

Team Incentives in BitTorrent Systems

UCLA Computer Science Department Technical Report 09-0002

Rafit Izhak-Ratzin, Nikitas Liogkas, Rupak Majumdar
 {rafiti, nikitas, rupak}@cs.ucla.edu

Abstract—Although the popular BitTorrent protocol strives to limit free-riding via its tit-for-tat incentives, recent research efforts have shown that it does not strictly enforce fairness. Free-riding opportunities indeed exist, mainly via optimistic unchokes, a BitTorrent mechanism that facilitates the continuous discovery of better peers to interact with. Our results in this work also show that increasing numbers of free-riders can considerably hurt the performance of compliant peers.

In an effort to address this problem, this paper proposes a BitTorrent-like protocol that dynamically organizes peers of similar upload bandwidth in *teams*— groups of peers collaborating for mutual benefit. Team members mostly satisfy their data download needs inside their team and only perform optimistic unchokes when absolutely necessary. We show that, as a result, the team protocol improves peer performance via explicit cooperation within teams. At the same time, it limits bandwidth spent on optimistic unchokes, thereby rendering the system more robust against free-riders.

We have modified an existing BitTorrent implementation to implement the team protocol and have evaluated its impact by running real experiments on a controlled PlanetLab testbed. Our results show that the protocol enables slightly improved performance for compliant peers, while free-riders are unable to sustain high download rates, as compared to regular BitTorrent. In addition, we observe a higher degree of robustness: increasing numbers of free-riders in the system have a significantly lower negative impact on the performance of contributing peers.

I. INTRODUCTION

The popular BitTorrent (BT) protocol [5] for peer-to-peer content distribution strives to ensure fairness: clients who do not contribute data to the system should not be able to achieve high download throughput. However, it has been recently demonstrated (e.g., [11], [12], [16], [20]) that BitTorrent does not strictly enforce fairness, and that free-riding opportunities indeed exist. Such free-riders might significantly hurt the performance of compliant peers, a hypothesis validated by our experimental results.

One major way for a peer to obtain data without uploading to others is via so-called *optimistic unchokes*, a BitTorrent mechanism that facilitates the continuous discovery of better peers to interact with. Despite this shortcoming, optimistic unchokes have been shown to greatly contribute to BitTorrent’s robustness [11], making them difficult to replace.

This paper proposes exactly such an alternative mechanism that mostly obviates the need for optimistic unchokes in BitTorrent, while improving fairness and system robustness. Our approach advocates dynamically organizing peers of similar upload bandwidth in *teams*— groups of peers collaborating for mutual benefit. Team members mostly satisfy their data download needs within their team and only perform optimistic unchokes when absolutely necessary.

We have modified an existing BitTorrent implementation to incorporate our team protocol and evaluated its impact on contributing and free-riding peers alike, by running real experiments with different configurations on a controlled PlanetLab testbed. These include setups with peer upload capacities derived from measurements of real BitTorrent swarms [16]. Our results demonstrate that free-riders are able to attain significantly lower download rates in the presence of teams, as compared to the original protocol, while at the same time the performance of contributing peers improves slightly. In particular, we show that the team protocol exhibits the following properties.

- 1) it provides incentives for contributing peers to join a team by rewarding them with improved download rates compared to being independent.
- 2) it discourages free-riding by limiting the number of optimistic unchokes, thereby directly hurting free-rider performance.
- 3) it results in a more robust system: increasing numbers of free-riders have a significantly lower negative impact on the performance of contributing peers, as compared to regular BitTorrent.

We also justify our protocol design and parameter choices by developing an analytical model that describes peer interactions in the presence of teams. Based on this model and game theory, we prove that the dominant strategy of a rational peer in a team-enhanced BitTorrent system is to be a contributing team member, thus providing a clear incentive to join teams.

The rest of this paper is organized as follows. Section II provides a brief description of BitTorrent. Section III then describes the design of the team protocol, while Section IV presents the analytical model that has guided this design. We discuss some implementation details in Section V, and then present the results of our PlanetLab experiments in Section VI. Lastly, Section VII sets our approach in the context of related work, and Section VIII concludes.

II. BITTORRENT OVERVIEW

BitTorrent is a popular peer-to-peer content distribution protocol that has been shown to scale well with the number of participating clients. Prior to distribution, the content is divided into multiple *pieces*, while each piece is further divided into multiple *subpieces*. A *metainfo file* containing information necessary for initiating the download process is then created by the content provider. This information includes SHA-1 hashes for each piece—which are used to verify received data—and the address of the *tracker*, a centralized component that facilitates peer discovery.

In order to join a *torrent* (or *swarm*)—the collection of peers participating in the download of a particular content—a client retrieves the metainfo file out of band, usually from a Web site. It then contacts the tracker who responds with a *peer set* of randomly selected peers currently online. These might include both *seeds*, who already have the entire content and are sharing it with others, and *leechers*, who are still in the process of downloading. The new client can then start contacting peers in this set and request data.

Most clients nowadays implement a *rarest-first* policy for piece requests: they first ask for those pieces that exist at the smallest number of peers in their peer set. Although peers always just exchange subpieces with each other, they only make data available in the form of complete pieces: after downloading all subpieces of a piece, a peer notifies all others in its peer set using a *have* message. Peers are additionally able to determine which pieces others have based on a *bitfield* message exchanged upon the establishment of new connections, containing a bitmap denoting piece possession.

Regarding who to exchange data with, leechers independently make such decisions by executing the *choking algorithm*, which gives preference to those who upload data to the given leecher at the highest rate (this is sometimes referred to as *tit-for-tat*). In particular, once per *rechoke period*, typically every ten seconds, a leecher considers the receiving data rates from all leechers in its peer set. It then picks out the fastest ones and only uploads to those for the duration of the period. In BitTorrent parlance, a peer *unchokes* the fastest uploaders via a *regular unchoke* and chokes all the rest. The number of peers to unchoke (*unchoke slots*) depends on the implementation; it may be fixed or it might depend on the available upload bandwidth.

Seeds, who do not need to download any pieces, follow a different unchoke strategy. Some implementations unchoke those leechers that *download* data at the highest rates, in order to better utilize seed capacity. Others, including the one we use for our experiments, unchoke leechers in a round-robin manner striving to distribute data uniformly.

