UNVEILING BITTYRANT: WHEN THE DEVIL IS NOT SO BLACK AS HE IS PAINTED.

Damiano Carra
Giovanni Neglia
Pietro Michiardi

January 2008, ver. 1.3

Technical Report # RR-08-209

Unveiling BitTyrant: When the Devil Is Not So Black as He Is Painted*

Damiano Carra[1], Giovanni Neglia[2], and Pietro Michiardi[1]

[1] Institut EURECOM, Sophia Antipolis, France
{carra,michiardi}@eurecom.fr
[2] INRIA, Sophia Antipolis, France, and Università degli Studi di Palermo, Italia
giovanni.neglia@ieee.org

Abstract. This work focuses on BitTyrant, a strategic client that has been recently proposed as an alternative to BitTorrent. 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. In this paper we analyze in details the various mechanisms used by BitTyrant and precisely identify their contribution to the increased 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 for 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 apparent 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 details. In contrast, a widespread adoption of the latest version of the *mainline* BitTorrent client would provide increased benefit for all peers.

1 Introduction

The success of BitTorrent [1] has recently motivated a huge amount of work in peer-to-peer networks and applications. The properties of BitTorrent and of its key building blocks have been dissected through measurement [2], simulation [3] and analytical studies [4] with the ultimate goal of assessing the performance and scalability of the protocol and its algorithms. While the majority of works in the literature focus on performance, there are few notorious examples [5][6] that aim at assessing the possibility of exploiting the inherent altruistic nature of BitTorrent: recent days have seen the emergence of alternative BitTorrent clients exhibiting an aggressive behavior with the goal of locally improving their performance, even in the extreme case of lack of cooperation. Indeed, it is well known that BitTorrent is based on a tit-for-tat (TFT) strategy, i.e., each peers tries to enforce reciprocation from peers it is connected to. In theory, peers that do not contribute by uploading data to their neighbors would eventually be banned. However, in order to reach a steady state regime in which the full system capacity is utilized, BitTorrent peers need to perform a search process to discover

* This work was partially supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission

their best neighbors in terms of reciprocation: this requires a certain amount of altruism because the very nature of a TFT strategy calls for some initial risk taken by peers to initiate a reciprocation process. This built-in mechanism can be exploited by parasite nodes.

In this paper we focus on the BitTyrant client [5]. Qualitative experimental results have shown the performance gain achieved by such an aggressive client and hint at its possible widespread adoption by the user community. Our goal is to pinpoint the key reasons for the improved performance of a single BitTyrant peer and we do so both by extending the analytical model presented in [5] and by a through simulation study of the performance of BitTyrant.

Our key findings challenge previous results obtained in [5]: the subtle uplink allocation algorithms used by BitTyrant is only slightly responsible for its increased performance. Contrary to what a strategic client aiming at locally improving its performance would do, BitTyrant is helpful for a system of BitTorrent peers, especially during the initial phase of the content distribution process. Instead, the main factor that accounts for increased performance lies in the technique used by BitTyrant to increase the size of its neighborhood, that results in an increased probability of exploiting the altruism of BitTorrent peers. Based on our results, we further extend the analysis to a new version of the *mainline* BitTorrent client and we make the case for the extreme scenario in which all users adopt the BitTyrant client. For this latter case, we show that aggressive clients can jeopardize the content distribution process when compared to a scenario in which only the legacy and the recent versions of BitTorrent are used.

2 Background

In this section we briefly outline the key algorithms used by BitTorrent and BitTyrant; [1] presents a detailed description of the BitTorrent protocol.

BitTorrent. The BitTorrent protocol (hereinafter **BT**) is designed for bulk data transfer. The file is divided into pieces, which can be downloaded in parallel from peers belonging to a specific *torrent.* A central entity, called *tracker*, keeps track of all peers downloading the content and bootstraps new peers joining the torrent with a random set (of size 50 peers) of remote 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 (termed the piece selection algorithm) and one that is used to select remote peers to upload data to (termed the peer selection algorithm, or the *choke algorithm*). In this work we focus on the choke algorithm, and gloss over the details of piece selection. With the choke algorithm, a node builds a subset of its neighborhood that is termed *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. With the choke algorithm, peers discover and maintain an active set (of size $k + \omega$) composed by neighbors that maximize *reciprocation*, i.e. the amount of data downloaded given the amount of data uploaded to remote peers.

