BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives

Dave Levin
University of Maryland
dml@cs.umd.edu

Katrina LaCurts
University of Maryland
katrina@cs.umd.edu

Neil Spring
University of Maryland
nspring@cs.umd.edu

Bobby Bhattacharjee
University of Maryland
bobby@cs.umd.edu

ABSTRACT

Incentives play a crucial role in BitTorrent, motivating users to upload to others to achieve fast download times for all peers. Though long believed to be robust to strategic manipulation, recent work has empirically shown that BitTorrent does not provide its users incentive to follow the protocol. We propose an auction-based model to study and improve upon BitTorrent's incentives. The insight behind our model is that BitTorrent uses, not tit-for-tat as widely believed, but an auction to decide which peers to serve. Our model not only captures known, performance-improving strategies, it shapes our thinking toward new, effective strategies. For example, our analysis demonstrates, counter-intuitively, that BitTorrent peers have incentive to intelligently under-report what pieces of the file they have to their neighbors. We implement and evaluate a modification to BitTorrent in which peers reward one another with proportional shares of bandwidth. Within our game-theoretic model, we prove that a proportional-share client is strategy-proof. With experiments on PlanetLab, a local cluster, and live downloads, we show that a proportional-share unchoker yields faster downloads against BitTorrent and BitTyrant clients, and that under-reporting pieces yields prolonged neighbor interest.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; H.1.0 [**Information Systems**]: Models and Principles—*General*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative computing*; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*

General Terms

Algorithms, Design, Economics, Theory

Keywords

Incentive systems, BitTorrent, tit-for-tat, auctions, proportional share

SIGCOMM'08, August 17–22, 2008, Seattle, Washington, USA.

1. INTRODUCTION

BitTorrent [2] is a remarkably successful decentralized system that allows many users to download a file from an otherwise under-provisioned server. To ensure *quick* download times and *scalability*, BitTorrent relies upon those downloading a file to cooperatively trade portions, or *pieces*, of the file with one another [5]. Incentives play an inherently crucial role in such a system; users generally wish to download their files as quickly as possible, and since BitTorrent is decentralized, there is no global "BitTorrent police" to govern their actions. Users are therefore free to attempt to strategically manipulate others into helping them download faster. The role of incentives in BitTorrent is to motivate users to contribute their resources to others so as to achieve desirable *global* system properties—fast download times for all peers, resilience to failures, and so on—without having to resort to a centralized coordinator.

Though long believed to be robust to strategic manipulation, BitTorrent's incentives have recently come under scrutiny. Clients such as BitThief [17] and BitTyrant [22] empirically demonstrate that users do *not* currently have incentive to follow the BitTorrent protocol. These results are a surprising contrast to the general understanding that BitTorrent uses a tit-for-tat-like mechanism. We address this apparent contradiction by considering the following natural questions: Can we rigorously show *why* BitTorrent is vulnerable to strategic manipulation? Can we develop *new incentive mechanisms* to make BitTorrent robust to a wide range of selfish gaming while retaining, or even *improving*, its performance?

Analyzing BitTorrent's incentives

We find that these questions have, to date, gone unanswered in large part due to a basic misinterpretation of BitTorrent's incentives. BitTorrent is widely understood to use tit-for-tat [5, 13, 18, 19, 23, 24].[1] The first broad contribution of our work is in developing a game theoretic model of BitTorrent's incentives that shows that *BitTorrent does not use tit-for-tat*. We find that *an auction-based model is more accurate*. The difference is subtle, as BitTorrent does share some of the same properties of tit-for-tat. But we show that re-framing the existing mechanism as an auction is powerful in that (1) it captures the known, performance-improving strategies of BitThief and BitTyrant, (2) it reveals *new* means of strategically manipulating BitTorrent, and (3) it provides insight into the design of a more robust incentive mechanism.

The studies of BitTorrent's incentives have focused predominately on how to game the *unchoking algorithm* which peers use to determine how much to upload to others. Our auction-based model captures this algorithm, but also reveals another area for strategic manipulation: the *piece revelation strategy*, which dictates what

[1]These are but several examples; that BitTorrent uses tit-for-tat is virtually a consensus in the literature.

pieces a peer tells others that it has. The standard practice is for a peer to truthfully report all of the pieces it has to other peers. To the best of our knowledge, we are the first to consider non-trivial piece revelation strategies. We draw inspiration from the economic phenomenon of *decoupling*, in which a group of market countries shed their dependence on another by trading among themselves. We show that a BitTorrent peer has incentive to intelligently *under-report* the pieces it has and to intentionally reduce the multiplicity of others' pieces so as to keep peers uploading to it instead of to each other. With evaluation on PlanetLab, we demonstrate that strategic piece revelation can yield prolonged neighbor interest, which can result in faster download times. We also find that even when *all* peers under-report their pieces to one another, system-wide download times increase by a surprisingly small percentage.

In a system as complex as BitTorrent, there are potentially many components peers could game. We believe that the two we consider—the unchoking algorithm and the piece revelation strategy—constitute the essence of a "BitTorrent-like" file swarming system. To obtain a more complete understanding of BitTorrent's incentives, we also consider what role *piece rarity* plays in strategic behavior. We show experimentally that achieving a block monopoly is infeasible—even with a large fraction of colluding peers—and conclude that BitTorrent's rarest-first piece selection is a natural deterrent to such strategies.

Improving BitTorrent's incentives

The second main contribution of our work is in applying our theoretical understanding of BitTorrent to develop new, more robust incentives. Our auction-based model reveals a rather straightforward modification in the way a peer *clears* its auction, that is, in how much a peer rewards others for uploading to it. We consider a *proportional share* auction clearing, in which each player is rewarded with an amount of good proportional to how much it bid. This approach has recently received attention in resource allocation [8, 12], and has been shown to converge quickly to a market equilibrium [30]. Applying this straightforward model in practice introduces several technical challenges which we address, including how to find other peers with whom to trade, and how to bootstrap new participants.

We emphasize that our proportional share mechanism is intended to *replace* the current unchoking algorithm, not to game it like BitThief and BitTyrant. However, we show that proportional share performs better against BitTorrent peers than the standard protocol. Further, with the intent of incremental deployment, we present a solution that *does not require any modification to BitTorrent's wire protocol*; the only modifications we make are to the *local* decisions a peer makes. Our client is therefore usable today, and we show with extensive experiments on PlanetLab and live swarms that as more users adopt our client, system-wide performance improves.

Roadmap

The rest of this paper is structured as follows. We present related work in Section 2. Section 3 contains the necessary background to our analysis: our goals, assumptions, and the pertinent aspects of the BitTorrent protocol. In Section 4, we present our auction-based model of BitTorrent and use it to prove several properties of BitTorrent, including: it is indeed not tit-for-tat and it is susceptible to Sybil attacks. We consider the role block rarity plays in incentives and consider piece revelation strategies in Section 5. In Section 6, we apply the proportional share mechanism as a BitTorrent unchoker, and prove several properties in the context of BitTorrent: that it is fair, not susceptible to Sybil attacks, and more resistant to collusion. We present in Section 7 several extensions to our proportional share mechanism we found useful in implementation. We have implemented our proportional share client, and evaluate experimentally on PlanetLab, a local cluster, simulations, and live swarms; we present these results in Section 8. In Section 9, we propose a bootstrapping algorithm that improves upon BitTorrent's optimistic unchoking by ensuring that new peers contribute to the swarm as soon as they join. Finally, we conclude in Section 10.

2. RELATED WORK

File swarming has received significant attention from researchers and users alike, BitTorrent [2] in particular. Our work involves modeling, gaming, and improving BitTorrent's incentives.

2.1 Studies of BitTorrent's incentives

Cohen began the study of his BitTorrent protocol's incentives by demonstrating that tit-for-tat-based incentives make BitTorrent robust to strategic gaming [5]. However, Cohen later found strict tit-for-tat to come at too high a cost [4], and weakened the protocol's incentives to achieve better performance. Strategies to game BitTorrent to obtain faster downloads were not long to follow.

Gaming BitTorrent

BitTyrant [22] is a modification of the Azureus BitTorrent client that exploits the "last place is good enough" observation discussed in Section 4. Schneidman et al. [26] were the first to observe this strategy; BitTyrant is an empirical study of its effectiveness. BitTyrant attempts to find the smallest winning bid by beginning at some initial bid and then increasing or decreasing by some small percentage. BitThief [17] studies the feasibility of downloading in BitTorrent without uploading. A BitThief client attempts to enter as many peers' optimistic unchoke slots as possible: the so-called large view exploit [27]. This strategy results in a tragedy of the commons; entering as many optimistic unchoke slots as possible is a rational strategy for all peers to make (it always improves the expected completion time), but when a significant percentage[2] of the peers run the BitThief client, overall performance will logically decrease.

