On the Robustness of BitTorrent Swarms to Greedy Peers

Damiano Carra, Giovanni Neglia, Pietro Michiardi and Francesco Albanese

Abstract—The success of BitTorrent has fostered the development of variants to its basic components. Some of the variants adopt *greedy* approaches aiming at exploiting the intrinsic altruism of the original version of BitTorrent in order to maximize the benefit of participating to a torrent.
In this work we study BitTyrant, a recently proposed strategic client. BitTyrant tries to determine the exact amount of contribution necessary to maximize its download rate by dynamically adapting and shaping the upload rate allocated to its neighbors. We evaluate in detail the various mechanisms used by BitTyrant to identify their contribution to the performance of the client.
Our findings indicate that the performance gain is due to the increased number of connections established by a BitTyrant client, rather than to its subtle uplink allocation algorithm; surprisingly, BitTyrant reveals to be altruistic and particularly efficient in disseminating the content, especially during the initial phase of the distribution process. The possible gain of a single BitTyrant client, however, disappears in the case of a widespread adoption: our results indicate a severe loss of efficiency that we analyzed in detail.

Index Terms—Peer-to-Peer, BitTorrent, Performance analysis.

1 Introduction

BitTorrent [1] is a peer-to-peer (p2p) content distribution application that has been adopted by millions of end-users [2], [3], [4]. BitTorrent (BT) has also attracted the attention of a large body of researchers that focused on its building blocks (such as the incentives, as in [5]) and its performance analysis through measurement [6], [7], [8], simulation [9], [10] and analytical [11], [12], [13] studies. These previous works indicated that the key of BitTorrent success can be substantially attributed to its scalability and its greater robustness to free-riding in comparison to previous p2p proposals.

Some recent studies [14], [15], [16] have proposed new clients, that are compliant to BitTorrent message protocol but adopt greedy strategies with the purpose of optimizing the local performance of the modified client. For example, authors in [15] designed the BitThief client, that tries to maximize the content download rate without uploading any content. Another prominent example is that of BitTyrant [16], which tries to maximize its download rate by shaping its contribution to remote peers. Note that, while BitThief client is intrinsically a free-rider, BitTyrant makes its whole upload capacity available to spread the content.

Our research interest is twofold. First, we want to evaluate the competitive advantage of greedy clients over standard ones and hence the possibility of wide adoption by the peer-to-peer community. Second, we want to investigate the possible effects that a widespread adoption could induce to the system performance. We first focus on a single client (we chose BitTyrant in this work because it merges several greedy techniques discussed in the literature) and characterize its performance gain over standard clients. We do so by isolating its key mechanisms to quantify their contribution to the improved performance. We then make the case for a gradual adoption of BitTyrant by users and discuss its implications on the whole community.

The main contributions of this work can be summarized as follows:

- We generalize the analytical model presented in [16] to identify the extent to which BitTorrent can be exploited by greedy clients. Unlike previous results discussed in [16], our findings indicate that exploiting the altruism of BitTorrent is effective only during a *short* transient regime when the system is bootstrapping;
- We study the different components of a prominent example of a greedy client, BitTyrant [16], and we evaluate to what extent each part of the proposed approach is responsible for the performance achieved; we also compare the results with the ones obtained by the mainline BitTorrent client;
- We cast light on the subtle choke algorithm used by BitTyrant: while we show its unexpectedly positive impact on system performance - especially during the startup phase of content distribution - we also point to an undesired periodic behavior that limits its performance;
- Finally, we make the case for a gradual adoption of the BitTyrant client by the mass; we show that a widespread adoption of BitTyrant clients seems

- *D. Carra is with the Computer Science Dep., University of Verona, Italy. E-mail: damiano.carra@univr.it*
- *G. Neglia is with INRIA Sophia Antipolis, Sophia Antipolis, France. E-mail: giovanni.neglia@ieee.org*
- *P. Michiardi and F. Albanese are with EURECOM, Sophia Antipolis, France.E-mail: michiard@eurecom.fr, albanese@eurecom.fr*

unrealistic, and in any case the increasing adoption of BitTyrant progressively degrade the system performance.

2 Background

In this section we briefly outline the key algorithms used by BitTorrent [1], BitTyrant [16] and BitThief [15].

BitTorrent. The BitTorrent (BT) protocol is designed for bulk data transfer. A file is divided into pieces, which can be downloaded in parallel from peers belonging to a specific *torrent*. A central entity, the *tracker*, keeps track of all peers sharing the content and provides new peers a random set of peers to connect to. The neighborhood of a peer is called the *peer set*.

A BT peer executes two key algorithms, one that is used to select pieces of the content to download (piece selection algorithm) and one that is used to select remote peers to upload data to (the *choke algorithm*). In this work we focus on the choke algorithm. With the choke algorithm, a node builds a subset of its peer set that is termed the *active set*: peers in the active set are entitled to request pieces of the content. The choke algorithm is executed every 10 seconds: all remote peers are ranked based on their upload rate and only the first k top peers are unchoked. Along with regular unchokes, every 30 seconds a peer randomly unchokes ω peers irrespectively of their rank: this technique is termed *optimistic unchoke* and allows a peer to explore its peer set and discover fast neighbors. The choke algorithm aims at maintaining a high level of *reciprocation* among peers, being that peers uploading less have less chances to be unchoked by their neighbors.

In the first version of BT - that we term **BTold** - k and ω are empirically set parameters: generally $k = 4$ and $\omega = 1$. The upload bandwidth of a peer is shared equally (beside TCP effects) among all unchoked peers; the portion of the bandwidth that each peer is able to obtain is defined as *equal-split*. In the latest version of the BT mainline protocol the choice of the parameters in the choke algorithm is different: the number of regular unchokes is determined as a function of the uplink capacity C of a peer, that is $k = \lfloor \sqrt{0.6 \cdot C} \rfloor$ (C is expressed in KBytes/s). Moreover, $\omega = 2$. With these new parameters, peers with a high uplink capacity open more active connections.

Note that the values of the parameters k and ω in BT and BTold are completely arbitrary; the interested reader can find a discussion about the impact of these parameters in [17].

In rest of the paper BT is the main reference to evaluate the performance of greedy clients, whereas in the online extension of this work we include the results we obtained focusing on BTold.

BitThief. The primary aim of this client was to show the intrinsic weakness of the optimistic unchoke adopted by BT. BitThief continues to contact the tracker in order to increase as much as possible its peer set size. As a consequence, the probability to be optimistically unchoked increases, and the client can receive the content without uploading at all.

BitTyrant. BitTyrant (hereinafter BTyr) adopts the same mechanisms of BitThief but also introduces a new peer selection algorithm. As for BT, the number of unchoked peers is a function of a peer's uplink capacity. However, BTyr uses a dynamic bandwidth allocation algorithm by which uplink capacity is assigned on a per-connection basis. During the initial phase of the download process, a BTyr peer allocates the same bandwidth $c = 15$ KBytes/s to all connections. This initial value has been derived in [16] from a measured peer bandwidth distribution in order to guarantee that the probability of reciprocation (i.e. the probability of being unchoked) from remote peers is high.

Subsequently, the alternative BTyr choke algorithm works as follows: if a remote peer reciprocates for at least 3 unchoking intervals, the bandwidth allocated for this active connection is reduced by a factor of 0.9. If an unchoked neighbor stops reciprocating, then the bandwidth allocated to the active connection is increased by a factor 1.2. Every choke interval (set to 10 sec.), neighbors are sorted according to the ratio of the amount of data *received* to the amount of data *sent* in the last 20 sec.; the available uplink capacity is then progressively allocated to remote peers in descending order. Hence, the amount of bandwidth allocated to a remote peer should converge to the exact value required to guarantee reciprocation.