In addition to regular unchokes, one leecher is periodically selected at random for an *optimistic unchoke*. This enables the continuous discovery of better peers to interact with, and also enables new leechers, who do not have any pieces yet, to download some data to start trading with others. Optimistic unchokes are typically rotated every three rechoke periods to allow time for leechers to demonstrate cooperative behavior.

III. DESIGN

We now present the design of the team protocol, which augments BitTorrent with the notion of *teams*—groups of peers with similar upload bandwidth collaborating for mutual benefit. The main goal of the protocol is to improve peer performance via explicit cooperation, in addition to the tit-for-tat choking algorithm already employed by BitTorrent. At the same time, it aims to limit bandwidth spent on random optimistic unchokes, a major opportunity for free-riding [20].

To achieve these goals, we design the team protocol to exhibit the following properties.

- it enables a peer to find a team of other peers with similar upload bandwidth, and provides that peer with a clear incentive to join the team, by rewarding it with improved download rates compared to being independent.
- it discourages free-riding by significantly limiting the number of optimistic unchokes team members perform, thereby directly hurting free-riders' performance.
- it enables team members to collectively detect and punish free-riders inside their team.

The protocol comprises three main parts: the process by which teams are formed, the algorithm peers use to decide who to unchoke, and the detection and punishment of free-riders. We describe these in turn.

A. Team Formation

Team formation is centrally coordinated by the tracker, who decides how to allocate peers into teams based on their available upload bandwidth. It makes this decision based on the following procedure. Every team has to satisfy the condition $\frac{U_{max}}{U_{min}} \leq (1+x)$, where the *range factor* parameter $x \in [0, 1]$ defines the allowable range of upload bandwidth for peers belonging to the given team; U_{max} and U_{min} are the upload bandwidths of the fastest and slowest team member respectively. Thus, to assign a peer P with upload bandwidth U_P to a team, the tracker makes sure that the following two *range factor conditions* are satisfied:

$$U_P \leq U_{min} \cdot (1+x), \text{ and } U_P \geq \frac{U_{max}}{(1+x)} \quad (1)$$

As a result, the slowest team member can be slower than the fastest team member by at most a factor of $1+x$. Given an appropriate selection of x , this ensures that all team members have similar upload bandwidth. There is of course a trade-off involved in the value of x . It should not be too large, as fast team members will then suffer from interacting with much slower team members. It should not be too small either, as that could cause the creation of many small-sized teams. We discuss a concrete method of calculating a good value for x in Section V-B.

We define a *full team* to be a team with T members, where T is a fixed constant that depends on the number of available unchoke slots at peers. For our PlanetLab experiments, for instance, we set this to 4, which is the default number of unchoke slots for the BitTorrent implementation we use. A *matching team* for a peer i is simply a team that is not full whose bandwidth range includes i 's upload bandwidth.

In order to thwart free-riding attempts targeted at the new mechanisms the team protocol introduces, the tracker enforces certain rules for a peer joining a team. First, it imposes a limit on the frequency of switching from one team to another, by keeping the history of team joins for each peer in the system for a short time period. In this manner, a peer can only join a fixed number of different teams within a given time interval. In addition, a peer is not allowed to join a team if it has been

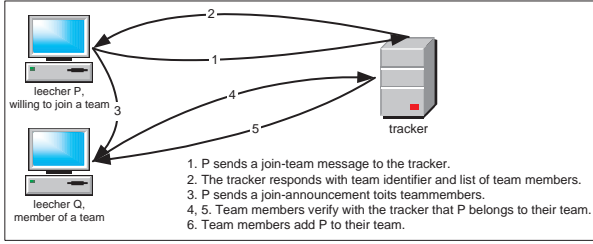


Fig. 1. Message flow for the team formation process

found non-compliant with the team protocol in the recent past, e.g., if it lied about its upload bandwidth. In such a case, i is blacklisted by the tracker and is not allowed to participate in teams. We describe in Section III-C how such non-compliant peers can be detected.

The team formation process is shown in Figure 1.

Step 1. A leecher P who wishes to join a team needs to estimate its own available upload bandwidth. This can be done either by simply consulting the upload limit set by the user, or by monitoring its own uploads during a short initial measurement period. The leecher then sends a *join-team* message to the tracker (separately from the regular initial BitTorrent message), which includes this estimated bandwidth.

Step 2. Upon receiving such a message, the tracker checks whether the given leecher is blacklisted due to misbehavior or has exceeded the limit on the frequency of switching teams. If none of these is true, the tracker looks for a matching team for the leecher. If more than one are found, the tracker selects the team that minimizes the bandwidth differences between team members, i.e., the one with $\min_T d(P, T)$, where $d(P, T) = \frac{1}{|T|} \sum_{P' \in T} |U_P - U_{P'}|$. If the tracker cannot find a matching team, it creates a new team for the leecher. It then sends back a *join-team response* that includes a unique team identifier and a list of the other team members.

Step 3. Upon receiving the response from the tracker, the leecher sends a *join-announcement* to all team members, notifying them of its wish to join their team.

Step 4. A team member receiving a join-announcement message verifies its validity by sending a *join-verification* message to the tracker with the announcing leecher's identifier (peers in BitTorrent are uniquely identified by strings chosen by peers themselves). This step prevents malicious peers from circumventing the previous protocol steps in an attempt to join an arbitrary team.

Step 5. Upon getting a join-verification message, the tracker checks if its sender and the announcing leecher contained in the message should indeed belong to the same team, and if so, sends an affirmative response.

Step 6. Upon receiving such a response, the team member adds the announcing leecher to its team. Otherwise, it ignores the join-announcement message.

This process ensures that leechers can only join teams after the approval by the tracker. It is also robust to malicious team members, who can in no way cause the acceptance of a new

member by the rest of the team.