Some have also considered implementation-specific attacks, such as uploading garbage data [16, 26]. We focus only on the components of BitTorrent's protocol that we find to be characteristic of its class of file swarming, not any particular implementation. Strategically gaming BitTorrent helps further the understanding of its incentives; our auction-based model (Section 4) is validated in part by the fact that it captures both BitTyrant and BitThief, and reveals new means of gaming BitTorrent.

Modeling BitTorrent's incentives

Others have considered game-theoretic models of BitTorrent. Coupon replication [18] has been used as a way to model trading in BitTorrent and show that altruism does not play a critical role in file swarming systems' performance, nor that rarest-first block scheduling is of critical importance. This model may prove too simplistic in modeling BitTorrent's incentives in a heterogeneous environment; by assuming that blocks are traded in discrete segments (coupons), it effectively assumes homogeneous bandwidth. Proportional share [8, 12, 30] has been studied in many contexts. Zhang and Wu show that proportional share in a BitTorrent-like system quickly achieves a market equilibrium. However, they do not consider any of the common externalities that can destabilize the equilibrium—

[2]The precise percentage depends on the number of unchoke slots per peer.

block rarity and availability (Section 5), and churn—nor do they address bootstrapping the system (Section 9).

2.2 Improving BitTorrent's incentives

Much work has gone into encouraging cooperation among selfish BitTorrent participants. We present related work based on the incentive mechanisms used to achieve this.

Tit-for-tat

Tit-for-tat is a common incentive mechanism in which peers provide blocks to those who have provided them blocks in the past. BitTorrent was originally described as using tit-for-tat [5]; we discuss BitTorrent's mechanism in more detail in Section 4. Jun and Ahamad [11] propose removing optimistic unchoking from BitTorrent in favor of a k-TFT scheme, in which peers continue uploading to others until the *deficit* (blocks given minus blocks received) exceeds some *niceness* number k. Their results show, as Cohen found with BitTorrent [14], that removing optimistic unchoking increases the average completion time, but does indeed punish free-riders more heavily. Jun and Ahamad observe that if free-riders *can* benefit in a system, then eventually the system will devolve into all free-riders, so a strong disincentive to free-ride is necessary to halt this "evolution." Garbacki et al. [10] consider an *amortized tit-for-tat* scheme that effectively allows contributions made while downloading one file to apply in a tit-for-tat-like manner to other files in the future. Other solutions to this problem generally involve monetary mechanisms.

Monetary mechanisms

Dandelion [28] is a file distribution protocol that uses currency and key exchanges through a centralized server to provide incentive for sharing across different downloads. By requiring centralized servers, Dandelion trades off scalability for incentive-compatibility. BitStore [25] is an exploratory work into applying monetary, second-price auctions

Both Dandelion and BitStore use currency to address what we call the *seeder promotion problem*: providing incentive to peers who have completed downloading a file to seed the file. This is a very important problem, as well; solving it would increase file availability. We believe the mechanisms we present in this paper can complement future solutions to seeder promotion.

Topology-based reciprocation

FOX's structured, cyclic topology provides a means for peer to punish nodes both upstream and downstream from it [15]. Ngan et al. [21] take a similar approach by having peers periodically reform a SplitStream [3] structure, with the hopes that peers who were downstream from a cheater will later be upstream from it, thereby giving it the ability to punish. In addition to the overhead of reforming the structure, peers must wait for a new restructuring to even have the possibility of punishing free-riders. In both of these systems, fairness is defined very strictly; peers receive as much as they give. We find this to be unnecessarily strict, as there may be highly provisioned nodes who are willing to give much more to the system as long as they can download more; FOX, for one, does not provision for this.

Bootstrapping file exchange

We present a mechanism in Section 9 to provide newly joined peers an initial set of pieces of the file to trade. The standard mechanism in BitTorrent is *optimistic unchoking*; each peer uploads to at least one other peer at random, with a weighted preference to new nodes. Optimistic unchoking is the basis of BitThief [17] and the large view exploit [27], while our mechanism encourages peers to trade immediately. Our mechanism is similar in nature to Dandelion [28], but is much lighter weight; it need only be applied when a node has no currency—either when it first joins the system or when it has no blocks of interest to its neighbors—and not for every subsequent block. Also, the technique does not require any trade-off of scalability for incentives, and can be applied to any scenario employing simultaneous exchange of blocks with node churn. *Super-seeding* is a new feature in some BitTorrent clients in which a seeder will upload block b to some new peer p, but will not upload any more blocks to p until the seeder observes that other peers have block b, and hence p must be uploading to others. Our mechanism differs from super-seeding; since ours applies to selfish leechers, not altruistic seeders, we cannot afford to give a block and hope for one to be returned later. Instead, our mechanism ensures that new peers upload blocks *at the same time* as downloading. Further, super-seeding is clearly susceptible to a Sybil attack, in which p requests b, then with Sybil p' states that it has block b. We show that our mechanism is resilient to Sybil attacks.

3. BACKGROUND AND GOALS

In this section, we provide a basic overview of the portions of the BitTorrent protocol pertinent to our study of its incentives. Our description of BitTorrent is intentionally broad so as to capture a broader class of "BitTorrent-like" protocols. We also define the goals of the individual users—their *utility*—and the goals of the system designer—the *social good*. The goal in designing an incentives mechanism is to achieve some desirable, provable properties of the social good, while appealing to users' selfishness.

3.1 BitTorrent basics

BitTorrent peers are classified on a per-file basis as *leechers* if they are downloading (and uploading) the file and *seeders* if they have the entire file and are uploading (only) to leechers. A *tracker* stores a small amount of state to assist leechers in discovering other peers. Files in BitTorrent consist of *pieces* which in turn consist of *blocks*. A leecher ℓ is *interested* in another peer p if p has pieces that ℓ does not; similarly, ℓ finds p *interesting*. All leechers are interested in all seeders.

BitTorrent peers may maintain open connections to multiple *neighbors*, but generally only upload to, or *unchoke*, a small number of them. Users may adjust this number of *unchoke slots*, but it is generally either some small constant (4) or some function of their upload bandwidth. The *unchoking algorithm* dictates to whom and how much to unchoke. To bootstrap, the unchoking algorithm consists of at least one random peer to *optimistically unchoke* regardless of that peer's contribution.

Peers inform one another of the pieces they have; when they first connect to a neighbor, they send a bit array of their pieces, called a *bitfield*, which they later update with per-piece *have messages*. All peers maintain an estimate of each piece's *availability*: a count of how many neighbors have that piece. When a peer p begins unchoking leecher ℓ, ℓ informs p which of p's blocks it wishes to receive next. The common strategy is *rarest-first*, in which ℓ prioritizes the pieces it views as least available.

3.2 Assumptions: Selfishness and rationality

The value that leecher ℓ gains from a file swarming system can be naturally defined to be how quickly ℓ downloads the file. We can thus define ℓ's utility u_ℓ to be the average download speed: if it takes ℓ time T_ℓ to download a file of size F then $u_\ell = F/T_\ell$. Note that we do not define utility at some specific time t, as that would not capture the fact that faster download speeds at the begin-

ning of a download can be offset by poor download speeds at the end. We assume that *all leechers are selfish*—they each attempt to maximize their own utility—and *rational*—for any two strategies, ℓ will choose the one that yields greater expected utility.

Our definition of ℓ's utility does not capture the leecher's cost of uploading; this is intentional. We do not believe it is a leecher's goal to minimize the amount it has to upload in order to download as quickly as possible. Instead, we rely, as do existing BitTorrent clients, on the user to specify the amount of upload capacity they wish to allow BitTorrent to allot. Users are thus expected to find their own best *return-on-investment*, that is, the ideal ratio of utility (download speed) to cost. A peer's upload costs can come in many forms, for instance: (1) actual costs to send, as in a price-per-byte cell phone data plan, or (2) the contention with other networking applications' performance. Upload costs can therefore be extremely complex and change over time. We believe it is best to not include it in a formal definition of u_ℓ, and to instead allow the user to set her own upload capacity. In game theory parlance, the user's chosen upload capacity would represent her *type*, and the optimal advertised type would depend on externalities of which the BitTorrent application is unaware.

3.3 Goals: Desirable system properties

As system designers, we prefer outcomes that maximize some notion of the *public good*, which may in fact be at odds with selfish peers' individual strategies. A goal of our work is to understand the *price of anarchy*: the difference between the social good obtained from fully cooperative peers and that obtained from selfish peers. There are many ways to define public good: max-min utility, max-average utility, max-average return-on-investment, and so on. Strict "get exactly as much as you give" fairness properties seem to require undesirable overhead, such as extensive bookkeeping [20], topology constraints [15, 21], or a monetary system [29]. Further, we find that strict fairness requirements can degrade overall performance. In FOX [15], for instance, peers are motivated to give precisely as much as they receive. Under heterogeneous network conditions, higher-provisioned FOX peers may not have incentive to give more to the system, even though they would have been willing to, if it led to faster downloads.