In the basic version of BT, k and ω are empirically set parameters: generally $k = 4$ and $\omega = 1$. The upload bandwidth of a peer is equally split (beside TCP effects) among all unchoked peers. Recently, a new version the mainline BT protocol has been released. Despite its rather small diffusion among users (only 2% of the clients appear to be of type mainline [5]), we analyze in this work the impact of this new client, that we termed **BTnew**. The key difference of BTnew lies in the choice of the parameters of the choke algorithm. The number of regular unchokes is determined as a function of the uplink capacity C of a peer, that is $k = \sqrt{0.6C}$. Moreover, $\omega = 2$. With these new parameters, peers with a high uplink capacity open more active connections.

BitTyrant. The key modifications introduced by BitTyrant (hereinafter **BTyr**) are related to the peer selection algorithm. As for BTnew, 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 to all connections using an empirically set parameter b_i. This initial value is set such that the probability of reciprocation from remote peers is high. The authors in [5] compute b_i considering a bandwidth distribution derived from real measurements: as in this work we adopt the same distribution, we also use the same value for b_i. Subsequently, the alternative 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 stop 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 between the amount of data received and 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 converges to the exact value required to guarantee reciprocation.

3 Revisiting BitTyrant Basics: Theoretical Analysis

In this section we revisit the model proposed in [5] for the standard BT client and study its altruistic nature, where altruism is defined as the difference between the upload and the download rates (if positive) of all peers. As in [5], we neglect the effects of content availability: we assume that each peer always holds interesting data, and we consider infinite peer set size. Our contribution is twofold: (i) we present results not only for BTnew, but also for BT (not considered in [5]), and (ii) we take into account the discovery process by which BT peers can find peers with similar capacity. The details of the analysis are in the appendix.

3.1 Tit-for-Tat Matching Time

Let us consider a set of peers that start the download process without any prior knowledge of other peers equal-split capacity: this knowledge is gradually inferred using *optimistic unchoking*.

At the beginning of the download process, the equal-split probability distribution for the active connections of a given peer coincides with the probability distribution of the equal-split across the whole population. This distribution changes in time for two key reasons: i) once a peer interacts with a number of peers greater than the maximum active set size, it can start choking its worst uploaders; ii) a peer could be choked by some of its best uploaders as they are also involved in the discovery process and may find better peers to reciprocate with. Throughout the discovery process, peers prune and establish active connections so as to find peers with a similar equal-split capacity; authors in [5] term this effect *matching* and define the "TFT matching time" as the time a peer needs to fill its active peer set with remote peers exhibiting equal or greater capacity. As stated in [5], the TFT matching time can be considered as a lower bound for the matching process to converge, because peer churn and content availability are neglected. We further note that the definition of matching time also neglects if any effective reciprocation between peers take place: we address this issue in the following Section.

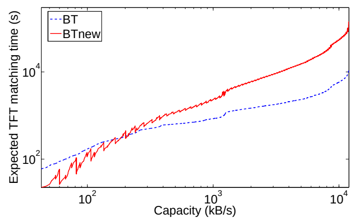


Fig. 1. Time required for a new peer to discover a number of peer of equal or greater equal-split to fill its active set size.

In Fig. 1 we show the TFT matching time for BT and BTnew clients with different uploading capacities. The sawtooth behavior of the BTnew curve is due to some asymmetries that arise among peers with similar capacities which may open a different number of connections, thus exhibiting smaller equal-split capacities: this only slightly affects the discovery process for slow peers. We observe that the matching time increases with the capacity of a peer: it is greater than 1 and 10 hours for BT and BTnew clients (with high capacity) respectively. Moreover, the higher number of active connections established by BTnew account for a non negligible increment in the time to reach a "stable" matching: fast peers systematically exhibit higher discovery times when using BTnew than when the original BT choke algorithm is used.