B. Unchoking Decisions

Once in a team, a leecher has to periodically decide which peers to unchoke. In regular BitTorrent, this decision is made based on two separate strategies, the regular unchoking algorithm and optimistic unchokes. The former guides a leecher to unchoke those that upload data at the highest rates, while the latter selects peers at random, independently of their contributions. Our team protocol extends these with a third, *team unchoke* strategy, which dictates that a team member should always unchoke all members of its team. The rationale is that team members should upload to other team members whenever possible, in order to maximize team performance. At the same time, at least one regular unchoke is performed to an external peer to avoid isolating the team from the rest of the swarm; external interactions are required to introduce new content pieces into the team. Uploading via optimistic unchokes is performed only as a last resort, when a leecher has no data other team members are missing and thus cannot upload to them.

Assuming that there are n_S available unchoke slots in total, the combined unchoke decision algorithm is as follows.

Step 1 (team unchokes). A leecher who belongs to a team strives to satisfy as many of its team members as possible. We denote the number of unchoke slots reserved for team members as a function of time with $n_T(t)$, where $0 \leq n_T(t) \leq n_S - 1$, since that a leecher always reserves at least one slot for regular unchokes to external peers. Note that the team size T then becomes equal to n_S . If a peer is *independent*, i.e., not a member of a team, then $n_T(t) = 0$.

Step 2 (optimistic unchokes). A leecher performs optimistic unchokes to external peers with probability

$$P_T(t) = P_0 \frac{n_S - n_T(t) - 1}{n_S - 1} \quad (2)$$

where P_0 is the probability for an optimistic unchoke in regular BitTorrent, which depends on the implementation. We denote with $n_O(t)$ the number of slots used for optimistic unchokes as a function of time.

Step 3 (Regular unchokes). For the rest of the unchoke slots ($n_S - n_T(t) - n_O(t)$), a leecher selects the peers with the highest uploading rates to itself that were not selected already.

Note that the probability P_T of independent leechers to perform optimistic unchokes is the same in the team protocol as in regular BitTorrent (for $n_T = 0$, $P_T = P_0$). However, for team members, this probability decreases as more team members are served (n_T increases). Those who can serve all their team members at the same time perform no optimistic unchokes at all (for $n_T = n_S - 1$, $P_T = 0$).

Based on the design of this unchoking algorithm, we expect it to have two effects.

- 1) team member performance should be improved, as there are now peers who make it their priority to serve them.
- 2) the total number of optimistic unchokes in the system should be reduced, thereby directly hurting free-riders who critically depend on such unchokes to obtain data.

Our experimental evaluation results bear out both these hypotheses. In a sense, *teams mostly replace BitTorrent's optimistic unchoke mechanism* as the means of discovering better partners. Having the guarantee of other leechers with similar upload bandwidth willing to upload to them, team members no longer need to explore the network as much.

C. Detection and Punishment

So far we have assumed that all peers follow the team protocol in a compliant manner. We now describe mechanisms that can be used to detect and punish team members who attempt to circumvent the protocol for their benefit. A compliant team member can periodically execute the following steps to verify compliant behavior by other team members.

Step 1 (self upload examination). Every team member keeps track of the total amount of data it downloaded from and uploaded to all other team members. We denote these with $D_i(t)$ and $U_i(t)$ respectively for a team member i at time t . Every team member then checks if it is providing satisfactory service to others by verifying that the condition $\frac{D_i(t)}{U_i(t)} \leq (1+x)$ is true for the majority of its team members (this condition follows from Equation 1). If the condition is false, a compliant leecher sends a *leave-team* message to its team, and a *switch-team* message to the tracker to switch to another team that is more appropriate for its upload bandwidth. This message includes the leecher's upload bandwidth, like the initial join-team message.

Step 2 (team upload examination). Moreover, every team member examines other members' upload rates to verify they are not lower than expected (every team member i has to satisfy the condition $\frac{U_i(t)}{D_i(t)} \leq (1+x)$). If this is not the case, the team member executes the punishment process, described below, to attempt to expel peer i from its team.

Step 3 (team satisfaction examination). Even without free-riders in a team, a leecher might still not be satisfied with the team's performance. This can happen either because its available upload bandwidth has increased since joining the team, or because other team members did not concur to punish a peer that did not satisfy step 2. Thus, the peer can also determine its satisfaction with the team as a whole by checking whether $\frac{\sum_{i=1}^{T-1} U_i(t)}{\sum_{i=1}^{T-1} D_i(t)} \leq (1+x)$. If this is false, the peer can send a switch-team message to the tracker to switch teams.

Note that the execution intervals for steps 2 and 3 should be longer than that for step 1, to allow team members to leave their team if they cannot upload fast enough before being blacklisted. However, it should not be too long, so as to allow free-riders to abuse team unchokes before being detected. Determining the optimal values for these parameters is the subject of ongoing work.

Punishment process. The punishment process can be invoked by any team member j for another team member i if j is not satisfied with the upload rate it is getting from i , and the team is not already in the process of deciding whether to expel i . To initiate the process, j sends *unsatisfied* messages about i to its team members (excluding i). Upon receiving these messages, peers execute the team upload examination on i , and, if they

also find it to be unsatisfactory, they too send an unsatisfied message to all other team members (excluding i).

Once a team member receives unsatisfied messages from the majority of its team, it sends an unsatisfied message to the tracker, who keeps count of how many such messages it receives. If this count exceeds half of the size of the team, the tracker proceeds to remove peer i from the team and adds it to a blacklist. This prevents it from joining another team for a given period of time. Note that a single malicious team member can in no way cause another team member to be blacklisted. Only a collusion of at least half the team members can achieve that, in which case compliant leechers would arguably leave the team on their own due to the team satisfaction examination.

When blacklisting a team member, the tracker also replies back to the unsatisfied team members with a *punish* message, which they then proceed to forward to the rest of their team, including peer i , effectively notifying it of its expulsion. This process of forwarding messages is necessary, because, unlike team members, the tracker does not maintain open connections with all leechers in the team. The compliant team members remove peer i from their local team list upon receiving a punish message from at least half the team or from the tracker. Since teams are typically small, the communication overhead of the punishment procedure should not be prohibitive, but we are still in the process of evaluating this.

IV. ANALYTICAL MODEL

We have validated the protocol design presented in the previous section both via analysis and by running real experiments on a controlled testbed. The analytical model we have developed describes peer interactions in the presence of teams as a repeated game with rational players. This section presents this model and demonstrates that the dominant strategy of a leecher in a team-aware BitTorrent system is to be a contributing team member. The next section presents our experimental results.