We favor a simpler, more practical definition of fairness: *the more a peer gives, the more it gets*. Consider the ramifications this fairness property would have on the social good; a highly provisioned peer will upload as much as possible to download as quickly as possible. While serving the selfish peer, the rest of the peers benefit because the highly provisioned peer gives as much as it can. Legout et al. [13] suggest a similar, relaxed form of fairness, and conclude that BitTorrent's current rarest-first and unchoking strategies suffice. We arrive at a different conclusion in section 4.3: that BitTorrent's incentives are not fair, even under this relaxed definition.

4. BITTORRENT AS AN AUCTION

Auctions offer a natural model of interactions among selfish BitTorrent peers. In such auctions, each peer places bids in the form of bandwidth to its neighbors, who in return give bandwidth as the good. With an auction-based model, we expect to gain insight into BitTorrent's incentive structure. Surprisingly, though our specific model is quite simple (a two-line algorithm), it is validated by the fact that it captures recent attacks on the BitTorrent protocol, and reveals new attacks, as well. Thus, while we do not expect our model to be the most accurate representation of BitTorrent, it does provide the necessary motivation and insight into a new solution (Section 6). In this section, we use our model to analyze what incentives BitTorrent gives selfish clients to cooperate, and BitTorrent's susceptibility to gaming.

4.1 An auction-based model

We propose the following auction-based model of BitTorrent. Each peer i separates time into *rounds*[3], which are not synchronized among peers. At round t, i runs the auction in Algorithm 1.

Algorithm 1 BitTorrent's current auction.

1. Run an auction for i's bandwidth; accept bandwidth $b_j(t-1)$ from *interested* peer j as j's bid.
2. Send $1/s$ fraction of i's total outgoing bandwidth to each of the highest $(s-1)$ *interesting* bidders from the previous round, as well as one other *interested* peer at random.

4.2 BitTorrent is not tit-for-tat

BitTorrent has historically been described as employing tit-for-tat. The tit-for-tat strategy in a repeated game is defined as cooperating in the first round, and, in every subsequent round, replicating the opponent's strategy from the previous round [1]. Though similar to tit-for-tat—in that i sends predominately to peers that send to it—Algorithm 1 differs fundamentally in terms of *to whom* i sends.

To see the difference, consider a rational peer p's response to node i playing Algorithm 1. The ideal strategy against tit-for-tat is to cooperate in every round [1]. BitTyrant [22] is built off of the following observation of p's behavior: *It is in a rational player p's best strategy to bid as little as possible to still be one of the $(s-1)$ (deterministic) winners of i's auction.* BitTyrant finds this minimal amount with by adjusting their estimate by small multiplicative factors; Piatek et al. observe that binary search is not suitable as node churn can rapidly alter peer reciprocation. Further, Piatek et al. find that, with a finite upload bandwidth "budget," one should favor uploading to peers who offer a high return on investment, potentially uploading none to peers that offer poor returns.

That BitTorrent does not employ tit-for-tat is not in and of itself a bad thing. In fact, this choice was intentional, to improve performance [4]. However, it brings to light the importance of understanding precisely what properties BitTorrent's incentive structure offers.

4.3 BitTorrent is not fair

There are many definitions of fairness. Our proposed, natural fairness property from Section 3.3 can be formalized as follows: if peer p_1 uploads more to peer q than does peer p_2, then q should reward p_1 more than p_2. This is clearly not the case in BitTorrent. Only the top $(s-1)$ uploading peers are guaranteed to receive any data from q, and each of these receive the same allocation: $1/s$ of q's bandwidth. Further, of the top $(s-1)$ uploaders to q, as they upload more, they are not guaranteed to obtain more. Once a highly-provisioned peer wins all of the s auctions in which it bids, it has no incentive to bid more in these auctions.

4.4 BitTorrent is susceptible to Sybil attacks

A Sybil attack [6] consists of a single host representing itself as many peers in an attempt to gain more from the system than it could as a single peer. BitTorrent is susceptible to two classes of Sybil attacks. The first is in regards to optimistic unchoking. By creating S

[3]Azureus uses 10 second rounds.

Sybils $\{\sigma_1 \ldots, \sigma_S\}$ and using each of them to request to be in others' optimistic unchoking slots, the host will obtain in expectation S times more optimistic unchoking slots. The only Nash equilibrium of such an attack is for all peers to create as many Sybils as possible and, as in BitThief [17], to request to be optimistically unchoked by as many other peers as possible. Indeed, for every peer, this is a *dominant strategy*. Unfortunately, this results in a tragedy of the commons, like in BitThief (see Section 2). We address this problem with a proposed bootstrapping mechanism that would remove the need for optimistic unchoking (Section 9).

The auction in Algorithm 1 reveals another class of Sybil attacks to which BitTorrent is susceptible, but which, to the best of our knowledge, has yet to be exploited by any selfish clients. Recall that a rational peer has incentive to come in last, $(s-1)$'th, place in the auction so that it receives the *same* good for the cheapest price. To obtain *more* good, the peer would have to win more than one slot. Let c_i be the current bid on (cost of) slot i, $c_{s-1} \leq c_{s-2} \leq \cdots \leq c_1$, and let ϵ be the smallest increase such that if a peer were to bid $c_i + \epsilon$, it would win slot i. The optimal strategy of rational peer p with upload capacity U_p is:

Algorithm 2 A Sybil attack on the BitTorrent .

1. Find the maximum k such that $k \cdot (c_{s-k} + \epsilon) \leq U_p$.
2. Create k Sybils, $\{\sigma_1, \ldots, \sigma_k\}$, and with σ_i bid $c_{s-k} + \epsilon$.

To summarize Algorithm 2, p would attempt to come in last (s'th) place, $(s-1)$'th place, ..., and $(s-k+1)$'th place. To do so, p would have to outbid the current $(s-k+1)$'th place peer with each of p's bids, or else the $(s-k+1)$'th place peer would obtain one of the last k places.

The feasibility of Algorithm 2 in practice depends on the number of the victim's upload slots (s), the other peers' bids, and the attacker's upload capacity. The best case scenario for an attacker p would be to find a peer who is currently receiving many small bids, in which case p could potentially win all of the bids at that peer. The worst case scenario for an attacker p would be where p could only afford the last-place slot at any other peer; in this case, Algorithm 2 is identical to BitTyrant. Hence, we can view this Sybil attack as a *generalization of BitTyrant*.

In a sense, BitTorrent could be made more fair if each peer employed this Sybil-based strategy; if a peer so desired, it could upload more (with more Sybils) to a fast neighbor to download more. The mechanism we present in Section 6 has the similar property of "the more you give the more you get."

General-purpose solutions to Sybil attacks could apply, such as money [29] or proofs of work [7]. We show in Sections 6 and 9, however, that there exist natural incentive schemes that are Sybil-proof and would not require additional infrastructure or significant computation overhead.

4.5 BitTorrent's susceptibility to collusion

BitTorrent's auction-clearing mechanism is open to collusion. To date, selfish BitTorrent clients such as BitTyrant and BitThief have considered only non-collusive strategies.[4] We briefly sketch here how one could collude against BitTorrent. In Section 6, we present a mechanism that is less susceptible to this class of gaming.

A set of colluding peers can form a *coalition* $\mathcal{C}$ against a set of victims $\{v_1, \ldots, v_k\}$ by simply agreeing with one another to upload only nominal amounts to each v_i. The coalition is built on the premise of "turning victims into seeds" by uploading only a nominal amount of data to the victims and getting a full unchoke slot in return.

The feasibility of such an attack depends on how many "cheap slots" victim v_i has. In the best case scenario for a coalition $\mathcal{C}$, each v_i would currently not be receiving any bids, in which case the coalition can effectively act as the Sybil attack in Algorithm 2 and achieve all of each v_i's unchoke slots. In the worst case scenario, each v_i with s_i slots would be receiving at least s_i large bids. Put another way, for a victim v_i to be resilient to collusion, it must have s_i neighbors who are not participating in the coalition. We present a mechanism in Section 6 in which collusion fails as long as a *single* neighbor is not in the coalition.

5. MAXIMIZING INTEREST

In the auction in Section 4, a peer sends only to those in whom it is interested, and, equivalently, receives only from those interested in it. Clearly, a peer benefits from being interesting to as many of its interesting peers as possible. A peer's interest in one of its neighbors is based solely on what pieces that neighbor claims to have. In this section, we consider what piece revelation strategies a peer can employ to maximize the number of neighbors who are interested in it. That such strategies are feasible is counter-intuitive; it seems natural that peers should advertise all they have to offer to remain interesting. We show, however, that a peer can prolong interest by *under-reporting* what blocks it has.

A natural question follows: how successful can an under-reporting strategy be? Can a peer, for instance, obtain a piece monopoly? We show empirically that piece monopolies are infeasible. BitTorrent's rarest-first piece selection strategy is a viable deterrent to the style of piece *hoarding* that would be necessary to obtain a monopoly, even when a large number of leechers collude.