Interestingly enough, our results pinpoint an increment in the time scale of one order of magnitude with respect to what is discussed in [5]: this is due to a problem in the original analysis, that we have fixed and detailed in the appendix. Our results reinforce the key intuition behind the design of BTyr. The behavior of BitTorrent peers, that explore their neighbors by altruistically uploading pieces of the content to remote peers, can be exploited and the fact that the time for the discovery process to converge can be very long plays in favor of peers adopting the BTyr client. However, the level of reciprocation dictated by the TFT strategy of BT and BTnew clients could have non negligible effects on the actual gain of a BTyr client. We investigate on this aspect in the following.

3.2 Probability of Reciprocation and Expected Download Rate

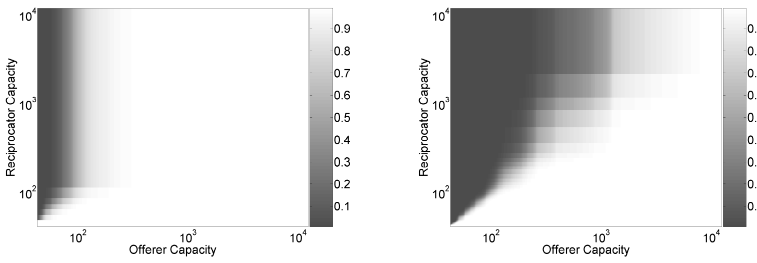
In order to evaluate the degree of altruism of a peer, we need to evaluate its expected download rate. This requires to evaluate the probability of reciprocation of remote peers. Given a peer i, let N_i be its neighbor set, $A_i \subset N_i$ the set of active connections, u_i the equal split capacity peer i assigns to active connections and $\rho_{i,j}$ the probability that peer $j \in A_i$ is willing to reciprocate peer i.

On the basis of the duration of TFT matching time, the authors of [5] derive the reciprocation probability for peer i assuming that every neighbor j is reciprocated from its own neighbors, i.e. $\rho_{j,h} = 1\ \forall h \in N_j$, and that peers in A_j can be considered drawn from the original population distribution, ignoring they have been selected from peer j's peer set. Under such assumptions, peer i has to provide a higher equal-split than the worst uploader of peer j to obtain reciprocation, i.e. $\exists h \in A_j$ such that $u_i > u_h$, and the probability of this event is $\rho_{i,j}$. The expected download rate for peer i is then assumed to be proportional to the the number of active connections ($|A_i|$).

Our analysis accounts for the time required by the discovery process to converge through optimistic unchokes. Hence, 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 and its reciprocation probability fluctuates in time: reciprocation from peers with a higher capacity than peer i decreases (because they may discover similar peers) while reciprocation from peers with a lower capacity than peer i increases (because they are progressively choked by their best uploaders). Since each peer optimistically unchokes ω new peers every $T = 30$ sec., we consider a discrete time system where every $T/(2\omega)$ sec. each peer discovers the equal split capacity of one new peer. The system starts from an initial state where every peer has an empty active set: in this case $\rho_{i,j} = 1$, as any peer can be potentially selected for reciprocation. Then at each step we evaluate a new reciprocation matrix $\rho_{i,j}$ on the basis of the reciprocation matrix at the previous time slot and the new discovered peer. From the reciprocation matrix we can determine the set of reciprocating peers, hence the corresponding aggregate rate using standard order statistic results (see the appendix for details). This approach requires computing the factorial of the number of discovered peers, hence we cannot explore the evolution for a large time interval. However, our analysis clearly shows that a more realistic model of the system exhibits results that are significantly different from those in [5].

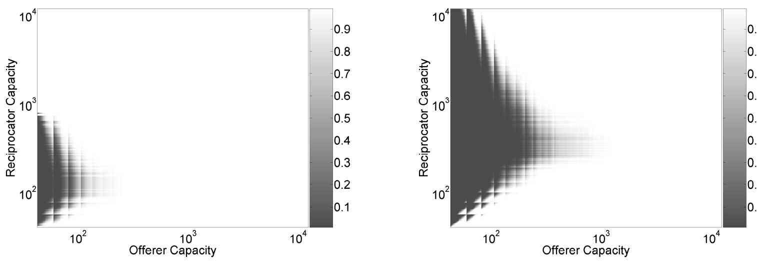
Fig. 2 shows the reciprocation probability for BT clients after 150 seconds and after 15 minutes, corresponding respectively to the time required for each peers to discover the equal splits of 10 and 60 remote peers. Every point (x, y) of the figure indicates the probability that a peer with capacity x (the offerer) is going to be reciprocated by a remote peer with capacity y. After 150 seconds, most of the peers have already discovered enough neighbors to select the best remote peers. After 15 minutes, almost all the peers are willing to reciprocate only with peers with similar or higher capacities.