A. Optimistic Unchoke Probability

We assume that all teams in the system have the same size T (enforced by the tracker), and that all contributing leechers have the same total number of unchoke slots n_S (which is a good approximation for most implementations). In addition, a leecher belonging to a team with at least two members is considered a team member, otherwise it is considered an independent leecher or free-rider, depending on its strategy choice. We define $n_{T_{max}}$ to be the maximum number of unchoke slots that can be taken up by a leecher's team members, and, as we explained, $n_{T_{max}} = n_S - 1 = T - 1$. Given the unchoking decision algorithm (Section III-B), if all contributing leechers belong to teams and all of them can serve all their team members at time t , then the probability for an optimistic unchoke by any peer in the system at that time will be 0. As a result, free-riders will not be able to receive any data via optimistic unchokes and will only be able to download data from seeds, thereby achieving significantly lower download

x	bandwidth range factor
T_i	size of the team that leecher i belongs to
n_S	total number of unchoke slots
$n_T(t)$	number of unchoke slots taken up by team members at time t ($n_T \leq n_S - 1$)
$n_O(t)$	number of unchoke slots used for opt. unchokes
P_0	probability for an opt. unchoke in original BitTorrent
P_T	probability for an opt. unchoke by a team member
K	total number of pieces of the content
$P_{hasall}(i, j)$	probability of leecher i havinf all the pieces that leecher j has
U_i	utility function of peer i
u_i	upload capacity of peer i
R	punishment duration (in rechoke periods)
δ	future discount factor for punishment

TABLE I
MODEL NOTATION

rates. We will now prove that the probability for this to occur in a team-aware BitTorrent system is very high.

Similarly to Qiu *et al.* [17], we assume that:

A1. If K is the total number of content pieces, the number of pieces each leecher has will be uniformly distributed in $[0, \dots, K - 1]$, due to the random arrival of peers to the network.
A2. Let k_i denote a random variable representing the number of pieces a given leecher i has. These k_i pieces would then be chosen randomly from the set of all content pieces. As shown by Dale *et al.* [6], this is a reasonable assumption, as BitTorrent peers typically follow a rarest-first piece selection policy, which is effective in replicating pieces uniformly.

Theorem IV.1. *Under assumptions A1 and A2, a team member's probability of performing an optimistic unchoke $P_T \rightarrow 0$ as $K \rightarrow \infty$, with high probability.*

Proof: The probability in question is given by Equation 2, with P_0 and n_S given constants. In order then to show that $P_T \rightarrow 0$ with high probability, we need to show that $n_T \rightarrow n_S - 1 = n_{T_{max}}$, with high probability. Let us define $P_{hasall}(i, j)$ the probability that team member i has all the content pieces that team member j has, so that j cannot upload anything of interest to i , and thus might have to perform an optimistic unchoke instead. Qiu *et al.* [17] have already shown that $P_{hasall}(i, j) \approx \frac{\log K}{K}$, under the assumptions stated above. Thus, the probability that a team member is able to upload to all other team members at time t is $P(n_T = n_{T_{max}}) \approx (1 - \frac{\log K}{K})^{n_{T_{max}}}$. A team member performs an optimistic unchoke when it cannot serve all other team members simultaneously. This occurs with probability

$$\begin{aligned}
P_T &= P(n_T \neq n_{T_{max}}) \cdot P_0 \frac{n_S - \bar{n}_T - 1}{n_S - 1} \\
&\leq P(n_T \neq n_{T_{max}}) \cdot P_0 \\
&\approx (1 - (1 - \frac{\log K}{K})^{n_{T_{max}}}) \cdot P_0
\end{aligned} \tag{3}$$

where \bar{n}_T is the average value of n_T . Therefore, as the number of content pieces increases, $\lim_{K \rightarrow \infty} P_T \rightarrow 0$. ■

Note that, even for smaller values of K , the probability P_T for a team member to perform an optimistic unchoke is still

close to 0. For instance, a typical piece size in BitTorrent is 256 kB and a typical value for n_S is 4, which would make $n_{T_{max}} = 3$. In addition, P_0 is typically 0.25. Given these parameters, a 100 MB file will consist of 400 pieces, giving us $P_T \leq 0.015$. Similarly, for a 1 GB file, $P_T \leq 0.002$.

B. The Game

Having proven that the probability of optimistic unchokes by team members is low, we now turn to incentives.

Definition. We describe the process of distributing a given content as an infinitely repeated game with rational participants and punishment [14]. The players (leechers) repeatedly engage in rounds and choose their actions simultaneously. A player can follow one of the following four strategies.

- 1) **an independent peer:** a contributing leecher who does not belong to a team.
- 2) **a free-rider:** an independent peer who does not upload data to others.
- 3) **an honest team member:** a player who joins a team and follows the protocol rules, e.g., reports its correct upload bandwidth to the tracker.
- 4) **a dishonest team member:** a player who joins a team, but does not follow the protocol rules, e.g., lies about its upload capacity.

Leecher Utility Function. We define the utility function of leecher i to be the average download rate from all other leechers, $U_i = \bar{d}_i$. Note here that there should also be a term expressing downloads from the seeds. As this term does not depend on the existence of teams, we can safely ignore it for the rest of the analysis.

Fan *et al.* [8] have already expressed the leecher utility function for the original BitTorrent protocol as a combination of the regular (selective, as they call it) and the optimistic (non-discriminative) unchoke policy. Our protocol adds yet another term due to the team unchoke policy. Thus, the utility function of a leecher i will be

$$\bar{U}_i = \bar{d}_i(\text{regular}) + \bar{d}_i(\text{optimistic}) + \bar{d}_i(\text{team}) \tag{4}$$

Due to the way team formation works, and given a credible punishment mechanism inside a team, we can assume that all team members have a similar uploading capacity u_i , and that they distribute this capacity uniformly over all their unchoke slots. In addition, we assume that a leecher's download bandwidth is greater than or equal to its upload bandwidth, which is typically the case in real torrents. Each team member will then be served with approximate probability $(1 - \frac{\log K}{K})$ by each of the other team members ($T - 1$ in total) and will enjoy a download rate of $\frac{u_i}{n_S}$ from each team member. As a result, the team policy term will be

$$\bar{d}_i(\text{team}) \approx (1 - \frac{\log K}{K}) \cdot (T - 1) \cdot \frac{u_i}{n_S} \tag{5}$$

Let us define $P_{dt} = (1 - \frac{\log K}{K}) \cdot (T - 1) \cdot \frac{1}{n_S}$. P_{dt} not only describes the download rate a leecher enjoys from its team members, but also represents the percentage of upload bandwidth it spends on those team members.