To the best of our knowledge, we are the first to consider non-trivial piece revelation strategies. BitThief's strategy is effectively to obtain as large a set of neighbors as possible and to reveal that it has no blocks of interest to any of its neighbors [17]. Shneidman et al. [26] suggest *over-stating* what blocks a peer has and uploading *garbage* blocks, but this is infeasible, as peers can easily detect and punish this defection. Other theoretical work assumes either an infinitely-sized file [30] or cooperative peers [18].

5.1 Leechers want all the attention

We begin our study of piece revelation strategies by considering what outcomes are preferable to leechers. In Figure 1, we show a leecher i's preferences over various scenarios, and denote preferences $\succeq_i$. Clearly, i prefers to have as many neighbors that are *interesting* to i to be *interested* i as possible. This will result in more peers bidding at i, and thus faster download times. Thus, (a) $\succeq_i$ (c) $\succeq_i$ (e).

The degree to which leecher i's neighbors find *other peers* interesting affects i's future interest. If i's neighbors trade pieces that i has with one another, then i risks becoming uninteresting to its neighbors sooner than if i's neighbors *only* had interest in i. This gives us (b) $\succeq_i$ (d) $\succeq_i$ (f) from Figure 1.

Note that case (f) can only occur when i does not know at least one of j or k. To see this, suppose the converse: that j and k are interested in one another, that i has no interest in either, but that they are all neighbors with one another. Then j has some piece p_j that k does not have. Since i is not interested in j, i must have p_j. k would therefore be interested in i due at least to i having p_j, a contradiction.

Nonetheless, (e) $\succeq_i$ (f); i prefers the scenario in which *no* peers are interested in one another to the scenario in which *other* peers

[4]Though BitTyrant peers ramp allocations to one another, they do not directly collude against BitTorrent peers.

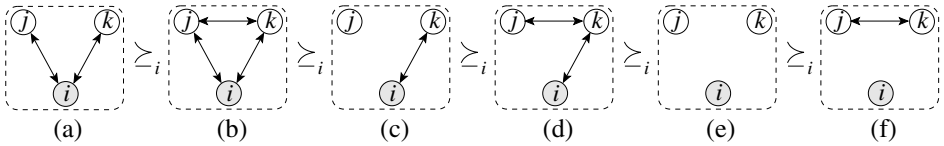

Figure 1: Node i prefers to be as interesting as possible, and more interesting to its neighbors than its neighbors' neighbors.

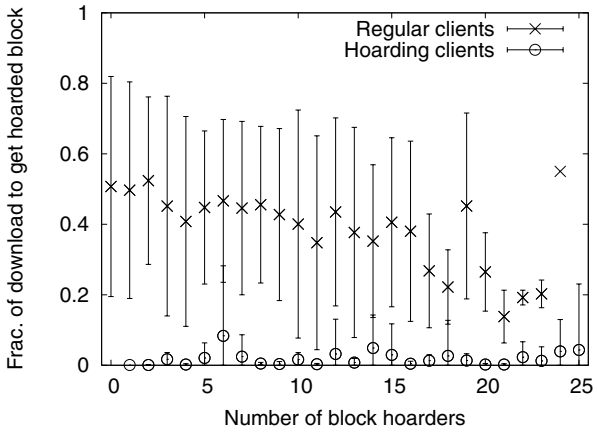

Figure 2: Rarest-first overcomes attempts at block monopolies.

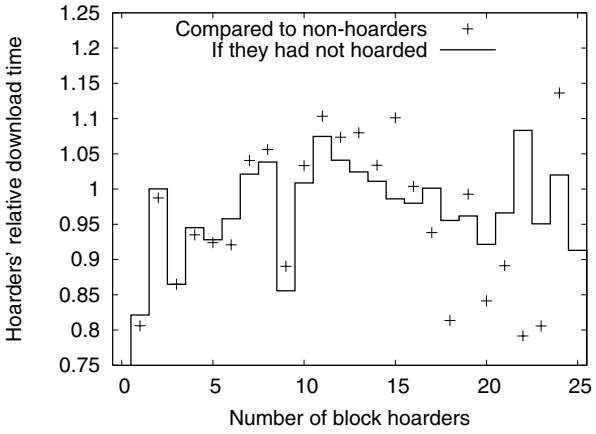

Figure 3: Hoarders' download times.

are progressing while i is not. This is because the sooner j and k finish downloading the file, the sooner they may leave the swarm, reducing the potential number of peers eventually bidding at i. This is an example of when selfish participants' preferences conflict with the social good.

In the remainder of this section, we consider strategies a leecher i can take to achieve preferable scenarios from Figure 1.

5.2 Forcing a block monopoly is infeasible

A tempting initial attempt at maximizing a peer's demand is to try to obtain a *block monopoly*. If p were the only peer in the system with some block b, then p could extort bandwidth from all other peers, giving out b to others at a rate so slow that p is the only one making discernible progress toward completing the download. Of course, with a seeder, a single node obtaining a monopoly is virtually impossible.

We consider here the feasibility of a more relaxed form of monopoly: forcing the rarity of a block up by intentionally *hoarding* it, that is, refusing to give it out to peers until it becomes so rare that peers may be willing to pay a premium price for them. Clearly, BitTorrent is sufficiently robust to overcome a single hoarding node, but what about a colluding set of peers? We answer this by modifying the Azureus BitTorrent client to perform block hoarding. In our implementation, a hoarder knows the blocks he wishes to hoard *a priori*, attempts to obtain these block(s) as soon as possible, and then refuses to inform others (with BitTorrent have messages) that he has the block. We ran experiments on a local cluster of 25 machines, varying the number of nodes hoarding the same piece.

Figure 2 shows that normal (non-hoarding, rarest-first) clients overcome block hoarding; that is, with an arbitrary fraction of hoarders, the normal clients obtain the block. Indeed, they obtain it more quickly; the expected point in the download at which a block is downloaded is 50% into the download, but the hoarders force the observed availability of the block down, causing normal clients to attempt to download it sooner. This is reflected in Figure 2; with no hoarders, the given block is centered around the expected value of 50%, but with more hoarders, normal clients tend to obtain the block sooner than halfway through the download.

Our experiments capture what happens when a node's hoarding strategy does *not* work, in the sense that at no point does a hoarder reveal its hoarded block and attempt to use it to download more from others. Figure 3 demonstrates that, when a hoarding strategy does not work, it can have adverse effects on the peer's download time. The dots represent the relative download times to the other, non-hoarding clients, and the histogram represents the hoarders' download time relative to if they had not hoarded, i.e., relative to the download time when the number of hoarders is zero. One can interpret this figure as follows: it is in a peer's best interest to hoard blocks only when the dots are below the line.

There is a parabolic trend in Figure 3 that is consistent across all runs of this experiment. A few hoarders experiences faster download times than non-hoarding peers, slower when the number of hoarders is roughly equal to the number of non-hoarders. When hoarders far outnumber non-hoarders, hoarders perform comparatively faster, but, as shown by the solid line in Figure 3, harm overall performance.

We conclude that hoarding-based strategies *can* work, but (1) since it would require such widespread collusion, it is infeasible, and (2) given that it can *increase* download times if many peers perform it, it may not be a worthwhile risk for a peer to take.

5.3 Under-reporting to prolong interest

In both the BitTorrent and prop-share auctions, bidders consist solely of the peers *interested* in the peer running the auction. Although forcing a block monopoly appears to be infeasible, increasing the number of bidders—the number of interested peers—is possible with *strategic piece revelation*.

A piece revelation strategy dictates which blocks a peer claims to make available to its neighbors. Suppose β_i represents peer i's block bit-field, where $\beta_i(k) = 1$ if i has block k and 0 otherwise. i reports its bit-field to new neighbors, and sends updates in the form of have messages to existing neighbors. Let β_i' repre-

sent the bit-field i presents its neighbors, which need not be its *true* value β_i. If i *over-reports* block k, that is $\beta'_i(k) > \beta_i(k)$, then i's neighbors could easily detect this by not obtaining k upon request, and could subsequently punish i by allotting him less bandwidth. Hence, we can assume that $\beta'_i(k) \leq \beta_i(k)$, that is, that i will not over-report the blocks it has, but may under-report. Further, if i reports some block k and peer j requests it, then i must deliver that block. Otherwise, if i sends block k', it may hurt j's download time—j may have requested k' from another peer, and thus j could have spent time downloading two copies of the same block—thus j would have incentive to punish i. Hence, we can further assume that peer i delivers the blocks it reports, and that the only avenue for strategic piece revelation is under-reporting.