3 Misuse Opportunities in BitTorrent: an Analytical Perspective

In this section we analyze the extent to which the altruistic behavior of BT might be exploited by self-interested peers. We adopt a *data agnostic* approach, i.e. we assume that each peer has always pieces that every other is missing. The analysis extends and formalize rigorously the key observations made in [16], which are behind the design of BTyr.

Note that the analysis is focused on the possible exploitation of BT: how a client is going to use this information is independent from the analysis. In the rest of the paper we will focus only on BTyr as strategic client, because it merges several greedy techniques discussed in the literature. We do not consider here the BitThief scheme since the evaluation of its benefits are straightforward.

3.1 Matching Time

As noted in prior studies [18], [19], the choke algorithm can be seen as a distributed algorithm for the stable b-matching problem, that converges to a (weakly) stable state in which peers are matched based on their upload capacity and no peer has an incentive to deviate from its matches. The algorithm converges to a stable state through a series of exploration rounds (i.e. optimistic unchokes) in which unstable matchings are formed. During this intermediate phase a peer may be matched to remote peers that cannot sustain a fair reciprocation. This implies that some peers might offer more upload bandwidth than they receive.

The time it takes for the algorithm to converge could be exploited by a peer striving for maximizing the reciprocation it receives from remote peers. In the following we derive a rough estimation of the convergence time, termed *matching time* hereinafter. Our analysis of the matching time (i) assumes a large swarm, with a fixed peer population, (ii) does not take into account content availability and (iii) ignores that some remote peers could be not willing to reciprocate. The last issue is going to be addressed in the following section.

During a time interval equal to T_{opt}, a peer discovers (using optimistic unchokes) the equal split of ω new peers and its equal split is discovered by other ω new peers. Given peer i with equal split u_i, let A_i be the set of its active connections (i.e. the neighbors it has unchoked). We denote with $b(u)$ and $B(u)$ respectively the Probability Density Function (PDF) and the cumulative distribution function (CDF) of the equal split. $b(u)$ ($B(u)$) can be evaluated through an empirical distribution[1].

The expected number of interactions peer i needs to find a peer with higher equal split is geometrically distributed, with expected value $1/(1-B(u_i))$. The expected number of interactions needed to discover a number of peers equal to the number of active connections $|A_i|$ is simply $|A_i|/(1-B(u_i))$. A peer has one interaction every $T_{opt}/(2\omega)$ seconds, then the matching time is:

$$\frac{T_{opt}}{2\omega}\frac{|A_i|}{1-B(u_i)}. \tag{1}$$

The equation shows that the matching time increases when the number of active connections or the equal split increases.

We derived the matching time for BT clients with different uploading capacities (cf. Appx. A.1). Matching time for BT clients is as large as 10 hours and it is not negligible with respect to typical download times.

Long matching times pave the way for clients such as BTyr that tries to exploit high-capacity peers as long as their discovery phase has not converged yet.

3.2 Probability of Reciprocation and Expected Download Rate

The extremely long convergence time toward a stable matching has encouraged the design of subtle techniques [16] to exploit peers until a global matching is reached; then peers would be immune to greedy strategies. A greedy peer, however, is not guaranteed to be reciprocated from remote peers at all times during the matching time.

1. In this work we use the same empirical distribution as in [16].

We show this by studying the *evolution in time* of the probability of reciprocation and its impact on the expected download rate of a peer. The following analysis constitutes a significant extension to that sketched in [16]. As noted above, the download rate peer i can achieve varies over time. Indeed peer i can select its $|A_i|$ best uploaders from a progressively larger set, but reciprocation from its peer set fluctuates: reciprocation from peers with higher capacity decreases (because they discover similar peers), while reciprocation from lower capacity peers increases (because they are progressively choked by their best uploaders). Being that each peer optimistically unchokes ω new peers every T_{opt}, we consider a discrete time system where every $T_{opt}/(2\omega)$ seconds each peer discovers the equal split of a new peer. Let us define $\rho(u_i, u_j, k)$ the probability that a node with equal split u_j is willing to reciprocate with a node with equal split u_i at the k-th interaction. The probability that a randomly selected peer is willing to reciprocate to peer i at the k-th interaction is

$$\int_0^\infty \rho(u_i, v, k) b(v) \mathrm{d}v,$$

and the expected number of peers not reciprocating peer i ($\overline{R}_i(k)$) is:

$$\overline{R}_i(k) = k\left(1 - \int_0^\infty \rho(u_i, v, k) b(v) \mathrm{d}v\right). \tag{2}$$

We simplify our analysis assuming that: (i) the number of peers not reciprocating peer i is always equal to the integer nearest to $\overline{R}_i$ (we denote it as $\hat{R}_i$) and (ii) that these peers are the best uploaders of peer i. These assumptions are going to be justified by our asymptotic analysis for $\rho(u_i, v, k)$. In fact we are going to show that, as k diverges, $\rho(u_i, v, k)$ converges to 0 and 1, respectively for $u_i < v$ and for $u_i > v$. Hence, at least asymptotically, peer i will not be reciprocated by all its neighbors with higher equal split (its best uploaders). This justifies assumption (ii). Moreover, being that the reciprocation probability exhibits asymptotically a sharp transition from 0 to 1 at $v = u_i$, the number of non-reciprocating peers is distributed approximately as a binomial distribution with parameters k and $B(u_i)$. This justifies assumption (i), being that a binomial distribution is concentrated (in the sense of the law of large numbers) around its mean for large value of k.

Given these two assumptions, if we rank the uploaders of peer i on the basis of their equal split in decreasing order, peer i at the k-th interaction will reciprocate peers with rank from $\hat{R}_i(k) + 1$ to

$$w_i = \hat{R}_i(k) + |A_i| = \lfloor \overline{R}_i(k) + 0.5 \rfloor + |A_i|, \tag{3}$$

assuming that it is willing to open up to $|A_i|$ connections. We use basic order statistic results [20] to derive the equal split PDF of the z-th uploader of peer i (out of the k neighbors peer i has interacted with by the k-th interaction):

$$b^{(z)}(v,k) = \frac{k!B(v)^{k-z}(1-B(v))^{z-1}}{(z-1)!(k-z)!}b(v). \tag{4}$$

Note that this distribution is the same for all the peers, because optimistic unchoking selects neighbors uniformly at random.

We derive the expected download rate of peer j as:

$$\sum_{z=\hat{R}_i(k)+1}^{\hat{R}_i(k)+|A_i|} \int_0^\infty v b^{(z)}(v,k)\mathrm{d}v + \omega \int_0^\infty v b(v)\mathrm{d}v, \tag{5}$$

where the first term corresponds to the aggregated rate from active connections, while the second one to the aggregated rate from optimistic unchoking.

Finally we derive the reciprocation probability. The probability that peer i is going to be reciprocated from peer j at the following interaction is equal to the probability that peer i has a higher equal split than that of the w_j-th uploader of peer j[2], then:

$$\rho(u_i,u_j,k+1) = \int_0^{u_i} b^{(w_j)}(v,k)\mathrm{d}v. \tag{6}$$

The system starts from a state where every peer has an empty active set and it is willing to reciprocate with everyone else ($\rho(u_i,u_j,0)=1$), then equations 2, 3, 4 and 6 can be used to evaluate the evolution of reciprocation probabilities.

Now we are going to deepen our understanding of the system, by studying the asymptotic limit of the reciprocation probability, and the relation of these results with the matching time as defined in Sec. 3.1.