Fan *et al.* [8] have shown that, for regular unchokes, the download bandwidth of a leecher is approximately equal to its upload bandwidth, i.e., $d_i \approx u_i$. The percentage of the upload bandwidth dedicated to regular unchokes is $1 - P_{dt} - P_T$. Therefore, the regular unchoke term will be

$$\bar{d}_i(\text{regular}) \approx (1 - P_{dt} - P_T) \cdot u_i \quad (6)$$

From Equations 4, 5, and 6, the utility function of team member i will therefore be

$$U_i \approx (1 - P_T) \cdot u_i + \bar{d}_i(\text{optimistic}). \quad (7)$$

On the other hand, for a free-rider who does not upload anything and does not belong to a team, $U_{FR} = \bar{d}_{FR}(\text{optimistic})$. Also, the utility function of an independent leecher will be $U_{IL} \approx (1 - P_0) \cdot u_i + \bar{d}_{IL}(\text{optimistic})$. Note that the optimistic unchoke yield $\bar{d}_i(\text{optimistic})$ is not sensitive to a peer's strategy, but is based on random selection among all peers.

The detection and punishment mechanism presented in the previous section can be used to prevent free-riding inside a team. More specifically, punishment was defined as expelling a leecher from a team and preventing it from joining any other team for a fixed number of R rounds (rechoke periods). R should be long enough to decrease a free-rider's utility and therefore make the punishment credible as a threat. In particular, the utility of a dishonest team member should always be lower than that of a contributor. To examine whether this is true in our model, we define the following variables. d_T is the download rate of a leecher due to its participation in a team during a single round, $U_i[0..R]$ is the cumulative utility from time 0 to time R , and $\delta \in [0, 1]$ is a *future discount factor*, which expresses how important future contributions from others are to the utility of a leecher. (e.g., the closer δ is to 0 the less future contributions are valued). In our case, δ should be close to 1, as every piece of the content carries the same importance. Therefore, based on the folk theorem [14], in order for the punishment to be credible, the following equation should be satisfied.

$$\begin{aligned} & U_i(\text{dishonest team member})[0..R] \\ &= \frac{1}{R+1} (d_T + \sum_{j=1}^R \delta^j ((1 - P_0) \cdot u_i + \bar{d}_i(\text{optimistic}))) \\ &< \frac{1}{R+1} \sum_{j=0}^R \delta^j ((1 - P_T) \cdot u_i + \bar{d}_i(\text{optimistic})) \\ &= U_i(\text{honest team member})[0..R] \end{aligned} \quad (8)$$

So we see that a credible punishment will always lead to a lower utility for a dishonest team member compared to an honest contributing peer.

Nash Equilibrium.

Theorem IV.2. *The dominant strategy of a rational leecher in a team-aware BitTorrent system is to be an honest team*

member. Therefore, there is a unique Nash Equilibrium (NE) when all team members follow the honest strategy.

Proof: In order to prove this, we need to show that the utility of an honest team member is always higher than with any other strategy. From Equation 3 we can see that for values of $K \geq 3$, $P_T < P_O$ with high probability. Therefore, the utility of an independent leecher i ($U_i(IL) = (1 - P_0) \cdot u_i + \bar{d}_i(\text{optimistic})$) will always be lower than the utility of an honest team member ($U_i(\text{honest}) = (1 - P_T) \cdot u_i + \bar{d}_i(\text{optimistic})$).

Moreover, we assume that the punishment mechanism constitutes a credible threat and must therefore satisfy Equation 8. Therefore, the utility of a dishonest team member will also be lower than the utility of an honest one.

Finally, the utility of a free-rider will be $U_i(FR) \leq \bar{d}_i(\text{optimistic}) + d_T$, since a free-rider might try to join a team. This is again lower than the utility of an honest team member, since $(1 - P_T) \cdot u_i > d_T$ due to the satisfaction condition of Equation 8. As a result, being an honest team member will afford a leecher a higher utility than being a free-rider.

We can thus conclude that the dominant strategy in the game is to be an honest team member, and that there is a unique NE when all leechers follow the honest team member strategy. ■

V. IMPLEMENTATION

The analytical model gives us confidence that teams will indeed limit free-riding in a real BitTorrent system. To examine this claim, we have modified existing open-source BitTorrent client and tracker implementations to realize the team protocol. This section discusses interesting aspects of our prototypes and justifies our protocol parameter choices.

A. Client

Our team-aware BitTorrent client is based on Enhanced CTorrent, version 3.2 [7]. We added code to enable the client to run in *team mode*, in which it follows the honest team member strategy. The client can also run in *regular mode*, in which it simply behaves as an independent peer, or in *free-riding mode*, where it remains independent and does not upload any data at all. We also added functionality for a team member to maintain state about its team, including the team identifier and other members' uploading history. We have also extended the BitTorrent peer protocol with the following messages.

- *join-team request*, sent by a leecher to the tracker in order to join a team (contains the leecher's upload bandwidth)
- *join-team response*, sent back by the tracker (contains the team identifier and the list of team members)
- *join-team announcement*, sent by a prospective leecher to team members
- *join-verification request*, sent by a team member to the tracker to verify the joining of a prospective leecher (contains the leecher's identifier)
- *join-verification response*, sent by the tracker back to a team member verifying the prospective leecher is allowed to join the team (contains the leecher's identifier)

All in all, we added 988 non-comment lines of code to Enhanced CTorrent, while we did not need to remove or modify any of the existing code.