An under-reporting strategy

Why would i under-report its blocks? Certainly, i would not under-report to the point of becoming uninteresting to its peers, as the more interesting i is, the greater number of potential bids it can receive. A myopic peer might attempt to maximize its interest by declaring its true bit-field, as BitTorrent and BitTyrant currently do. However, a peer need not report *all* of the blocks it has to be equally interesting. Further, it is important to note that *the blocks that i gives out at round t affects how interesting other peers find i in future rounds*: Consider the simple motivating example in Figure 1(a), in which i has two neighbors j and k, both of whom find i interesting but do not find one another interesting. This scenario is preferable to i; both j and k will bid at i and i alone. Were i to truthfully report $\beta_i(t)$, j and k could request from i different blocks, which would make j and k interested in each other. This would then place i into scenario (b) or, if those were the only two blocks holding j and k's interest, scenario (e). Thus, i would under-report his blocks in order to maintain his neighbors' prolonged interest.

We conclude that peers indeed have incentive to strategically *under*-report what blocks they have. This leads us to the piece revelation Algorithm 3.

Algorithm 3 Strategic piece revelation. Run by i when peer j becomes uninterested in i.

1. Let β'_j denote j's bitfield, and $L_i(j)$ the list of pieces that i has revealed to j.
2. If there does not exist any piece p such that $\beta'_j(p) < \beta_i(p)$ then quit; i cannot truthfully gain j's interest.
3. Find the piece p with $\beta'_j(p) < \beta_i(p)$ that *maximizes* the number of other neighbors k for which (i) k also has p: $\beta_k(p) = \beta_i(p)$, or (ii) i has revealed p to k: $p \in L_i(k)$.
4. Send a have-message to j, revealing that i has piece p, and add p to $L_i(j)$.

Algorithm 3 is reactive; peer i only reveals that he has a piece to j when j loses interest. Instead of revealing the rarest piece he has, peer i reveals the *most common* piece he has that j does not. Providing j with a rare piece would make j more interesting to his neighbors, potentially removing some interest from i. This corresponds to i attempting to maintain the state in Figure 1(a) as opposed to (b), (d), or (f).

Evaluation of strategic piece revelation

We evaluate our strategic piece revelation on PlanetLab, and present the results in Figure 4. The experiment consists of two runs. In both, all but one peer start at the same time and run the standard BitTorrent client, revealing all of their pieces. The remaining peer runs one of the two strategies and starts 20 seconds after all the other peers, to test whether strategic block revelation can maintain the interest of peers who have many more blocks than he does. This result clearly shows the power in under-reporting one's pieces; the strategic peer is able to maintain others' interest over a prolonged period of time. Also clear from Figure 4 is the power of maintaining interest; the strategic peer downloads more quickly, as more peers place their bids throughout the download.

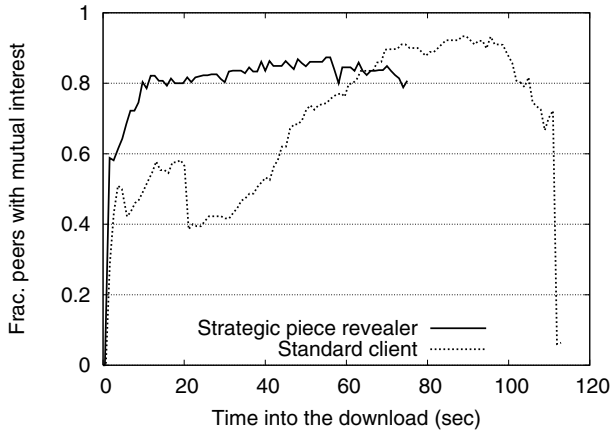

Figure 4: An example run, showing that strategic piece revelation can yield prolonged interest from neighbors, and hence faster download times in general.

It seems intuitive that strategic piece revelation would induce a tragedy of the commons: that as the number of strategic piece revealers increases, system-wide download times would be severely increased. We ran experiments in which all peers strategically revealed their pieces. We saw on average a 12% increase in system-wide download times when *all* peers strategically revealed. That there is an increase in download times is unsurprising; a strategic piece-revealing peer does not know *which* pieces to offer to another strategic peer to garner its interest. The increase in system-wide download times is not much higher because peers continue to reveal pieces to their neighbors until they gain their interest.

These results indicate that selfish BitTorrent clients can benefit from under-reporting, but that as this practice becomes widespread, there may be an overall performance loss. There is therefore more to be understood about how strategic piece-revealers should interact with one another. For instance, one area of future work is to consider strategic *interest* declaration, as a means of encouraging neighbors to reveal as many pieces as they truly have.

6. CLEARING AUCTIONS WITH PROPORTIONAL SHARE

The auction-based model of Section 4 motivates a new means of clearing BitTorrent peers' auctions. We extend recent results on the market equilibria achieved by *proportional share* [30]. Let $b_x^y(t)$ denote the amount of bandwidth x gives to y during round t. Algorithm 4 captures a proportional share auction, run at round t.

Clearly, the prop-share auction requires bootstrapping; peers must give some initial amount of goods to one another to obtain any in return. Though similar to BitTorrent's optimist unchoking in that a peer must initially give without getting anything in return, the amount of "altruism" of a prop-share client is much less. A peer

Algorithm 4 Proportional share auction clearing.

1. Run an auction for i's bandwidth; accept bandwidth $b_j^i(t)$ from peer j as j's bid.
2. Let B_i represent i's total available upload bandwidth. Send to peer j his *proportional share*:

$$b_i^j(t) = B_i \cdot \frac{b_j^i(t-1)}{\sum_k b_k^i(t-1)}. \quad (1)$$

could give an arbitrarily small amount, e.g., a single block. In the subsequent round, a prop-share client will ramp up its allocations according to Eq. (1). This bootstrapping mechanism also serves as a means of discovering new peers; we return to this point in the context of "finding a good deal."

Our extensions to Zhang and Wu's work [30] focus predominately on how this simple mechanism can be applied to dynamic network settings. A successful mechanism must maintain BitTorrent's proven robustness to such conditions.

We stress that prop-share is intended to be a *replacement* of BitTorrent's current auction clearing mechanism, not a strategy to game existing BitTorrent clients. As we showed in Section 4.4, BitTyrant with a Sybil attack is the ideal strategy against BitTorrent, and we do not expect prop-share to perform as well as this strategy. In the rest of this section, we show that prop-share is (nearly) a Nash equilibrium, and is thus not susceptible to a single peer's deviation.

6.1 Best response to prop-share

We can capture peer i's *best response* to all other nodes playing proportional share as follows:

$$\text{maximize} \quad \sum_j B_j \cdot \frac{b_i^j(t)}{\sum_k b_k^j(t)}$$
$$\text{s.t.} \quad \sum_j b_i^j \leq B_i \;\text{ and }\; \forall k : b_i^k \geq 0$$

An immediate observation of this nonlinear program is that each peer has incentive to allocate all of its upload bandwidth, that is, $\sum_k b_i^k = B_i$. The solution to this nonlinear program [8] involves finding the bid to peer j at which the marginal benefit of increasing that bid is less than the marginal benefit of increasing the bid to some other peer k.

The best response to all peers playing prop-share is distinct from prop-share; that is, *prop-share is not always a Nash equilibrium*. This is because there may be some peers whose marginal benefit never exceeds others', and the best response is to never send to these peers. Conversely, with prop-share, peer i will provide bandwidth to *all* peers that uploaded to i in the previous round.

Computing the best response to other nodes playing prop-share requires peer i to know each of its neighbors' upload capacity, B_j, and the sum bids its neighbors have received from other peers. This information is reasonable to assume in the setting Feldman et al. [8] studied, wherein the auctioneers are not profit-maximizing, but this is clearly not the case in file swarming. One could envision trying to estimate these values, but the accuracy of such an estimation would be at best difficult to ensure, and would likely require extensive book-keeping.

6.2 Prop-share is enough

We show that simply employing prop-share with incomplete information achieves nearly the same return on investment as having perfect information, even against perfectly-informed peers. To demonstrate this, we simulated the scenario in which all peers except for peer p play prop-share, and compare p's allocations and resulting download speed when it plays prop-share versus the best-response algorithm provided by Feldman et al. [8]. In the case of best-response, we allowed p to have *complete information* of other peers' bandwidth, B_j, and bids, b_{-i}^j. For prop-share, such complete information is unnecessary, as p needs only know the local information of how much each peer gave to p in the previous round. We present values averaged over 30 runs, varying the number of peers as well as p's bandwidth. Each game lasted for 30 iterations, but we observed that they converged after just a few rounds, confirming previous results [8, 30].

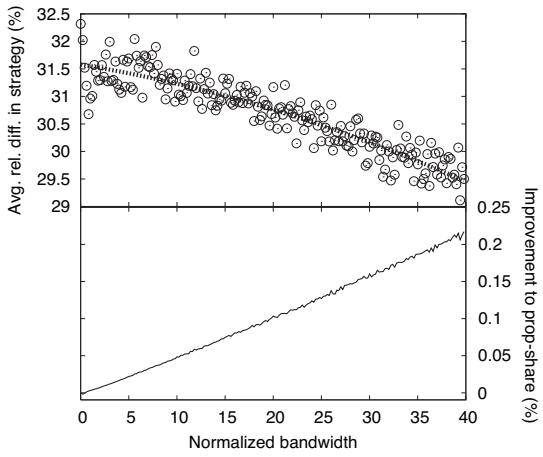

Figure 5: Prop-share is distinct from the best response. Though significantly different, completion times were consistently no worse than 0.25% longer.