First, we are going to show that, as k diverges, $\rho(u_i,v,k)$ converges to 0 and 1, respectively for $u_i < v$ and for $u_i > v$. If this is the case, then

$$\lim_{k\to\infty} \frac{\overline{R}_j(k)}{k} = \int_0^{u_j} b(v)\mathrm{d}v = B(u_j).$$

Let us define

$$\alpha(u_j) = \lim_{k\to\infty}\left(1 - \int_0^\infty \rho(u_j,v,k)b(v)\mathrm{d}v\right),$$

we will then show that $\alpha(u_j) = B(u_j)$. We observe that $\alpha(u)$ is a decreasing function of u (and hence in particular invertible), because the reciprocation probability $\rho(u,v,k)$ is increasing in u. We observe that w_j behaves asymptotically as $k\alpha(u_j)$, hence we can apply the results for *central quantiles* (or central order statistics) [20] to $b^{w_j}(v,k)$, concluding that it is asymptotically distributed as a normal with mean $B^{-1}(\alpha(u_j))$ and variance

$$\frac{\alpha(u_j)(1-\alpha(u_j))}{k\left[b(B^{-1}(\alpha(u_j)))\right]^2} \underset{k,\infty}{\rightarrow} 0.$$

Being that the variance converges to zero, we can derive from Eq. 6 that $\rho(u_i,u_j,k)$ converges to 1 if $B^{-1}(\alpha(u_j))$ -the mean of the Gaussian- is included in the integration range, otherwise it converges to zero. In conclusion $\rho(u_i,u_j,k)$ converges to 0 and 1, respectively for $u_i < B^{-1}(\alpha(u_j))$ and for $u_i > B^{-1}(\alpha(u_j))$. Due to the monotonicity of $B()$ and $\alpha()$, $u_i < B^{-1}(\alpha(u_j))$ if and only if $u_j < \alpha^{-1}(B(u_i))$. From Eq. 2 it follows that

$$\begin{aligned}\lim_{k\to\infty}\frac{\overline{R}_i(k)}{k} &= \lim_{k\to\infty}\left(1 - \int_0^\infty \rho(u_i,v,k)b(v)\mathrm{d}v\right) \\ &= \int_0^{\alpha^{-1}(B(u_i))} b(v)\mathrm{d}v = B(\alpha^{-1}(B(u_i))).\end{aligned}$$

By definition this is equal also to $\alpha(u_i)$:

$$B(\alpha^{-1}(B(u_i))) = \alpha(u_i).$$

This equality holds for every value of u_i, then it has to be $\alpha() = B()$. We have so concluded our proof that $\rho(u_i,u_j,k)$ converges to 0 and 1, respectively for $u_i < u_j$ and for $u_i > u_j$.

We may want to quantify the convergence time similarly to what done in the previous section for the simplified mode. It is natural to consider the expected time needed for peer i to find $|A_i|$ peers (i) with higher equal split (as for the matching time) *and* (ii) willing to reciprocate. This second requirement makes this convergence time longer than the matching time (already several hours for high capacity clients). Hence, the refined analysis in this section points out an even longer phase during which the system can be exploited by greedy peers. At the same time, it shows that the duration of the exploitable phase is not the only important aspect. In fact, the probability to reciprocate to peers with lower bandwidth converges to zero (the faster the lower the bandwidth is). This limits the extent to which greedy peers can exploit compliant ones. In Appx. A.2 we quantify the possible gain by numerical simulations both of the time evolution of the probability of reciprocation and the expected download rate for different values of upload capacity.

3.3 Discussion

Our data agnostic analysis indicates that exploiting BT clients appears tempting in a first instance, if one considers the time required by the peer selection to stabilize. However, due to the variability in time of the probability of reciprocation, a greedy strategy would work best during the initial stages of the download process, where high capacity peers are still willing to serve low and intermediate capacity peers.

This conclusion raises the legitimate question of whether these results carry over when *piece availability* is considered. Indeed, piece availability plays a crucial role, especially during the initial phase of the download process, when the number of pieces being exchanged by peers is scarce. Due to the complexity of the analysis when piece availability is taken into account, we revert in the following to a simulation-based performance analysis.

2. If $k < w_j = \hat{R}_j(k) + |A_j|$, peer j will be always willing to reciprocate with a new peer.

4 Deconstructing BitTyrant: the Single Client Case

In the following we carry out a simulation-based analysis of the performance of BTyr. We decided to focus on BTyr because it merges several greedy techniques previously discussed in [15], [14]: (i) greedy peer set size and (ii) greedy uplink allocation:

(i) implies that peer set size in BTyr is larger than that of a BT client (this approach is adopted also in BitThief [15]) resulting in a higher probability of being optimistically unchoked;

(ii) implies that the uplink capacity of a peer is not equally split among its active connections, but shaped according to a greedy objective; hence, the number of active connections varies over time.

Here we characterize the contributions of the BTyr building blocks to the performance achieved by a **single** BTyr client in a torrent where all the other clients are BT.

4.1 Simulator Description, Methodology and Settings

Our work is based on the publicly available BitTorrent simulator called GPS [21], customized for our needs. In this environment, peers have infinite downlink capacity and a finite uplink capacity. The uplink capacity is randomly chosen according to the bandwidth distribution measured in [16].

The main performance metrics we use are:

- **Download time** of the single client (BT, BTyr or BTold) in the different scenarios;
- **Download time** of of all peers;
- **Number of pieces** uploaded by the single client during the download process.

We decided to follow the system parameters used in [16]: we analyze torrents of 350 peers where one initial seed shares a file of 50 MB. We chose such file size since we noted that the gain of a strategic client is mainly *concentrated at the beginning* of the distribution process (cf. Sect. 3), so BTyr should benefit more from downloading smaller files.

In order to verify the results against different scenarios, we have also performed experiments with bigger file size (350 MB) and bigger torrents (500 peers) obtaining similar results.

Peers randomly start to download the content within a small interval of time (10 sec.) and stay as seeds in the system once they finish downloading the content. This scenario represents the most favorable scenario for BTyr, since peers start downloading approximately at the same time and they do not have any knowledge of the other peers' bandwidth.

We carry out a comparative performance analysis of a *single peer* using whether BT or BTyr client when the *rest of the torrent population* use BT clients. In order to make a fair comparison, at every change of the client of the single peer, we leave unchanged its characteristics (bandwidth, arrival instant), as well as the characteristics of the other peers. We estimate the mean download time over multiple runs (if not specified, we perform ten runs for each experiment), along with the confidence interval for a confidence level of 95%.

4.2 Impact of the Peer Set Size and Active Set Size

In this Section we build a *baseline* scenario in which a single, fully-fledged BTyr client operates in a torrent of BT peers. In this case BTyr keeps contacting the tracker in order to increase the peer set as much as possible. A similar scenario has been used in the experiments showed in [16], with the difference that the torrent was composed by BTold peers. We show the results for this case (BTyr in a torrent of BTold peers) in Appx. B.1, which are coherent to the ones found in [16].

When we consider a BTyr client in a torrent of BT peers, the performance gain of BTyr disappears. Fig. 1 illustrates the download time of a single BT and BTyr client for different classes of uplink capacity. The reason is due to the large number of active connections (active set size) established by fast peers using BT. Their uplink capacity is over-partitioned, hence remote peers (including the BTyr client) receive smaller download rates as compared to the original BTold algorithm.

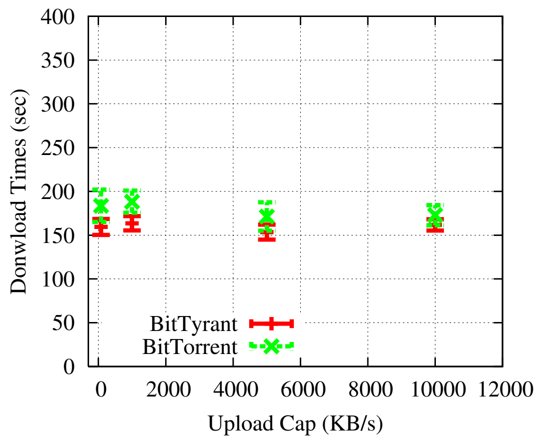

Fig. 1. Mean download time of a single client with different bandwidths (95% confidence interval).