For our experiments presented in the next section, we use the default client parameters, unless otherwise specified. Enhanced CTorrent uses a 10-second rechoke period, and optimistic unchokes are rotated every three rechoke periods. The number of unchoke slots n_S has a minimum value of 4, from which one is always devoted to optimistic unchokes in regular mode. When running in team mode, the number of optimistic unchokes is determined by our unchoking decision algorithm. In particular, a single optimistic unchoke is performed every three rechoke periods with probability given by Equation 2.

Although fully functional, our prototype currently has certain limitations. Most notably, the detection and punishment mechanism described in Section III-C is still under development. This mechanism is designed to limit the benefit free-riders can get from joining a team by selecting appropriate parameter values, such as the execution intervals for each step and the period a peer is blacklisted. Determining the optimal values for these parameters is the subject of ongoing work.

B. Tracker

Our tracker is based on BNBT EasyTracker, version 7.7r3 [2]. We augmented the message classifier inside the tracker to support the new protocol messages. We also added functionality to manage the information for all teams in the system, which includes the team identifier, list of team members, and additional data, such as members' upload bandwidth. All in all, we added 522 non-comment lines of code and did not remove or modify any existing code.

Since Enhanced CTorrent uses a minimum of 4 unchoke slots, the tracker sets the maximum team size T to 4 as well, for all teams in the system. To select the value of the range factor x , we use an empirical peer upload capacity distribution derived from measurements of real BitTorrent swarms [16]. Given the size of the network N , the maximum team size, and the upload capacity distribution, we can calculate x based on the desired p_f , the percentage of peers that will be members of full teams (for multiple-tracker swarms, the trackers could periodically communicate with each other to collaboratively calculate the size of the network).

For the team protocol to work well, the tracker should aim to keep p_f as high as possible. For instance, for $T = 4$, and $N = 500$, by setting $x = 0.05$, we get $p_f = 93\%$. for $N=1000$, we get $p_f = 94\%$ by setting $x = 0.03$, while for $N=200$, we get $p_f = 93\%$ by setting $x = 0.09$. In our experiments with 51 peers, we set $x = 0.1$, which results in $p_f = 90\%$. Therefore, within a given team, the slowest member was at most slower by 10% than the fastest member.

VI. EXPERIMENTAL EVALUATION

We now turn to real experiments to validate the properties of the team protocol. We first describe our experimental methodology and then discuss the results.

A. Methodology

We ran all our experiments on the PlanetLab experimental platform [1], utilizing nodes spread across the globe. These nodes are typically not behind NATs or firewalls, so these effects are not represented in our results, although we do not believe the conclusions we draw are predicated on that fact. We always execute all runs of an experiment consecutively in time on the same set of machines.

We ran extensive sets of experiments with content of different sizes, different numbers of leechers (24, 48), different numbers of free-riders (4, 8, 16, and 24), different leecher upload rates (20kB/s, 50kB/s, and 100kB/s), and different percentages of peers belonging to teams. We report here representative results for a 71 MB file divided into 284 pieces, each of size 256 kB (making $K = 284$), which demonstrate our main conclusions. Unless otherwise specified, we use the default CTorrent client parameters.

All leechers start the download process at the same time, emulating a flash crowd scenario, and leave the network once they have downloaded the entire content. The initial seeds stay connected for the duration of the experiment. The available bandwidth of PlanetLab nodes is relatively high for typical BitTorrent peers. We define upload limits on the leechers and seeds to model more realistic scenarios, but do not define any download limits to observe the effect of our protocol on peers' download rates.

B. Results

We look at comparative results for contributing leechers' and free-riders' performance, in order to validate the hypotheses stated in Section I. We also examine system robustness when increasing the number of free-riders. Finally, we additionally test our protocol using an empirical BitTorrent upload capacity distribution [16].

Contributor Performance. Figure 2 shows leechers' download completion time both in regular BitTorrent and in a team-enhanced system where all leechers join teams of size 4. This experiment involves 50 PlanetLab nodes, 2 seeds and 48 leechers, 8 of whom are free-riders. All contributing leechers have 50 kB/s upload bandwidth. For each leecher, we draw two boxplots [18] corresponding to the two scenarios. The top and bottom of each box represent the 75th and 25th percentile download rates over all 5 runs of the experiment. The marker inside the box is the median, while the vertical lines extending above and below represent the maximum and minimum values respectively (within the high and low ranges extending 1.5 times the box height from the box boundaries). Potential outliers are marked individually.

We observe that the *download time for all team members improves by 10% – 26%*, as measured by the median. Downloading from team members consistently enables a leecher to maintain high download rates. Looking at the amount of uploaded data for each contributor leecher in the two scenarios, we observe that 35 of the contributors upload less when they are in teams. In particular, 32 of them upload at least 10%

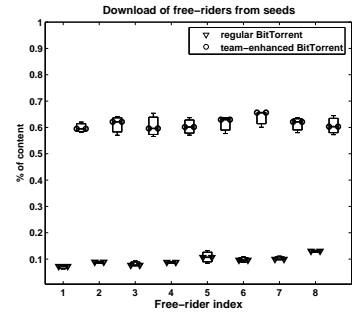
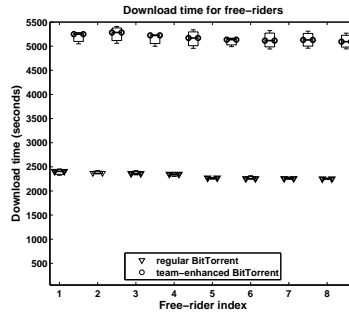
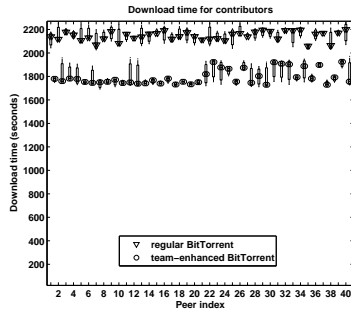


Fig. 2. Download completion time for contributors Fig. 3. Download completion time for free-riders Fig. 4. Content free-riders download from seeds

less data. This is because team members perform significantly less optimistic unchokes than in regular BitTorrent.