First we consider the question: does the best-response strategy result in similar allocations to those of prop-share, that is, are they effectively the same strategy? Figure 5 shows the average relative difference between best-response and prop-share allocations; on a per-peer basis, the best-response allocation differs by an average of roughly 30% from prop-share, and decreases as p has increasing bandwidth. Put simply: *best-response and prop-share strategies result in vastly different bandwidth allocations*. Even as a peer's "bargaining power" increases, best-response remains consistently "different" from prop-share, which we demonstrate by allowing p's bandwidth to vary from .1 to 40 times the average system-wide bandwidth.

Surprisingly, the clear difference between best-response and prop-share results in little improvement to utility. Best-response improved download speed by significantly less than 1% in all of our simulated scenarios. These results lead us to conclude that *prop-share is the preferred response to all other peers playing prop-share* because it achieves download speeds nearly equal to best response and, importantly, does not require complete information.

6.3 Prop-share is Sybil-proof

BitTorrent's auction is susceptible to Sybil attacks because its auction returns discretized goods (Section 4.4). We show that prop-share, on the other hand, is resilient to Sybil attacks. Note that a Sybil attack applies only to a given neighbor v. Let y_v denote the sum of other peers' bids at v. Suppose peer p were to create S Sybils, $p_1, \ldots, p_S$, where p_i contributes c_i to some victim neighbor v, for a total contribution $C = \sum_{i=1}^{S} c_i$. To show that prop-share is Sybil-proof, it suffices to show that, for a fixed amount of

contribution C, p will receive the same amount of bandwidth for any set of Sybils, including the set of size 1. Indeed, for an arbitrary set of Sybils, p will in turn receive from prop-share-playing node v:

$$\sum_{i=1}^{S} \frac{B_v \cdot c_i}{\sum_{j=1}^{S} c_j + y_v} = \frac{B_v \cdot \sum_{i=1}^{S} c_i}{\sum_{j=1}^{S} c_j + y_v} = \frac{B_v \cdot C}{C + y_v} \quad (2)$$

In practice, Sybils perform even worse against a prop-share client. Each Sybil would require additional communication overhead for transmitting various protocol messages: declaring interest, listing the pieces they have, etc. Eq. (2) shows that Sybils would not improve performance even without such overhead, and would in practice result in poorer performance, as the peer would be forced to spend more of its budget (bandwidth) on protocol messages in lieu of data.

6.4 Prop-share is (more) collusion-resistant

Coalitional strategies would succeed against BitTorrent because no colluding peer has incentive to upload more to its respective victim; doing so would not result in greater download speeds. This property does not hold in prop-share, and is the basis for prop-share's collusion resistance. Consider a coalition $\mathcal{C}$ against a victim v who is playing prop-share. Suppose that, if the coalition were to play prop-share, then peer $i \in \mathcal{C}$ would have uploaded b_i^p to p and received fraction $f_i = \frac{b_i^p}{\sum_k b_k^p}$ of p's bandwidth. Let $f_{min} = \min_{i \in C}\{f_i\}$. Ensuring that each member of $\mathcal{C}$ obtains as much bandwidth from p as if each member were to strictly play prop-share is one way to keep the coalition from dissolving To achieve this, each coalition member i can agree to upload bandwidth $\epsilon f_i / f_{min}$ to p, where ϵ is some nominal amount of bandwidth. p will thus return to i a fraction of p's bandwidth equal to

$$\frac{\epsilon f_i / f_{min}}{\sum_{j \in \mathcal{C}} \epsilon f_j / f_{min}} = \frac{f_i}{\sum_{j \in \mathcal{C}} f_j} = \frac{f_i}{1} = f_i$$

as long as the only peers uploading to p are the members of the coalition. If even a single peer $k \notin \mathcal{C}$ were to bid at p during $\mathcal{C}$'s collusion, k stands to gain a large return on investment. Indeed, as the coalition becomes more beneficial to its users ($\epsilon \to 0$), its susceptibility to being dissolved increases; k's share of p's bandwidth as the coalition's nominal bandwidth decreases is

$$\lim_{\epsilon \to 0} \frac{b_k}{b_k + \sum_{j \in \mathcal{C}} \frac{\epsilon f_j}{f_{min}}} = \frac{b_k}{b_k + 0} = 1$$

Hence, as long as there is a single peer not in $\mathcal{C}$, the coalition as a whole will have to dissolve, in which case all the members of $\mathcal{C}$ play prop-share. Compare this to BitTorrent, in which a node with s slots needs s non-colluding peers to dissolve a coalition.

While this demonstrates prop-share's natural defense against collusion, we discuss an extension in Section 7 that provides prop-share greater resilience.

7. IMPLEMENTATION

A nice property of the proportional share mechanism in Section 6 is its simplicity; each peer needs only to recall what its neighbors have most recently sent, and reply proportionally. An evaluation of proportional share's incentive properties is thus rather straightforward, and prove to be strong in theory. It is not unreasonable to assume that providing strong incentive properties would result in a decrease in performance. To study this, we have implemented a proportional share client, and evaluate it on PlanetLab and live swarms in Section 8. The core of the mechanism is the same in implementation as in Algorithm 4. We present in this section additional features we have found important in implementation.

7.1 Finding a good deal

Some peers are better deals than others, in the sense that some have higher values of $B/\sum_{j \neq i} b_j$. Zhang and Wu [30] assume that a node's neighbors are given, and show that proportional share converges quickly to an equilibrium. In reality, peers may learn of others' and *strategically* change their neighbor set. An optimal strategy would require a peer to have global knowledge of all other peers. The "large view exploit" [27], that BitThief [17] employs, attempts precisely this by continually requesting peers from the tracker, but with the goal of being optimistically unchoked by as many peers as possible. In fact, it is in any BitTorrent peer's best interest to learn of as many peers as possible so as to try to download from those with which it has a better connection.

However, such global knowledge does not scale, and instead each peer operates on a smaller subset of neighbors. Instead, in our implementation, we do not modify the number of neighbors a peer gets from the tracker, but a combination of prop-share and the large-view exploit would, we believe, result in a more efficient market equilibrium.

Bootstrapping prop-share allows peers to "research" new neighbors and potentially find those with better return on investment, that is, larger values of $B_j/\sum_k b_k^j$. Zhang and Wu's analysis [30] assumes a fixed topology and initialize everyone's sharing to some random value. However, in practice, node churn variable traffic conditions require peers to discover new neighbors. We propose that peers allot a fraction (80% in our implementation) of their bandwidth to returning proportional shares to their neighbors, and use their remaining "budget" to research new neighbors. This requires no changes to the BitTorrent wire protocol, and is what we use in our implementation.

7.2 What have you done for me lately?

Recall that a prop-share peer runs its auction (Algorithm 4) once per round, using only the information of its neighbors' contributions from the previous round. This simple mechanism can lead to oscillations. Suppose for instance that at round t, peer p decides to "research" his neighbor q as above. In the following round, q will reply with p's proportional share, but since q did not upload to p in the previous round, p uploads none to q. This will clearly continue oscillating unless there is some external force to bring them to convergence. In our implementation, we employ a weighted average of a node's neighbors' four most recent contributions, and reward peers proportionally based on this.

7.3 Fighting collusion with ratio caps

Although prop-share is *more* collusion-resistant than BitTorrent (Section 6.4), there still exists the possibility for a large coalition to extort a victim's bandwidth at little cost. Protecting peers from collusion is difficult, as it is often unclear whether nodes are, as a coalition, appearing slow to some victim, or whether they are actually slow. There is a very interesting trade-off here: One could err on the side of caution, and better protect peers from periods of attack. Alternatively, one could err on the side of the social good, allocating large amounts to new neighbors early at the risk of them not reciprocating the favor.

In our implementation, we simply cap the amount of bandwidth a peer gives to any of its neighbors to a factor, f. This is similar in nature to k-TFT [11], but differs largely in the sense that, in our implementation, it is intended to only be applied when there is a high disparity. When a peer attempts to undersell another peer,

this mechanism will keep that peer to achieving at most f times the amount of bandwidth it sends. When two peers wish to upload to one another, the multiplicative factor ramps up the allocation exponentially fast.

8. PROPSHARE EVALUATION

In this section we present our experimental evaluation of our PropShare client.

We emphasize the goal of our PropShare client: *to maintain robust incentive properties without sacrificing speed.* The previous sections of this paper show PropShare's resilience to many forms of strategic gaming. Demonstrating these points experimentally is difficult if not impossible, as one failed attempt at gaming a system is hardly proof that it is impervious to strategic manipulation. The auction-based model we presented in Section 4 is intended to serve as a general tool for a rigorous study of incentives in BitTorrent-like settings where experiments do not apply. Here, we focus on PropShare's performance both in live swarms and on PlanetLab.