Fig. 3 illustrates results for BTnew. While BTnew clients discover new peers twice faster than for the standard BT, after 150 seconds high capacity peers are still willing to reciprocate with every other peer, because the number of peers they have discovered is still lower than the maximum size of their active set. After 15 min., instead, high capacity peers reciprocate with peers with lower



(a) After 150 seconds (b) After 15 minutes

Fig. 2. Reciprocation probability for BT.



(a) After 150 seconds (b) After 15 minutes

Fig. 3. Reciprocation probability for BTnew.

capacity due to the high number of connections, i.e. they have not yet found enough (or there are not enough) high capacity neighbors to match with.

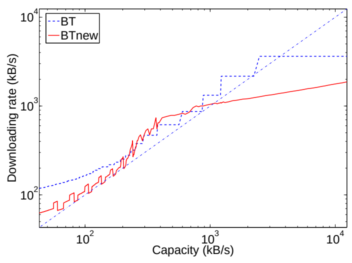


Fig. 4. Expected download rate for a peer of a given capacity after 15 minutes.

The effect of progressive matching on the download rate after 15 minutes is shown in Fig. 4. The total download rate is the sum of the download rate from the active connections the peer has set and of the rate deriving from the optimistic unchoking. The diagonal line in the figure corresponds to a fair scenario where every peer gets a download rate equal to its upload capacity. Focusing on results for BT clients, it appears that low capacity peers are able to gain more than their fair rate: this is not due to the inefficiency of TFT, but rather to optimistic unchoking. Indeed, plotting the rate due to active connections only, would reveal a slope parallel to the diagonal up to about 200kB/s. Peers with a capacity higher than 3000kB/s appear to offer more than they receive due to the lengthy discovery process, hence peers with intermediate capacities can exploit them. However, with time, this phenomenon is less pronounced.

Similar considerations hold also for BTnew clients, with the following differences: the advantage for low capacity peers is less conspicuous than for the BT case because high capacity peers open more connections, hence the download rate allocated to optimistic unchokes is lower; high capacity peers achieve a smaller download rate due to the large number of active connections. While the analysis in [5] was suggesting that only 80% of the peers were able to reach their fair rate, we show that after 15 minutes all the peers with capacity lower than 1000kB/s (according to the used bandwidth distribution, more than 90% of the population) have already reached their fair rate, and this percentage increases over time.

Observations : While our analysis recognizes the possibility for a strategic client to exploit the slow discovery process that affects BT and BTnew clients, we reveal through a more accurate system model that the TFT strategy adopted in the original design of BitTorrent is effective in mitigating the potential abuse of a pronounced altruism. Moreover, our results show that the matching time required by standard BT clients (which still represent the largest component that can be found in the Internet [5]) is shorter than for BTnew clients. Although one would expect BT clients to be more robust to the presence of strategic clients such as BTyr, in what follows we show that in practice this is not the case: a single BTyr client performs better in an environment of BT clients, while it mainly loses any competitive advantage with BTnew clients. The main reason is that content availability is playing a very important role that cannot be captured through our analysis.

4 Deconstructing BitTyrant: a Simulative Study

The results presented in [5] are based on a patched version of the Azureus client with the BTyr choke algorithm and have been collected on real torrents and on PlanetLab. Our goal here is to study, in a controlled simulation environment, the the key building blocks of the BTyr choke algorithm and gain insights on their contribution to the increased performance of the BTyr client.