We also performed experiments where only a fraction of contributing leechers belong to teams and the rest are independent. As expected, team members in these experiments did better when the percentage of independent peers increased, while independent peers did slightly worse. This is because team members spend their upload bandwidth mainly on other team members, while independent peers receive less data via optimistic unchokes.

In summary, we see that *the team protocol provides a clear incentive for contributing leechers to join teams*, since a team member achieves improved performance compared to an independent one, while at the same limiting the amount of data it needs to upload to others. In addition, this improvement would be even more pronounced with larger content, which is typical for BitTorrent swarms. Larger content would entail a higher number of pieces, which, according to Equation 3, will lead to an even lower probability of optimistic unchokes.

Free-rider Performance. Figure 3 shows the download completion times for free-riders in the same experiment, for the two scenarios. We observe that *the existence of teams slows down free-riders by more than 100%*. This is because free-riders can no longer take advantage of optimistic unchokes and thus have to depend mainly on the two seeds for downloading data. Indeed, Figure 4, which plots the percentage of content free-riders download from seeds, validates this hypothesis. This percentage increases from $\sim 10\%$ in regular BitTorrent to $\sim 60\%$ in a team-enhanced system.

Interestingly, our analytical model predicts an even higher percentage of download from seeds, closer to 100%. This is not the case here since the model only considers a system in its steady state, while, in reality, there are warm-up and endgame periods, during which contributing leechers may perform some optimistic unchokes, since they are not able to upload to all their team members at the same time.

Figure 5 explains this better by plotting the ratio of the probability of optimistic unchokes by team members in a team-enhanced system over the same probability in regular BitTorrent ($\frac{P_T}{P_0}$), as given by Equation 2, for all 10-second intervals (the rechoke period) during the download. This

value is averaged over all 5 runs for all 40 contributor leechers. In addition, the dotted line represents the percentage of contributors still present in the system at that point in time. We observe that the ratio is lower than 0.1 between the 40th and 130th time unit, indicating that very few optimistic unchokes are performed by team members in that period. On the other hand, there is clearly a higher probability for optimistic unchokes by team members during the warm-up and endgame periods.

System Robustness. *Robustness* is the degree to which a system is able to deliver good service despite the presence of free-riders. In a robust system, increasing numbers of free-riders should not be able to significantly affect the performance of contributing leechers.

Figure 6 plots the performance of contributing leechers with different numbers of free-riders, when all the contributors belong to teams, and in regular BitTorrent, when all are independent. All other experiment parameters remain the same, including the total number of leechers (48). We observe that contributors perform noticeably better ($\sim 15\%$ improvement) when they run as team members. Interestingly, the difference in performance is more pronounced for higher numbers of free-riders. Therefore, *a team-enhanced system appears to be more robust than regular BitTorrent*, especially in the presence of a high percentage of free-riders in the network.

Figure 7 examines free-rider performance. Although we expect their performance to deteriorate as their percentage increases even in regular BitTorrent, due to the decreased total upload capacity in the system, we observe that free-riders suffer considerably more in the presence of teams. For instance, when half the leechers are free-riders, they experience a 29% slowdown compared to regular BitTorrent.

Empirical upload capacity distribution. In another set of experiments we used a scaled upload capacity distribution observed in real BitTorrent systems, as presented in [16]. This distribution was based on empirical measurements of BitTorrent swarms including more than 300,000 unique IP addresses. Our experiments included 51 PlanetLab nodes: 40 contributors with upload capacities drawn from the empirical distribution, 8 free-riders, and 3 seeds with combined capacity

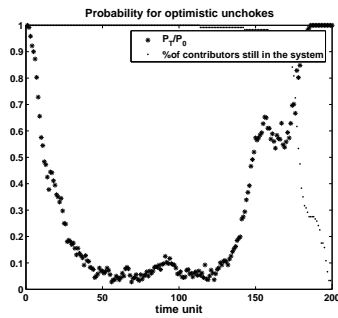


Fig. 5. $\frac{P_r}{P_0}$ probability ratio

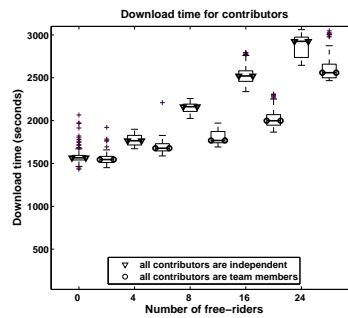


Fig. 6. Download completion time for contributors in the presence of different numbers of FRs

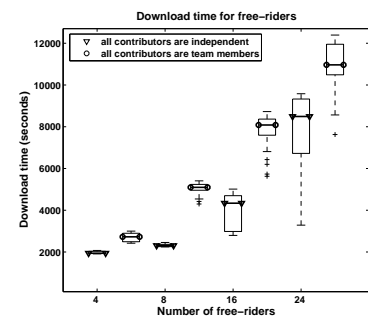


Fig. 7. Download completion time for FRs as their number increases

of 128 kB/s serving a 15 MB file. The other experiment settings remained the same. We compare the results of regular BitTorrent to a team-enhanced system. In the latter, not all teams were full due to differences in peer upload rates.

Our results (not shown here) demonstrate that download times for 76% of the team members improved by up to 33%, as measured by the median of 5 experiment runs. The majority of the team members who did not improve their download time were the slowest leechers. In regular BitTorrent these peers would significantly benefit from optimistic unchokes, since their upload rate is very slow compared to that of fast contributors and seeds. Regarding free-riders, we observed that the download time for all of them increased by up to 25%.

VII. RELATED WORK

There has been a fair amount of work on the performance and behavior of BitTorrent systems. We only mention here work that is directly related to BitTorrent's tit-for-tat incentives and identified opportunities that can lead to free-riding.

Bram Cohen, the protocol's creator, first described BitTorrent's main mechanisms and their design rationale [5]. Qiu *et al.* [17] developed a fluid analytical model of BitTorrent systems and studied the choking algorithm and its effect on peer performance. They were the first to observe that optimistic unchokes may provide a way for leechers to free-ride. More recently, Fan *et al.* [8] characterized the incentive design space for BitTorrent-like protocols and presented a model that captures the fundamental trade-off between performance and fairness. Our analytical model is inspired by their work. Additionally, Legout *et al.* [10] observed that tit-for-tat incentives tend to cause leechers to cluster together with others having similar upload bandwidth. Our protocol facilitates this explicitly by having the tracker assign such leechers to the same team.