The BTyr itself maintains a large number of active connections. On the one hand, by keeping a larger number of active connections, BTyr strives for maximizing the chance of being reciprocated. On the other hand, since during the initial phase of the download process the lack of fresh pieces to serve could cause uplink capacity underutilization, a larger active set size helps spreading available pieces to a large number of peers that would otherwise remain unserved. This increases the utilization of the uplink capacity of both the BTyr peer and its neighbors. Interestingly, the greedy strategy adopted by BTyr has actually a hidden altruistic nature. For further details, please refer to Appx. B.2.

As additional test, we obstruct the peer set construction of BTyr: the peer set size is then equal at most to 80 for every peer in the torrent. Fig. 2 shows a slightly decreased performances of BTyr, even if we can not exactly quatify this decrease, since the variation remains on the order of the confidence interval size. A similar result can be seen in Appx. B.1, where we show the same experiment in case of BTyr in a torrent of BTold peers.

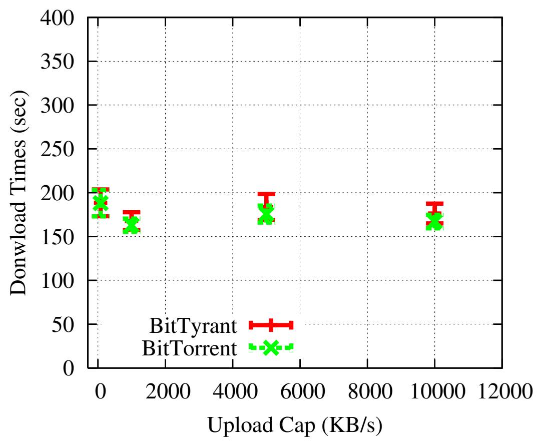

Fig. 2. Mean download time of a single client with a constrained peer set (95% confidence interval).

These results indicates that the increased peer set size may constitute one of the factors influencing download performance.

4.3 Impact of Greedy Uplink Capacity Allocation

In the previous section we focused on the peer set and active set sizes. It remains to evaluate if the uplink capacity allocation algorithm adopted by BTyr is actually able to increase the performance. In particular, we now focus on the interaction between a peer and its peer set and ask whether peers tend to match with neighbors with similar uplink capacity in variants of the BT client.

Fig. 3 shows the ECDF of the uplink capacity of the neighbors unchoked during the whole file download by a peer with an uplink capacity of 200 KB/s.[3].

We observe that a single BTold client unchokes a small subset of its neighbors (the ECDF has a few well defined steps) and mostly reciprocates remote peers with a similar uplink capacity. Ideally, a completely stratified system would imply a single step ECDF function centered on the observed peer's uplink capacity. In practice, as discussed in Sec 3, the system requires time to converge to a stable state, during which peers may end up cooperating with remote peers with different uplink capacities. In contrast to our observations on BTold, BTyr and BT interact with any neighbor in their peer set, irrespectively of the uplink capacity, in a manner that resembles to a round robin approach. Although the intuition behind the design of BTyr was to seek and exploit for the longest possible time the fastest peers using a clever uplink allocation algorithm, no stable matching appears.

3. The figure also reports the uplink capacity distribution of all peers that we used in our experiments.

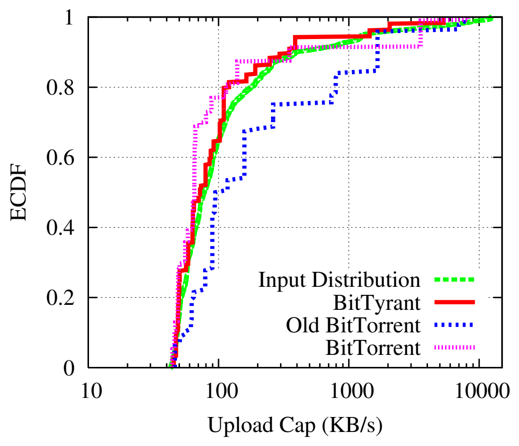

Fig. 3. ECDF of the upload bandwidths of the active set seen by a BT, BTold and BTyr client with an uplink capacity of 200 KB/s.

In order to understand if the dynamic uplink bandwidth allocation algorithm works properly, we take a different perspective. We neglect the effect of piece availability in our simulations: we focus only on the exact values allocated by the BTyr choke algorithm to remote peers, rather than on the actual amount of data sent or received.

To this aim, we show the uplink rate assigned by the BTyr peer to each neighbor, over time: for the observed BTyr peer we maintain a matrix E where the element e_{ij} represents the rate assigned to peer i at choking interval j.

Fig. 4 illustrates the matrix E for a BTyr peer k with 5000 KB/s uplink capacity. The value of rate is visualized using shades of gray: the darker regions indicate higher rates. During the initial phase of the download process, peer k allocates the same uplink rate to all its neighbors. Note that, since the initial rate for each unchoked neighbor is 15 KB/s, the BTyr peer unchokes *all* its neighbors.

The allocated uplink capacity varies over time, and it's possible to observe a specific trend: the bandwidth is initially equally divided among the peer set; then the peer assigns an an increasing amount of bandwidth, which degenerates into a periodic, on-off, phase. A deeper analysis shows that not all the high capacity neighbors are detected or maintained. For instance, in the figure, neighbors with ID 141, 153 and 199 have high bandwidth, and they are detected but not maintained (after 40 rounds the BTyr peer stops unchoking them). Moreover, the periodic trend, common to all the neighbors, represents an undesired behavior that limits the performance of the scheme: the received rate, in fact, goes periodically to zero for three rounds, forcing BTyr to try to be reciprocated again by the neighbors.

The existence of a periodic behavior hints at the pres-

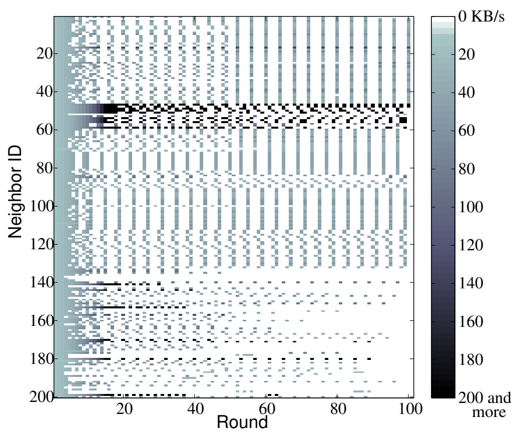

Fig. 4. Upload rate in the data agnostic case: snapshot for a fast BTyr client.

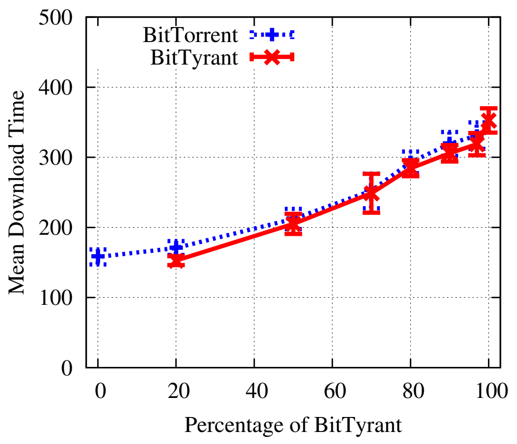

Fig. 5. Average of the mean download times for an increasing fraction of BTyr clients in a population of BT peers (95% confidence interval).

ence of closed loop dynamics that the original design has not foresee. The limited space does not allow for a deeper analysis from a control theory viewpoint, which we reserve for future works. Our extensive results illustrate an unexpected behavior of BTyr that may have an impact on the protocol performance. As a reference, the matrix representation of the upload algorithm in BTold is shown in Appx. B.3.