8.1 How do we expect PropShare to perform?

In our evaluation, we compare our PropShare client to BitTorrent and BitTyrant. While BitTyrant was built specifically to game BitTorrent, the goal of our PropShare client is to provide robust incentives even against future clients. Our implementation therefore takes *no BitTorrent- or BitTyrant-specific actions.* As such, we expect that BitTyrant will perform better in a swarm consisting predominately of BitTorrent peers.

In the publicly available BitTyrant implementation, BitTyrant peers ramp bandwidth allocations to one another using a k-TFT-like scheme. This strategy is not shown to be strategyproof [22]. We considered modifying the BitTyrant client to remove this potentially game-able strategy, or to announce our PropShare client as a Tyrant to gain from others' sharing. However, in practice, one would not be able to do this, and there may be many peers employing many different strategies. We thus opted to let our PropShare client "fly blind" against BitTyrant peers.

8.2 Experiments on live swarms

We begin our evaluation by comparing performance of BitTorrent, BitTyrant, and PropShare on live swarms. In these experiments, we chose torrents with a large leecher-to-seeder ratio to test the various clients' ability to trade with others. We started the three clients simultaneously from three separate machines, each on the University of Maryland network. We limited each client's upload bandwidth to 100 kilobytes per second.

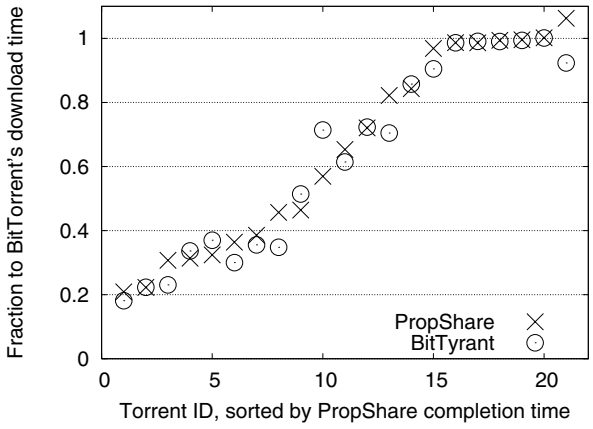

Figure 6: Runs on live swarms

Figure 6 validates our hypotheses regarding PropShare's performance relative to other clients. We plot results from 21 live swarms, sorted by PropShare completion time. BitTyrant, being tailored specifically to exploiting BitTorrent peers, frequently performs the best of the three clients. Although our PropShare client is a straightforward realization of the proportional share mechanism from Section 6 that employs no BitTorrent-specific mechanisms, it performs comparably well to BitTyrant, and in all but one download, much better than BitTorrent. We believe PropShare experiences good performance in live swarms because it shares some similarities with BitTyrant; PropShare allocates more bandwidth to peers with greater return on investment. The main difference between PropShare and BitTyrant is that PropShare will reward all peers, even those with poor return on investment, with bandwidth. The results in Figure 6 indicate that this additional expenditure is not detrimental, and in some cases improves performance.

We conclude that PropShare is incrementally deployable. Users could begin using and benefiting from a PropShare client today, and we intend to make our client available as open source. PropShare does not require a full deployment to benefit, or a change in the protocol to improve performance. This is a result of PropShare's resilience to strategic manipulation; PropShare does not require mechanisms such as voting or long-term agreements in order to benefit, and is even resilient to colluding peers. It is thus natural that a single PropShare peer would perform well in a swarm of non-PropShare (but rational) peers.

8.3 Competitive experiments

It is reasonable to assume that the vast majority of the peers contacted in our live swarms experiments ran unmodified BitTorrent clients. We now study how PropShare performs when either it or BitTyrant hold the majority.

Experimental setup

We ran competitive experiments on roughly 110 PlanetLab nodes. In these experiments, we pitted two clients against one another by keeping the number of peers fixed across all experiments, but varying the relative fraction of client types. We adopted Piatek et al.'s BitTyrant experimental setup: three seeders per file, with a combined upload bandwidth of 128KBps, a locally run tracker, and peers downloading 5MB files. We used the bandwidth distribution presented by Piatek et al. [22], and varied each peer's bandwidth cap in each run. Each peer left the swarm as soon as it was done downloading the file. We reduced dependencies across runs by not using the same file between two separate runs; many trackers impose a limit on how often peers can request new peers, which could force some of the slower peers from an earlier experiment to be forced into long waits at the beginning of the next. Each pair of points in the figures that follow represents the average over at least 3 runs, and error bars denote 95% confidence intervals.

BitTyrant vs. BitTorrent

The original BitTyrant study [22] measured average download times on swarms consisting of all BitTyrant or all BitTorrent peers, or when one BitTyrant peer attempted to game the rest of the system. We augment that study by considering intermediate ratios of clients. Figure 7 shows that there are interesting dynamics between the two extreme points. There is a clear trend toward an increase in BitTyrant performance as there are fewer of them.

The trend of Figure 7 is clear within the context of the auction model of Section 4. Consider a strategic bidder b; if there are few

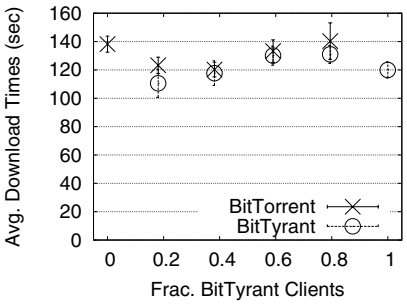

Figure 7: BitTyrant vs. BitTorrent

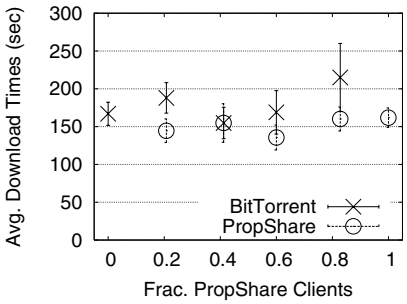

Figure 8: PropShare vs. BitTorrent

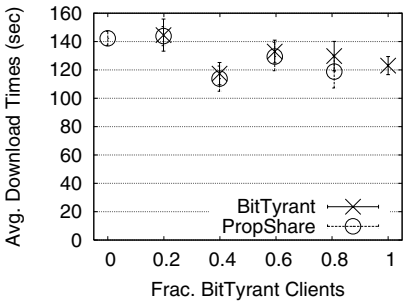

Figure 9: PropShare vs. BitTyrant

other strategic bidders in the system, then b can more easily obtain the last (but still paying) place in the auction. As bidders begin to learn that they do not need to place such large bids at their respective auctions, they are free to bid at more peers. This in turn raises the contention at each peer for the last-place slot.

Figure 7 also reveals a tragedy of the commons. The users with the best download times in this figure are the BitTyrant peers who were in the *minority*; the early BitTyrant adopters, so to speak, were rewarded well, but as more users switch to BitTyrant, overall performance degrades for all users. With a majority of BitTyrant peers, the predominate interactions system-wide are between BitTyrant peers; their strategy of ramping up bandwidth allocations to one another compensates for this tragedy of the commons. The end result is a system with overall better performance than BitTorrent—BitTyrant incurs lower average download times as a whole—but one that is not known to offer strategyproofness guarantees [22].

PropShare vs. BitTorrent

When run against BitTorrent, PropShare clients exhibit different behavior than do BitTyrant clients. Rather than experience the best performance when a minority, PropShare clients maintain low download times even as the number of PropShare clients increases. This appears to come at the BitTorrent peers' expense, as PropShare induces more significant increase in download times for non-PropShare peers. This implies, along with our live swarm results, that PropShare not only can be used today to achieve better download times, but that its performance will not degrade as more users switch to it.

PropShare vs. BitTyrant

Next, we run PropShare clients against BitTyrant clients. In each *mix* of clients in Figure 9, PropShare clients out-perform BitTyrant peers. This by no means proves that PropShare is not game-able, but lends further credence to PropShare's ability to be used today, among clients of varying strategies. A swarm consisting of all PropShare clients performs worse on average than a swarm of all BitTyrant clients. That PropShare would even be competitive while BitTyrant peers ramp allocations to one another is encouraging; it shows that *PropShare's cost of robust incentives is low*.

	All other peers	
One peer	BitTyrant	PropShare
BitTyrant	86.9 (6.40)	109 (15.6)
PropShare	70.5 (4.61)	107 (14.3)

Table 1: Average download times (seconds, with standard deviation) for a single client choosing between two strategies.