Other researchers have looked into the feasibility of free-riding behavior, when peers attempt to circumvent protocol mechanisms to download without uploading. Shneidman *et al.* [19] were the first to pinpoint that effective free-riding in BitTorrent is feasible. They briefly described an attack to the tracker and an exploit involving leechers lying about the pieces they have. Jun *et al.* [9] argued that the choking

algorithm is not sufficient to prevent systematic free-riding and proposed a new algorithm to enforce fairness in peer data exchanges. Furthermore, Liogkas *et al.* [11] designed and implemented three exploits that allow a free-rider to achieve high download rates under specific circumstances. Locher *et al.* [12] extended those results by demonstrating that limited free-riding is feasible even in the absence of seeds.

More recently, Sirivianos *et al.* [20] evaluated an exploit based on maintaining a larger-than-normal view of the system, which affords free-riders a much higher probability of receiving data from seeds and optimistic unchokes. Piatek *et al.* [16] also observed that high-capacity peers typically provide low-capacity ones with an unfair share of the content. They proposed a new choking algorithm that reallocates upload bandwidth to maximize peer download rates. Our work directly addresses one of the most important vulnerabilities identified in the aforementioned studies, namely exploiting optimistic unchokes to download data for free.

Complementary to our approach, some work has attacked the ability of free-riders to download data from seeds without uploading in return. Locher *et al.* [13] proposed a source coding scheme to replace tit-for-tat incentives. Seeds in this scheme only upload a fixed number of content pieces to each leecher, thereby placing a hard limit on the data free-riders can obtain in this manner. Chow *et al.* [4] presented another modified seed unchoking algorithm that gives preference to leechers who are either at the beginning or the end of their download. They show that this algorithm considerably hurts free-riders' performance.

Lastly, there have been some other approaches to having peers cooperate to download content. Wong's thesis [22] presents a system based on volunteer *helper* peers, who download popular content pieces and upload them to others. However, it does not provide any incentives for the helpers themselves. The 2Fast system [15] enables a *collector* peer to task others to download data on its behalf. However, it provides no mechanism to enforce reciprocation of this favor in the future, but rather relies on social (friend) pressure instead. In addition, it only helps improve the download rate of the collector. Our protocol's goal, on the other hand, is to improve download rates for all team members.

BTSlave [3] is a BitTorrent protocol extension that also attempts to leverage a friends approach. It too suffers from the absence of a mechanism to enforce reciprocation in the future. Lastly, Wang *et al.* [21] propose to utilize helpers for improved performance, by having them download a small number of random content pieces, which they then altruistically upload to those with a small fraction of the content. Their approach does not address the issue of incentives for helpers, and also creates an additional opportunity for free-riders, who can misreport their download progress to the tracker to receive more data from helpers for free. Our protocol, on the other hand, addresses all these problems by limiting free-riding and providing a clear incentive for leechers to join teams and upload to others.

VIII. CONCLUSION

We have presented an extension to the BitTorrent protocol that mostly replaces optimistic unchokes with an alternative mechanism that is less susceptible to free-riding. Results of experiments on a controlled testbed demonstrate that free-riders can attain significantly lower download rates in the presence of teams, while the performance of contributing peers improves, as compared to the original protocol, providing a clear incentive for peers to follow the honest team member strategy. Maybe more importantly, the resulting system is more robust than regular BitTorrent: increasing numbers of free-riders in the network have comparatively much lower negative impact on the performance of compliant peers.

ACKNOWLEDGMENTS

We wish to thank Eddie Kohler, Arnaud Legout, and Michael Sirivianos for their invaluable feedback.

REFERENCES

- [1] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating System Support for Planetary-Scale Network Services," in *NSDI'04*.
- [2] BNBT EasyTracker. [Online]. Available: <http://bnbteasytracker.sourceforge.net>
- [3] BTSlave protocol page. [Online]. Available: <http://btslave.sourceforge.net>
- [4] A. L. Chow, L. Golubchik, and V. Misra, "Improving BitTorrent: A Simple Approach," in *IPTPS'08*.
- [5] B. Cohen, "Incentives Build Robustness in BitTorrent," in *P2PEcon'03*.
- [6] C. Dale and J. Liu, "A Measurement Study of Piece Population in BitTorrent," in *IEEE GLOBECOM'07*.
- [7] Enhanced CTorrent. [Online]. Available: <http://www.rahul.net/dholmes/ctorrent>
- [8] B. Fan, D.-M. Chiu, and J. C. Lui, "The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design," in *ICNP'06*.
- [9] S. Jun and M. Ahamad, "Incentives in BitTorrent Induce Free Riding," in *P2PEcon'05*.
- [10] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and Sharing Incentives in BitTorrent Systems," in *SIGMETRICS'07*.
- [11] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)," in *IPTPS'06*.
- [12] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free Riding in BitTorrent is Cheap," in *HotNets-V (2006)*.
- [13] T. Locher, S. Schmid, and R. Wattenhofer, "Rescuing Tit-for-Tat with Source Coding," in *P2P'07*.
- [14] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. MIT Press, 1994, ch. 8.
- [15] Paweł Garbacki and Alexandru Iosup and Dick Epema and Marten van Steen, "2Fast: Collaborative downloads in P2P networks," in *P2P'06*.
- [16] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?" in *NSDI'07*.
- [17] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *SIGCOMM'04*.
- [18] J. W. T. Robert McGill and W. A. Larsen, "Variations of box plots," *The American Statistician*, vol. 32, pp. 12–16, 1978.
- [19] J. Shneidman, D. C. Parkes, and L. Massoulié, "Faithfulness in Internet Algorithms," in *PINS'04*.
- [20] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in BitTorrent Networks with the Large View Exploit," in *IPTPS'07*.
- [21] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran, "On the role of helpers in peer-to-peer file download systems: design, analysis, and simulation," in *IPTPS'07*.
- [22] J. H. T. Wong, "Enhancing Collaborative Content Delivery with Helpers," MSc thesis, University of British Columbia, September 2004.