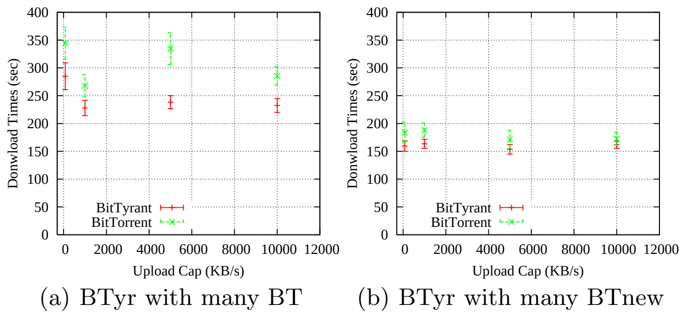
Our work is based on a customized version of the publicly available BitTorrent simulator called GPS [7]. GPS is a discrete time flow level simulator, featuring a simple fluid model of TCP: the available bandwidth between two peers is equally shared among active flows on the path joining the peers. Peers have infinite downlink capacity and a finite uplink capacity, which is distributed as we discuss next. The legacy BT system is implemented, including the piece selection, the choke algorithm and the Tracker. We complemented the simulator with an implementation of the new version of the *mainline* BT client (BTnew) and of the BTyr choke algorithm.

Our simulation settings reproduce the same scenario of [5]: we analyze torrents of 350 nodes where one seed distributes a file of 50 MB. The uplink bandwidth distribution is also the same as [5]. Peers randomly start to download the content within a small interval (10 sec.) of time and stay in the system once they finish downloading the content.

First, we carry out a comparative performance analysis of a single client using whether BTyr or BT/BTnew choke algorithm when the rest of the torrent population use the BT/BTnew algorithms. In a second set of experiments, we analyze the performance of an homogeneous system in which all peers execute the BT, BTnew or BTyr algorithms. The main performance metric we use in this work is the download time distribution for all peers, although many results we show in what follows focus on the single BTyr client. By performing multiple runs (i.e., we generate different random arrival pattern and we select for each pattern a new target peer) we are able to estimate the mean download time, along with the confidence interval for a confidence level of 95%. Other performance indexes used in this work are the number of pieces uploaded by the single BTyr peer and the set of uplink capacities of neighbors that the single BTyr peer unchokes in time.

4.1 Fact 1: Size Matters

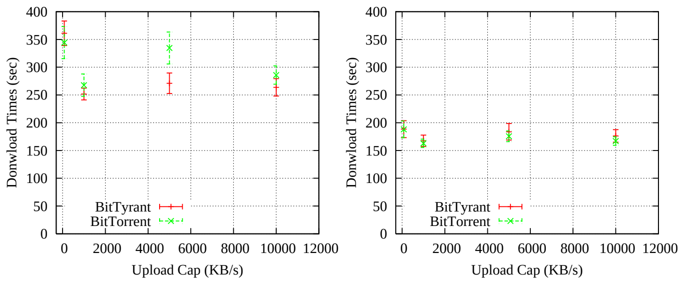
In this section we compare the performance of a single client, using as parameters the peer uplink bandwidth and the choke algorithm used by the peer. First, for validation purposes, we focus on the performance gain when one peer uses the BTyr algorithm as compared to the same peer using the original BT algorithm: Fig. 5(a) illustrates similar findings as for the PlanetLab experiments carried out in [5]. In Fig. 5(b) we show the same set of experiments, but in this case all clients but one adopt the BTnew algorithm. The performance gain of BTyr dramatically drops as compared to Fig. 5(a): this is due to the large number of active connections established by fast peers under the BTnew algorithm; their uplink capacity is over-curved, hence remote peers perceive smaller download rates as compared to the original BT algorithm. This particularly affects the gain of the BTyr client. With respect to the theoretical results obtained in the previous section, we can see how much the content availability (as well as the limited peer set) impact the performance.



(a) BTyr with many BT (b) BTyr with many BTnew

Fig. 5. Mean download time for BTyr and BT for different bandwidths.