5 The Multiple Clients Case

In this Section we study the behavior of a system in which an initial population of BT clients is gradually replaced by an increasingly larger fraction of greedy variants. This study is motivated by the fact that if the first users adopting a greedy variant experience improved performance in comparison to BT, further users can be attracted to adopt it. Since the BTyr clients have problems to interact with each other, [16] suggests to cope with this situation using a block-based Tit-For-That (TFT) strategy: however, a number of issues related to the BTyr block based TFT choke algorithm suggest that this technique might be ineffective (cf. Appx. C.1). Therefore, in the following we analyze the performance of multiple BTyr clients that do not implement the block based TFT mechanism.

5.1 Increasing Fraction of Greedy Variants

We consider the implications of gradual user adoption of BTyr clients. We ask whether it is possible to predict an equilibrium point by which there would be no incentive for users to adopt the BTyr variant. Hence, we focus on the effects of an increasing adoption of BTyr clients in a swarm composed by BT clients on users' download times. In the following experiment, we use the same settings described in previous sections, and introduce an increasing percentage of BTyr clients.

Fig. 5 shows this scenario. The is no performance improvement of clients switching from BT to BTyr, and no equilibrium point is found: therefore we can conclude that BT seems to be robust against an adoption of even a small fraction of BTyr clients. Interestingly, we note that the increase of the percentage of BTyr clients has a negative impact on the mean download time of the swarm: in the next section, we dissect this issue, considering the extreme case of a pure BTyr swarm.

In Appx. C.2 we study similar scenarios, where users with BTold clients start adopting BTyr clients, or BT. The interested reader is reffered to Appx. C.2 for furhter details.

5.2 The Case of Massive Adoption

We focus on the extreme scenario of massive adoption of BTyr. Fig. 6 illustrates the ECDF of the download times for a torrent of all BT and BTyr clients: a glance at the median and worst case download times indicates that a large-scale adoption of BTyr can indeed jeopardize the content distribution process, even with a constrained peer set size.

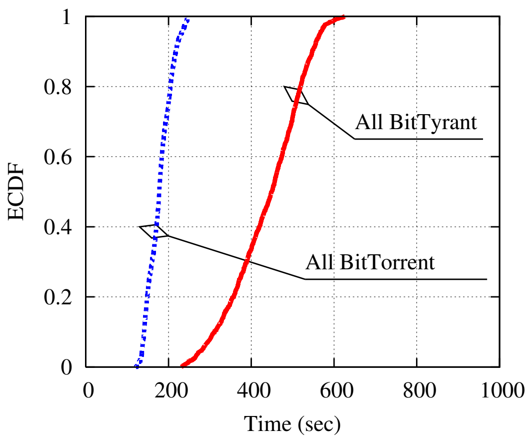

Fig. 6. ECDF of download times for BT and BTyr.

To understand exactly why system performance degrades when all peers use BTyr, we analyze the upload capacity allocation of a fast BTyr peer in Appx. C.3.

5.3 PlanetLab Experiments

We summarize the results for a set of experiments obtained using PlanetLab [22] as a testbed. For the details of such experiments, we refer to Appx. C.4.

We have reproduced a similar set of experiments shown in Fig. 5 using real BT and BTyr software distributions. We note that BTyr has no gain over BT. In any case, the results confirm that BTyr clients need a favorable environment, otherwise the gains seems unpredictable.

6 Conclusions

Recent days have witnessed the development of greedy peer-to-peer clients aiming at decreasing content download times by leveraging subtle techniques to exploit generous clients. In this work we focused on BitTorrent networks and analyzed two commonly deployed greedy techniques (implemented in BitTyrant). We showed that the BT protocol can be misused to gain an advantage over standard peers by building progressively larger peer sets.

We then deconstructed the other greedy building blocks – which constitutes the BitTyrant clients – and showed, on the one hand, some undesired behaviors, such as periodic chokes/unchokes, that limit the performance of the client; on the other hand the greedy uplink allocation algorithm of BitTyrant has some positive implications on the content distribution process, especially during its bootstrap phase.

Furthermore, we shifted our focus from the performance of a single BitTyrant client to the analysis of system performance when an increasing number of such greedy clients is adopted. Our results indicate that a gradual "invasion" of BTYr clients progressively degrades the mean dowload time of the swarm, suggesting that a massive adoption of BitTyrant has to be avoided. Nevertheless, we studied this extreme scenario and our results pinpoints at a severe performance degradation for all peers as a consequence of instabilities of the choke algorithm of BitTyrant, that was not designed to function in competition with other similar clients.

The instability problems of subtle variations of the choke algorithm highlighted by our work indicate that further work is required to exploit the potential benefits of alternative schemes to allocate the uplink capacity of a peer. Along the same lines, our results show that the last version of the legacy BitTorrent client is more robust than his predecessor to BitTyrant. Moreover, we noticed a significant performance improvement of BitTorrent due to a larger number of active connections for high capacity peers (cf. Appx. C.2). Hence, the adoption of the new version of the choke algorithm of BitTorrent in all clients supporting this protocol is highly recommended.

References

[1] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. of P2P-Econ*, Berkeley, California, USA, June 2003.

[2] http://www.guardian.co.uk/technology/2006/oct/19/guardianweeklytechnologysection.insideit

[3] http://www.theglobeandmail.com/servlet/story/RTGAM.20071128.wgtbittorrent29/BNStory/Technology

[4] http://arstechnica.com/news.ars/post/20080421-study-bittorren-sees-big-growth-limewire-still-1-p2p-app.html

[5] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," in *Proc. of ACM SIGCOMM* Seattle, WA, USA, Aug. 2008.

[6] A. Legout and G. Urvoy-Keller and P. Michiardi "Rarest first and choke algorithms are enough," in *Proc. of IMC*, Rio de Janeiro, Brazil, October 2006

[7] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems", in *Proc. of IMC*, Berkeley, California, USA, October 2005

[8] M. Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, Pascal Felber, Anwar Al Hamra, Luis Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime" in *Proc. of PAM* Antibes, France, April 2004

[9] X. Yang and G. de Veciana, "Performance of Peer-to-Peer Networks: Service Capacity and Role of Resource Sharing Policies," in Performance Evaluation, Vol. 63, Issue. 3, 175-194, March 2006

[10] A. R. Bharambe, C. Herley, V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *Proc. of INFOCOM*, Barcelona, Spain, May 2006

[11] D. Qiu and R. Srikant, "Modeling and performance analysis of BitTorrent-like peer-to-peer networks," in *Proc. of SIGCOMM*, Band, Alberta, Canada, August 2005

[12] F. Bin, D. M. Chiu, J. C.S. Lui, "The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design," in *Proc. of ICNP* Santa Barbara, USA, 2006

[13] F. Bin, D. M. Chiu, J. C.S. Lui, "Stochastic Differential Equation Approach to Model BitTorrent-like P2P Systems," in *Proc. of ICC*, Istanbul, Turkey, June 2006

[14] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)," in *Proc. of IPTPS*, Santa Barbara, California, USA, February 2006.

[15] T. Locher and P. Moor and S. Schmid and R. Wattenhofer "Free Riding in BitTorrent is Cheap," in *Proc. of HotNets-V*, Irivine, California, USA, November 2006.

[16] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?," in *Proc. of NSDI*, Cambridge, MA, USA, Apr. 2007.

[17] N. Laoutaris, D. Carra, P. Michardi, "Uplink Allocation Beyond Choke/Unchoke or How to Divide and Conquer Best," in *Proc. of ACM CoNext*, Madrid, Spain, Dec 2008.

[18] A. Legout, N. Liogkas, E. Kohler, and L. Zhang "Clustering and Sharing Incentives in BitTorrent Systems," in *Proc. of SIGMETRICS*, San Diego, CA, USA, June 2007.

