURE 3: DOWNLOAD PERFORMANCE FOR 114 REAL-WORLD SWARMS, SHOWING THE RATIOS BETWEEN DOWNLOAD TIMES FOR AN EXISTING AZUREUS CLIENT AND BITTYRANT

For each swarm, we compute the ratio of Azureus's download time and BitTyrant's download time. For example, if Azureus downloads a file in 30 minutes and BitTyrant completes in 15, this ratio is 30/15 = 2. These completion time ratios are summarized in Figure 3, which gives the fraction of swarms (*y* axis) with a ratio of a particular value (*x* axis) or less. For example, at ratio 2, the function takes the value 0.75, meaning that 25% of BitTyrant downloads finish in half the time of Azureus or less. Depicted this way, every point to the right of ratio 1.0 represents a performance improvement. For the vast majority of live Internet swarms, BitTyrant's strategic behavior improves performance.

Although these results demonstrate the significant performance benefits BitTyrant can realize today, the long-term outcome of strategic behavior on aggregate BitTorrent performance is unclear. Our paper [2] provides further experiments comparing BitTyrant and BitTorrent behavior, in particular examining the performance outcome if all users adopt BitTyrant. The performance for all BitTorrent users today depends on the altruistic contributions that all peers make. BitTyrant improves performance by identifying these altruistic contributions and reallocating them if possible. Because the bandwidth capacity of Internet end hosts is so skewed, high-capacity and even moderate-capacity peers tend to have such a disproportionate share of total resources that even after BitTyrant allocates all available bandwidth strategically, excess capacity remains. BitTyrant presents these users with a choice. If they continue to contribute all available capacity even after identifying the altruistic portion, overall performance improves. Alternatively, overall performance degrades if altruistic contributions are withheld. Ultimately, whether or not incentives stronger than BitTorrent's tit-for-tat are needed in future P2P systems will be determined by user behavior.

The BitTyrant client implementation we developed during the course of our work is publicly available for Windows, Mac OS X, and Linux at http://BitTyrant.cs.washington.edu and has received hundreds of thousands of downloads to date. Full source code for all platforms is also available.

FUNDING ACKNOWLEDGMENT

This work was supported by NSF CNS-0519696 and the ARCS Foundation.

REFERENCES

[1] Eytan Adar and Bernardo A. Huberman, "Free Riding on Gnutella," *First Monday* (October 2000).

[2] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani, "Do Incentives Build Robustness in BitTorrent?" *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '07),* 2007.