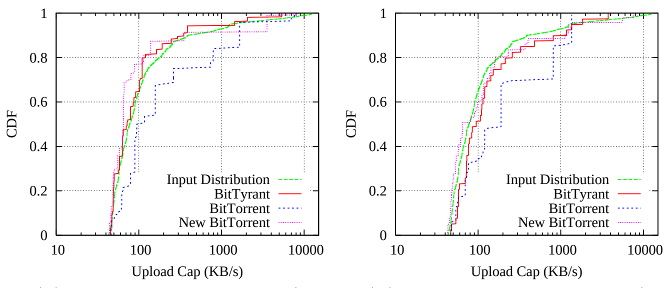
Beside the subtle uplink allocation algorithm that allows BTyr to exploit BT peers, a key point in the design of BTyr that is examined only superficially in [5] is the technique to manage the peer set size, which mimics the one used by another strategic client, BitThief [6]. A large peer set size is maintained by increasing the frequency in contacting the Tracker to obtain new remote peers in the torrent. This simple technique has been proven to increase the probability for a peer to be optimistically unchoked, leading potentially to high download rates even without any contribution to the system. We now isolate the contribution of the peer set size to the performance by forcing a maximum size of 80 peers, as for legacy BT clients. Fig. 6(a) and Fig. 6(b) illustrate the significant loss in performance of BTyr in a torrent of BT and BTnew clients respectively. Interestingly, the latter case indicates that BTyr could completely lose all its benefits.



(a) BTyr with many BT (b) BTyr with many BTnew

Fig. 6. Mean download time for BTyr and BT/BTnew with a constrained peer set.

The performance loss of BTyr further points at the impact of the number of active connections: our simulations indicate that the BTyr client unchokes a very large number of peers, especially at the beginning of the download process. While we further investigate on this phenomenon in the next Section, here we discuss on results obtained when a single BTnew client operates in a torrent of BT clients. Our experiments (we don't show results here due to space constraints) show that a single BTnew client achieves similar performance as BTyr (see Fig. 6(a)).



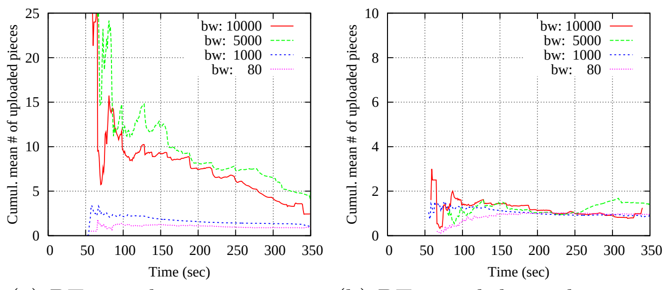
(a) Upload bw: 200 KB/s (b) Upload bw: 1000 KB/s

Fig. 7. CDF of the upload bandwidth of BT, BTnew and BTyr neighbors.

These results hint at the fact that the dynamic uplink bandwidth allocation algorithm adopted by BTyr appears to have little impact on performance. To gain further insights, we analyze the interaction between a single client and its neighbor peers. Fig. 7 shows the CDF of the uplink capacity of the neighbors of a client that use the BT, BTnew or BTyr unchoke algorithms. Similar results hold for a single peer with different upload capacity. We notice that, while a single BT client focuses on a small set of neighbors with well defined uplink capacities, BTyr and BTnew appear to interact with any neighbor, irrespectively of its 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 in a torrent using an intelligent uplink allocation algorithm, no stable matching appear from the results in Fig. 7.

4.2 Fact 2: Even Tyrants Can Be Altruistic

In the previous section we show that the performance gain of BTyr in comparison to BT is essentially due to the larger peer set and active set. While the effect of a larger peer set is well understood, it is not clear the advantage of a larger active set. Splitting the uplink capacity among a large number of connections could be useful, for example, when bottlenecks at the downlink are considered. In this case a fast peer could underutilize its uplink capacity because of downlink limits at its neighbors: opening more connections would alleviate this problem. However, in our simulation setting, the download capacity is infinite. The key aspect here is *content availability*. While the active set size determines the amount of bandwidth allocated to remote peers, it is not granted that they will be able to fully use it during the whole unchoke interval: especially during the initial phase of the download process, the lack of fresh pieces to serve could cause uplink capacity underutilization. This effect is mitigated with a larger active set size: while the per-connection uplink capacity decreases, the number of interested neighbors increases and the total uplink capacity of a peer can be used. As a by-product, peers that successfully upload fresh data, even at slower rates, to a large number of neighbors increase the probability of reciprocation. In BTnew (and indirectly in BTyr) the active set size is set empirically. We leave for future work the evaluation (and the feasibility) of an optimal value.