[19] A. Gai, F. Mathieu, F. de Montgolfier, J. Reynier, "Stratification in P2P networks: Application to BitTorrent," in *Proc. of ICDCS*, Toronto, Ontario, Canada, July 2007.

[20] H. A. David and H. N. Nagaraja, "Order statistics", John Wiley & Sons, Inc., 2003.

[21] W. Yang and N. Abu-Ghazaleh, "GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent," in *Proc. of MASCOTS*, Atlanta, Georgia, USA, September 2005.

[22] PlanetLab, http://www.planet-lab.org/, Dec. 2009.

Appendix A

A.1 Matching time

In Fig. 7 we show the matching time for BTold and BT clients with different uploading capacities. Matching times are as large as 1 and 10 hours respectively for high capacity BTold and BT clients and they are not negligible with respect to typical download times. The sawtooth behavior of the BT curve is due to non-continuous relation between the uploading capacity and the equal split. Given two peers with similar capacities, it can happen that the one with higher capacity opens one additional connection. Hence it has a larger set of active peers ($|A_i|$ is larger), but at the same time its equal-split is smaller and the time needed to discover a single faster peer is shorter ($1/(1 - B(u_i))$ is smaller). For the specific empirical distribution, this second effect prevails and this justifies the shape of the curve.

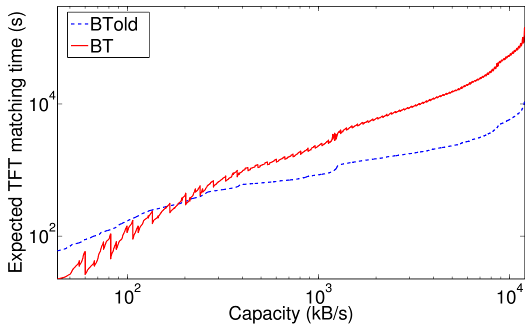

Fig. 7. Time required for a new peer to discover a number of peers of equal or greater equal-split to fill its active set.

A.2 Probability of Reciprocation and Expected Download Rate

Fig. 8 shows the reciprocation probability for BTold clients after 150 seconds and after 15 minutes, respectively the time intervals needed by each peer to discover the equal splits of 10 and 60 peers. Every point (x, y) of the figure indicates the probability that a peer with capacity x is going to be reciprocated by a peer with capacity y. After 150 seconds (Fig. 8-a), peers with lower uplink capacities are very unlikely to be reciprocated by fast peers; however, the probability for fast peers to reciprocate remote peers that cannot sustain their upload rates is very close to one. This observation no longer holds after 15 minutes, (Fig. 8-b): in this case a large fraction of peers is willing to reciprocate only with other peers with similar or higher capacities. In general we notice that there is a relation between the bandwidth of a peer and the duration of the possible exploitation of other peers: the lower the bandwidth a peer has, the less the probability it can exploit high bandwidth peers for a long time.

Fig. 9 shows the corresponding results for the BT client. BT appears to be more generous in that the

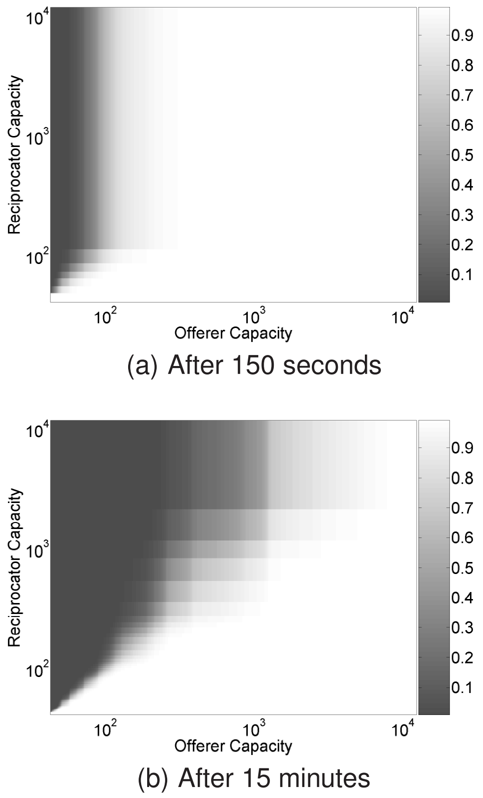

Fig. 8. Reciprocation probability for BTold.

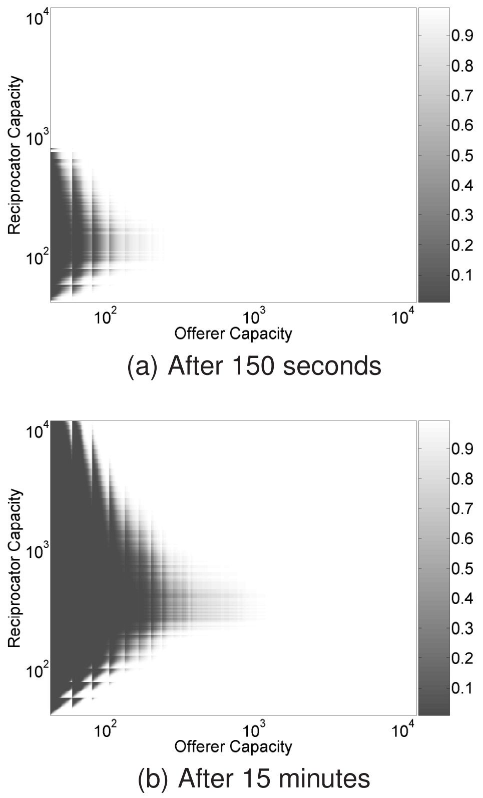

Fig. 9. Reciprocation probability for BT.

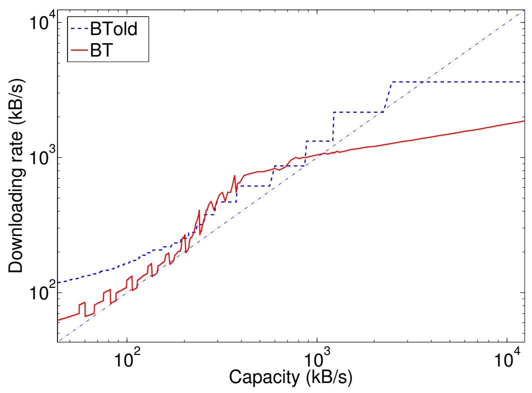

Fig. 10. Expected download rate for a peer of a given capacity after 15 minutes.

probability of an unfair reciprocation (a slow peer being served by a fast one) is still high after 15 minutes.

Fig. 10 reports the expected download rate of a peer with a given uplink capacity, after 15 minutes from the beginning of the download process. Fairness is achieved when the uplink capacity equals the expected download rate (diagonal line in the figure). We recall that both regular and optimistic unchokes contribute to the download rate observed by a peer. In the BTold case, Fig. 10 illustrates that low capacity peers are able to get more then their fair rate. This is mainly due to optimistic unchokes: focusing only on regular unchokes would reveal that the expected download rate is parallel to the diagonal up to roughly 200kB/s. On the contrary, peers with capacity greater than 3000kB/s offer more upload capacity than they receive: this is exploited by peers with intermediate upload capacity.

While similar observations can be drawn for the BT case, we notice that the advantage for low capacity peers is less pronounced: this is due to the larger number of active connections (hence lower uplink bandwidth dedicated to each of them) of a BT client.

Appendix B

B.1 Impact of the Peer Set Size in BTold

Fig. 11 illustrates the download time of a single BTold, and BTyr client for different classes of uplink capacity in the baseline case where all the other clients are BTold. We observe that the performance gain of BTyr over BTold is always present and particularly pronounced on fast peers.

Fig. 12 shows the download time for the same set of experiments shown above when the greedy peer set construction is obstructed. The results illustrate a performance loss of BTyr in a torrent of BTold clients, corroborating - along with the observations in Section 4.2 - the hypothesis that the increased peer set size constitutes one of the factors influencing download performance.