All-versus-one

From Figure 9, we see a general decrease in download times as the fraction of BitTyrant peers increases. We now study the cause of this. BitTyrant effectively runs two parallel strategies: one against other BitTyrants, and one against all other strategies. The inter-BitTyrant strategy has not been shown to be strategyproof [22]. We do not consider here how one might game such a strategy. Instead, we focus on testing the following hypothesis: that BitTyrant's improved download times are strictly a result of the (potentially gameable) actions they take against one another, and not indicative of BitTyrant gaming PropShare. To test this hypothesis, we run all-versus-one experiments, where $N-1$ peers run one strategy (PropShare or BitTyrant) the remaining peer runs alternates between the two strategies in subsequent runs. We present our results in Table 1. These results are consistent with Figure 9, in that a majority of BitTyrant peers yields faster system-wide download times on average. Table 1 shows that a BitTyrant peer does not on average successfully game PropShare clients, and that in fact a PropShare client is the preferred strategy against a swarm of all (other) BitTyrants. We conclude that the decrease in download times observed with more BitTyrant peers is not a result of gaming PropShare, but the speedup from the potentially game-able inter-BitTyrant strategy.

9. BOOTSTRAPPING FILE SHARING

Viewing BitTorrent as an auction further motivates the need for seeding: how can a node bid in an auction without any form of "currency" (blocks)? A node can exploit the optimistic unchoking by going to each peer; this is the so-called large view exploit [27] on which BitThief [17] is based. Optimistic unchoking attempts to trade off robust incentives for a greater social good by decreasing the average download time. However, the lack of incentives opens it to attack, which decreases in social good [17]. One might think that the seed nodes are enough to bootstrap the system, but in fact this is not strictly true, and is what motivates the need for optimistic unchoking at all peers. This is a somewhat orthogonal problem to the auction mechanism, so we propose a mechanism that solves this problem in the more general context of *bootstrapping* piece exchange.

Suppose two nodes A and B have been trading blocks for multiple rounds, and a new node n has just requested that A "bootstrap" him by giving him blocks. A may be suspicious for two reasons: (1) n may simply take the block and leave, never giving A anything in return, or (2) n may be a Sybil [6] of B (or some other node), and B is simply trying to extort more bandwidth out of A.

A may settle his suspicions by agreeing to send block b encrypted with a symmetric key K_A, $[b]_{K_A}$, to n. A must ensure that the bandwidth it gives to n is not "wasted," i.e., that (1) A receives bandwidth in return, and (2) A's established trading with B

is not adversely affected. Thus, A does not reveal K_A unless n forwards the block to B.

Algorithm 5 Bootstrapping piece exchange

1. n requests a block (of A's choosing) from A.
2. A informs B that it will be using n as a "proxy." A encrypts b with K_A and sends a hash of $[b]_{K_A}$ to B.
3. A informs n to forward packets to B.
4. A sends $[b]_{K_A}$ to n, which n forwards to B. Concurrently, B sends an encrypted block $[b']_{K_B}$ to A.
5. B verifies that the hash of the block n sent matches what A sent. If correct, B informs A that it received the correct block from n, i.e., that n performed the task as instructed.
6. If B sends A the proper hash, then A reveals K_A to n and B, and B reveals K_B to A.

The newcomer, n, pays his dues in two ways: (1) by forwarding (to B) as much as it receives (from A), and (2) by placing its trust in A to truthfully reveal K_A. Note that n need not place any trust in B; if B reports that the data it received is incorrect, then A will not reveal K_A to either B or n, so it is in B's best interest to truthfully report in step 6. Friedman and Resnick [9] observe that there is a trade-off between a system's barrier of entry (i.e., how easy it is to bootstrap) and the long-term participants' protection against free-riders with cheap pseudonyms. Algorithm 5 addresses this trade-off reasonably: n's barrier of entry is low (placing trust in A), and n cannot free-ride.

One could envision using weak keys, so that even if A does not reveal K_A, n can find K_A in a reasonable amount of time (though more time than it would take for A to have simply sent K_A).

10. CONCLUSION

We have formalized some of the debate on incentives in BitTorrent by focusing on two of its main components: the unchoking algorithm and the piece revelation strategy. Within a game theoretic model, we have shown that BitTorrent does not use tit-for-tat, and we have proposed an auction-based model that we find to be more accurate. When viewed as an auction, it becomes clear that BitTorrent's current unchoking algorithm does not yield the fairness and robustness guarantees desired from such a system. With the goal of "the more you give the more you get," we have investigated the use of a proportional share mechanism as a replacement to BitTorrent's unchoker, and shown that it achieves fairness and robustness without any wire-line modifications to the BitTorrent protocol.

Our auction-based model sheds light on a new class of strategic manipulation: under-reporting what pieces a peer has to its neighbors. We have demonstrated that reactively revealing only enough to keep neighbors interested can result in prolonged interest and faster download times. Lastly, we have proposed a new bootstrapping mechanism with the goal of replacing BitTorrent's optimistic unchoking in favor of an approach that encourages peers to contribute to the system as soon as they join.

There remain many interesting areas of future work. The two components we considered—piece rarity and unchoking—are, in this paper, treated orthogonally: Is there a model that unifies the two? The bootstrapping mechanism we propose is intended to be used by new peers, but is there incentive for an existing peer to use it, for instance, to obtain rare blocks from peers who are otherwise uninterested? Our results in this paper focus on incentives *within* a swarm, and not *between* swarms. The problem of *seeder promotion*—providing incentives to peers who have completed downloading the file to seed the file—is very important. We believe that the mechanisms we have presented can complement future solutions to seeder promotion.

Acknowledgments

We thank our shepherd Alex Snoeren, John Douceur, and the anonymous reviewers for their helpful comments. This work was supported in part by NSF Awards CNS 0626629 and ITR 0426683 and MIPS grant 3808.

11. REFERENCES

[1] R. Axelrod. *Evolution of Cooperation*. Basic Books, New York, 1984.
[2] BitTorrent. http://www.bittorrent.com/.
[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in a cooperative environment. In *ACM SOSP*, 2003.
[4] B. Cohen. Blog entry regarding avalanche. Online: http://bramcohen.livejournal.com/20140.html?thread=226988.
[5] B. Cohen. Incentives build robustness in BitTorrent. In *P2PEcon*, 2003.
[6] J. Douceur. The Sybil Attack. In *IPTPS*, 2002.
[7] C. Dwork, M. Naor, and H. Wee. Pebbling and proofs of work. In *CRYPTO*, 2005.
[8] M. Feldman, K. Lai, and L. Zhang. A price-anticipating resource allocation mechanism for distributed shared clusters. In *ACM EC*, 2005.
[9] E. J. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics & Management Strategy*, 10(2):173–199, June 2001.
[10] P. Garbacki, D. H. Epema, and M. van Steen. An amortized tit-for-tat protocol for exchanging bandwidth instead of content in P2P networks. In *SASO*, 2007.
[11] S. Jun and M. Ahamad. Incentives in BitTorrent induce free riding. In *P2PEcon*, 2005.
[12] K. Lai, L. Rasmusson, E. Adar, S. Sorkin, L. Zhang, and B. A. Huberman. Tycoon: an implemention of a distributed market-based resource allocation system. *Multiagent and Grid Systems*, 1(3):169–182, Aug. 2005.
[13] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *IMC*, 2006.
[14] R. LeMay. BitTorrent creator slams Microsoft's methods. *ZDNet Australia*, June 2005.
[15] D. Levin, R. Sherwood, and B. Bhattacharjee. Fair file swarming with FOX. In *IPTPS*, 2006.
[16] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting BitTorrent for fun (but not profit). In *IPTPS*, 2006.
[17] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap. In *HotNets*, 2006.
[18] L. Massoulié and M. Vojnović. Coupon Replication Systems. In *ACM SIGMETRICS*, 2005.
[19] G. Neglia, G. L. Presti, H. Zhang, and D. Towsley. A network formation game approach to study BitTorrent tit-for-tat. In *NET-COOP*, 2007.
[20] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *IPTPS*, 2003.
[21] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-compatible peer-to-peer multicast. In *2nd Workshop on the Economics of Peer-to-Peer Systems*, Cambridge, Massachusetts, June 2004.
[22] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *NSDI*, 2007.
[23] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM*, 2004.
[24] V. Rai, S. Sivasubramanian, S. Bhulai, P. Garbacki, and M. van Steen. A multiphased approach for modeling and analysis of the BitTorrent protocol. In *ICDCS*, 2007.
[25] A. Ramachandran, A. D. Sarma, and N. Feamster. BitStore: An incentive-compatible solution for blocked downloads in Bittorrent.
[26] J. Shneidman, D. C. Parkes, and L. Massoulié. Faithfulness in Internet algorithms. In *PINS*, 2004.
[27] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent networks with the large view exploit. In *IPTPS*, 2007.
[28] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *USENIX*, 2007.
[29] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A secure economic framework for P2P resource sharing. In *P2PEcon*, 2003.
[30] F. Wu and L. Zhang. Proportional response dynamics leads to market equilibrium. In *ACM STOC*, 2007.