(a) BTyr with no restriction (b) BTyr with limited peer set

Fig. 8. Time series of the ratio between cumulative uploaded pieces by BTyr and BT.

It is interesting to note that a large active set size has the inherent positive effect of spreading pieces of the content to a large number of peers that would otherwise remain unserved. With a dynamic uplink allocation algorithm that opens many active connections, BTyr reveals to be altruistic and to make an efficient use of its uplink capacity. Fig. 8(a) depicts the ratio between the cumulative number of uploaded pieces over time by single BTyr client with respect to the correspondent BT client. Especially during the early stages of content distribution, BTyr uploads up to 25 times (for a high bandwidth client) the number of pieces uploaded by BT. Subsequently, the total number of pieces uploaded for BTyr and BT converges. This effect is also present, though less apparent, when we consider a peer set size constrained version of BTyr (Fig. 8(b)).

This unexpected altruism of BTyr has a beneficial effect on all nodes involved in the distribution process. In Fig. 9 we show the cumulative distribution function of the download times of all peers. Results show that when even only one fast BTyr peer (with high bandwidth equal to 5000 KB/s) is present in the system, its influence is non-negligible. Interestingly, similar observations can be made when introducing one BTnew client.

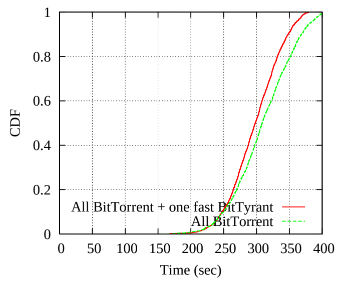


Fig. 9. CDF of download times with or without a single standard BTyr client with bandwidth 5000 KB/s.

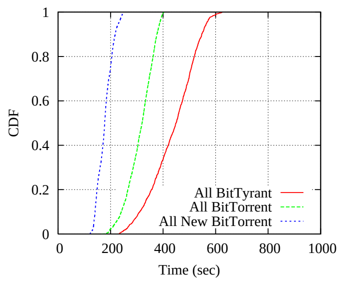


Fig. 10. CDF of download times for BT, BTnew and BTyr.

4.3 Fact 3: Many Tyrants Are Dangerous

Our results indicate that both BTyr and BTnew are potentially attractive for users thanks to their increased performance over the traditional BT. The extreme case of a widespread adoption of these alternative clients is analyzed in what follows. While [5] presents results for a torrent of BTyr peers, we study the case for BTyr peers with a peer set size constraint and for BTnew peers. In light of access control policies implemented in recent Trackers, it is realistic to assume the presence of mechanisms to reject frequent queries made by peers that wish to obtain a larger peer set size.

Fig. 10 illustrates the CDF of the download times for a torrent of BT, BTnew and BTyr peers: a glance at the median and worst case download times indicates that a large-scale adoption of BTyr can jeopardize the content distribution process. In BTyr, remote peers are ranked by the ratio of their download and upload rate, hence absolute contribution levels are lost. The most visible side effect is a race towards lower levels of contribution: we defer a more detailed discussion to Appendix B.

5 Conclusions

Recent days have witnessed the development of new, aggressive peer-to-peer clients aiming at decreasing content download times by leveraging on subtle techniques to exploit generous clients. In particular, this work focus on a detailed performance analysis of BitTyrant, a strategic client that was designed to capitalize on the inherent altruism that characterize the *mainline* version of BitTorrent. BitTyrant has been previously shown to exhibit a substantial performance improvement over BitTorrent, and in this paper we detail the key reasons for its success. This work covers also a new version of the *mainline* BitTorrent and reveals the benefits of a new choice of parameters for one of its key algorithms.

Our results indicate that the uplink allocation algorithm used by BitTyrant, which was originally designed to maximize the performance gain in terms of download times, turns out to be altruistic, especially during the initial stages of the content distribution process. While aggressive clients including BitTyrant and BitThief arrive at improving their download rate by simply increasing the size of their neighborhood, our study shows that the widespread adoption of such clients can severely impact the overall system performance.