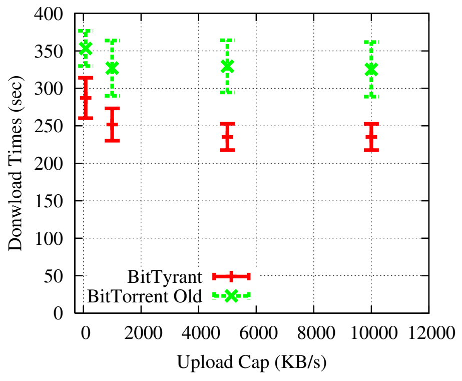

Fig. 11. Mean download time of a single client with different bandwidths (95% confidence interval).

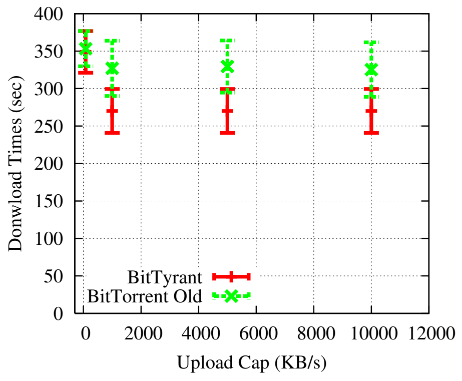

Fig. 12. Mean download time of a single client with a constrained peer set (95% confidence interval).

B.2 Altruistic Impact of the Active Set Size in BTyr

Fig. 13 depicts the ratio between the cumulative number of uploaded pieces over time by the single BTyr client with respect to the corresponding BT client. Especially during the early stages of content distribution, BTyr uploads up to 25 times (for a high bandwidth peer) the number of pieces uploaded by BT. During steady state, the ratio of the number of pieces uploaded from BTyr and BT converges to 1.

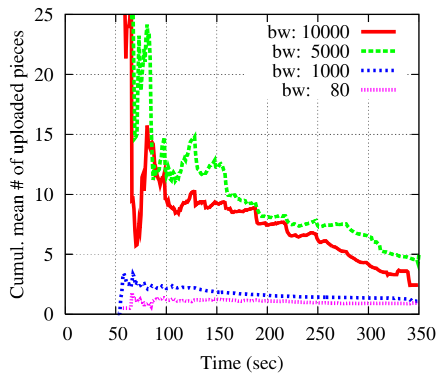

Fig. 13. Time series of the ratio between cumulative uploaded pieces by BTyr and BT.

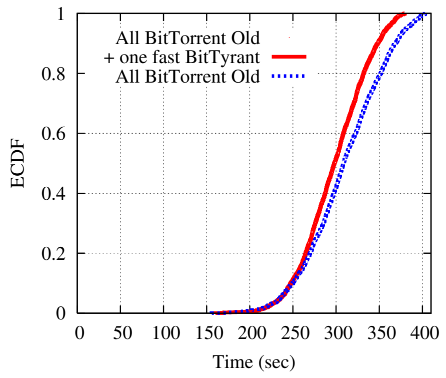

Fig. 14. ECDF of download times with or without a single standard BTyr client with bandwidth 5000 KB/s.

The unexpected altruism of BTyr has a beneficial effect on all peers involved in the distribution process. In Fig. 14 we show the empirical cumulative distribution function (ECDF) of the download times of all peers in the system. Results indicate that, when even only one fast peer (with high bandwidth equal to 5000 KB/s) adopts BTyr instead of BTold, there is a positive impact on the performance of all the other peers. We also note that similar observations can be made when introducing one BT client in a swarm of BTold ones.

The results obtained in this section hint toward an important direction of future research, that is the study of dynamic uplink allocation algorithms, where the number of active connections is not an empirically set parameter as done in BT. However, we showed in Section 4.3 that the apparently attractive uplink allocation strategy of BTyr cannot readily be used by all peers in a system.

B.3 Matrix Representation of BTold upload algorithm

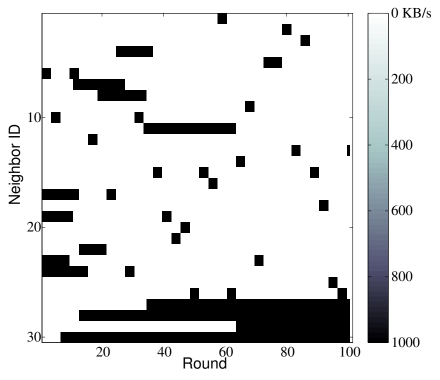

Fig. 15. Upload rate in the data agnostic case: snapshot for a fast BTold client.

We performed the same test described in Sect. 4.3 with a BTold peer, whose matrix E is showed in Fig. 15 The matching mechanisms is evident for BTold: after 30 rounds three out of four slots are stable matches. The optimistic unchokes then discover the last slot after 65 rounds.

Appendix C

C.1 Issues in torrents with multiple BTyr peers

Authors in [16] make the point that there are reasons to assume an increasing popularity of BTyr and present some initial results for the case of a torrent of all BTyr clients. They argue that a wide-spread adoption of BTyr may have a *negative impact* on global system performance: when two BTyr clients meet, in fact, they can start decreasing the rate they provide each other, yet maintaining a good ratio between the data received and sent. The final result is that both clients unchoke each other, but they provide a small rate (e.g., not even sufficient to upload one block in one chocking interval).

To cope with this problem, [16] suggests the following fix: when peers establish a connection and perform the initial handshake, if they realize that they both are using BTyr, they should switch to a block based TFT strategy, trying to increase the rate rather than decrease it. In order to evaluate BTyr in the multiple client case, this modification should be taken into account. However, as we illustrate in the following, a number of issues related to the BTyr block based TFT choke algorithm suggest that this technique might be ineffective.

The original BTyr variant has been implemented using the Azureus (now known as Vuze) client. An analysis of the source code reveals that the block based TFT has not been implemented, and a clear algorithm specification is not discussed in the original paper [16]. We argue that the main practical problem with a BTyr client to implement such a strategy relates to the difficulty of handling connections and share resources (that is, bandwidth) with different variants of BT. Consider the following example: a BTyr client A is interacting with another BTyr client B and with a BT client C. Assume that A receives from B and C a rate such that in the next choking interval they should be both unchoked. Client A is applying block based TFT with client B and decides to increase the uploading rate to that client. At the same time, client A is trying to maximize the reciprocation from client C. When the bandwidth has to be assigned to clients B and C, how can client A split it? If client B requires more bandwidth (two BTyr clients should try to increase the rate when they exchange data) is it better to give to B more bandwidth taking away from client C? In other words, is it better to invest in a block based TFT (which will give back the same amount of transferred data) or to maintain a risky connection with C (which can choke client A, but which can provide better reciprocation)?

In addition, clients' identities can be easily forged in the current BT protocol specification: regular BT clients might consider BTyr as a threat, hence a simple "admission control" mechanism that would refuse a connection to a BTyr peer could be implemented in future versions

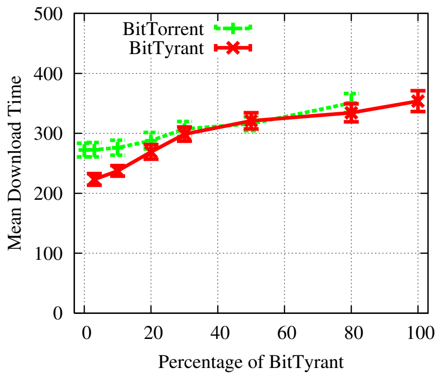

Fig. 16. Average of the mean download times for an increasing fraction of BTyr clients in a population of BTold peers (95% confidence interval).

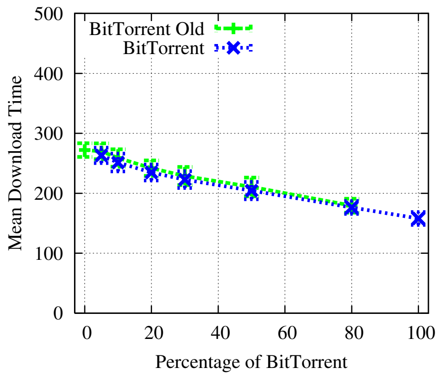

Fig. 17. Average of the mean download times for an increasing fraction of BT clients in a population of BTold peers (95% confidence interval).

of BT. As a consequence, BTyr peers should hide their identities in order to exploit BT clients: however, by doing so, they would be incapable of recognizing other BTyr peers and fail in switching to a block based TFT mechanism.

C.2 Increasing Fraction of Greedy Variants (BTold)

We ask whether it is possible to predict an equilibrium point by which there would be no incentive for users to adopt the BTyr variant. In this case, we focus on the effects of an increasing adoption of BTyr clients in a swarm composed by BTold clients on users' download times. In the following experiments, we use the same settings described in Sect. 5.1.

Fig. 16 illustrates the average of the mean download times respectively (with 95% confidence intervals) for the population of BTyr and BT clients, as a function of the percentage of BTyr clients in the swarm. For a small fraction of BTyr (3%), the performance improvement amounts to roughly 22% in favor of greedy clients. However, the gap shrinks as soon as 20% of peers use the BTyr client, and disappears at around 30% of invaders. Our experiments show that, a fraction of 30% BT clients seems to be a critical threshold in the system, which can be seen as an equilibrium point, in game theoretic terms. When 30% of the torrent is composed of greedy peers, download times for both BT and its variant are similar, hence there would be no incentive for a peer currently using BT to switch to BTyr (nor the inverse).

Armed with the observations on the new version of the BT protocol discussed in Sec. 4, we now study the effects of a widespread adoption of BT clients. We ask whether the update of the BT protocol brings an actual performance improvement. Fig. 17 illustrates the effects on the average of the mean download times of an increasing percentage of BT peers in a population of BTold peers. The conclusion of this experiment is that the performance of BTold and BT are comparable, so the single user gains a very little advantage with the BT upgrade. Interestingly, the mean download times of BTold and BT are highly correlated, meaning that BTold clients benefit from an increasing fraction of BT peers: when 50% of peers in the system adopt BT, we observe a performance gap of roughly 25% favoring both BTold and BT peers as compared to a swarm of BTold clients. The gap widens to approximatively 36% when most or all peers use BT. Our results indicate that the transition to a swarm composed by peers adopting the new version of BT has a sensible social benefit.

C.3 Massive Adoption

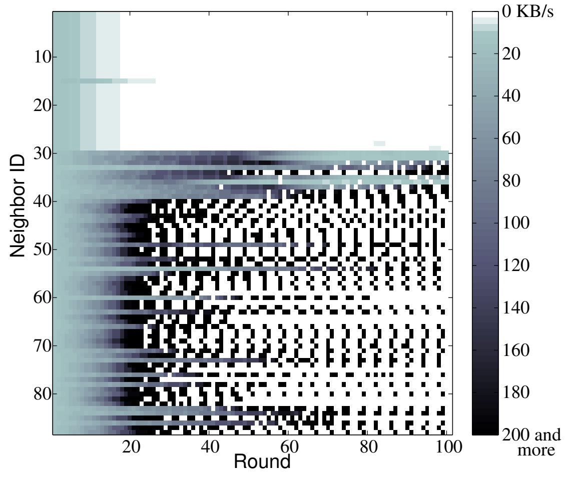

Fig. 18. Upload rate in the multiple clients case: snapshot for a fast BTyr client.

We generalize the matrix E for every peer k in the system, i.e., for every peer k we record a matrix $E^{(k)}$ where the element $e_{ij}^{(k)}$ represents the rate assigned by peer k to peer i at choking interval j. Fig. 18 illustrates $E^{(k)}$ for a peer k with 10000 KB/s uplink capacity.

It's possible to observe two different trends: (i) some neighbors of peer k are allocated less and less uplink bandwidth; (ii) other neighbors are assigned an increasing amount of bandwidth, which then degenerates into a periodic, on-off, phase.

The decreasing allocation of bandwidth, as explained in Appx. C.1, is related to the BTyr bandwidth allocation algorithm: when two BTyr clients meet, in fact, they can start decreasing the rate they provide each other, yet maintaining a good ratio between the data received and sent. The final result is that both clients unchoke each other, but they provide a little rate (e.g., not even sufficient to upload one block in one chocking interval).

When BTyr interacts with BT, a periodic pattern appears, as we observed in Sect. 4.3. In this case, instead of providing a constant rate to almost all the neighbors, the rate is continuously increasing. The initial increasing trend can be explained as follows: on the one hand, peer k has *spare* uplink capacity, thus it unchokes *all* its neighbors; on the other hand, its neighbors may have limited capacity, hence they choke peer k. As a consequence, peer k (that follows the BTyr choke algorithm) increases the uplink capacity to remote peers to increase the probability of reciprocation. This behavior is visible for the first 20-30 rounds. At this point, the rate allocated by peer k to remote peers reaches a very high value. As a consequence, (i) peer k starts choking some neighbors, since it does not have enough capacity for all of them; (ii) on the contrary, peer k's neighbors start unchoking it. These two phases are interleaved and concur in creating the periodic behavior.

A close look at Fig. 18 indicates that the periodicity is equal to three rounds[4]. This is a consequence of the probing period used by BTyr (and BT) to estimate the received/sent rate.

C.4 PlanetLab Experiments

Setting up a realistic testbed to explore different scenarios as done in simulation is a non-trivial task. Nevertheless, it is important to understand if our most important findings can be observed using clients deployed in more realistic settings. In this spirit, the PlanetLab results should be considered as additional results, not as a full evaluation of the BTyr client.

We use approximately 300 PlanetLab nodes: part of the nodes use the official BTyr client, while the remaining use the basic BT client in order to reproduce the experiments shown in Sect. 5.1 . The content size is the same used for the simulations, and the seed is a well provisioned peer hosted on dedicated machines.

As opposed to simulations settings, here we do not control the distribution of peers' bandwidths: the PlanetLab nodes are usually well provisioned, thus the bandwidth distribution differs from the one used in the simulations.

Alternatively, we could have set an upload cap to mimic the bandwidth distribution of our simulations, but this approach is problematic. Since this distribution has long tails and the maximum bandwidth in PlanetLab is limited, we would obtain a high percentage of nodes with very small bandwidths, resulting in extremely high download times and a too short choking interval (cf. [10]). Scaling all the parameters – the file size, the chunk size, the choking interval – may have unexpected effects on the execution of the key algorithms of BT.

In Fig. 19 we show the results obtained by varying the percentage of BTyr. For each scenario (e.g., 10% of BTyr nodes), we perform three different experiments, we compute the mean download time of each experiments, and then we calculate the average of these three mean download times, along with the confidence interval for a 95% confidence level.

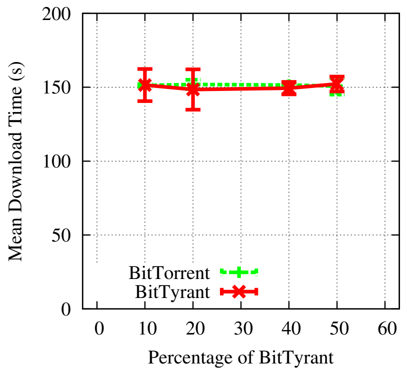

Fig. 19. Average of the mean download times for an increasing fraction of BTyr clients in a population of BT peers: PlanetLab experiments (95% confidence interval).

4. Similar results are obtained for a peer k with different uplink capacity.