Although the latest version of BitTorrent reveals to be beneficial for the entire system, our results indicate that an empirical choice of the key parameters that drive reciprocation is somehow inefficient. We believe that further study is required to design dynamic uplink allocation algorithms that would work for the benefit of all users of a torrent and that would maximize, for any bandwidth scenario, the overall system capacity utilization. The design of a new, strategy-proof client based on such algorithms will be addressed in our future work.

References

1. B. Cohen, “Incentives build robustness in BitTorrent”, in *Proc. of P2P-Econ*, Berkeley, California, USA, June 2003.
2. 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
3. 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
4. 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
5. 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.
6. 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
7. 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

Appendices

A Theoretical Analysis

We adopt the same notation introduced in [5], but we denote with $b(v)$ and $B(v)$ respectively the probability density function and the cumulative distribution function of the equal split (and not of the uploading rate).

Tit-for-Tat Matching Time

During a time interval equal to T, a peer discovers the equal split of ω new peers and its equal split is discovered by other ω new peers. Given peer i with equal split r, let $S(r)$ be the probability that a peer has equal split smaller than r. The expected number of interactions peer i needs to find a peer with the same or higher equal split ("a matching peer") is distributed as a geometric distribution, with expected value:

$$\frac{1}{1-S(r)}.$$

The expected number of interactions needed to discover a number of peers equal to the number of active connections $|A_i|$ is simply:

$$|A_i|\frac{1}{1-S(r)}.$$

If we consider that the peer has one interaction every $T/(2\omega)$ seconds, then the matching time is:

$$\frac{T}{2\omega}\frac{|A_i|}{1-S(r)}.$$

This result is different from the result derived in [5], because of an error in equation (9) of the "Modeling notes" appendix there. In fact $c(r,n)$ defined there is the probability to find at least one peer with equal split higher than r by time nT, i.e. with our terminology in at most $2\omega n$ interactions. Equation (9) erroneously evaluate the probability to find exactly one matching peer at time nT as

$$c(r,n)\prod_{i=1}^{n-1}(1-c(r,i)).$$

This is wrong because: 1) in time interval $((n-1)T, nT]$ up to 2ω matching peers can be found, 2) the probability to find at least one peer in time interval $((n-1)T, nT]$ is simply $c(r,n)(1-c(r,n-1))$.

Probability of reciprocation

In Section 3, we have defined $\rho_{i,j}$ the probability that node j is willing to reciprocate with node i. Being that this probability depends only on their equal splits and it changes after each interaction (we assume that all nodes have an interaction simultaneously every $T/(2\omega)$ seconds, 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. We have $\rho(u_i, u_j, 0) = 1$, being that we consider all nodes starting from a clean slate and hence willing to reciprocate with anyone else.

At each interaction a peer discovers the equal split of another peer. The equal split follows the distribution $b(v)$. The probability that a generic peer in N_i is not willing to reciprocate with 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:

$$\overline{R}_i(k) = k \int_0^\infty \rho(u_i, v, k) b(v) \mathrm{d}v.$$

We simplify our analysis assuming that: 1) the number of peers not reciprocating peer j is always equal to the integer nearest to $\overline{R}_j$ (we denote it as $\hat{R}_j$) and 2) that these peers are the best uploaders of peer j. Then if we rank the uploaders of peer j on the basis of their equal split in decreasing order, peer j at the k-interaction will be willing to reciprocate peers with rank from $\hat{R}_j(k) + 1$ to $w = \hat{R}_j(k) + |A_j|$, being that it is willing to open up to $|A_j|$ connections. Now 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 an higher equal split than that of the w-th uploader of peer j[3]. The probability density function of the equal split of the w-th uploader of peer j is:

$$b_{u_j}^{(w)}(v, k) = \frac{k!}{(w-1)!(k-w)!} B(v)^{(k-w)} (1 - B(v))^{(w-1)} b(v),$$

and the reciprocation probability at the $k+1$-th interaction is:

$$\rho(u_i, u_j, k+1) = \int_0^{u_i} b_{u_j}^{(w)}(v, k) \mathrm{d}v.$$

This is the recurrence equation we can use to evaluate the evolution of reciprocation probability over time.

Expected Download Rate

The expected download rate of peer j can be derived as:

$$\sum_{h=\hat{R}_j(k)+1}^{\hat{R}_j(k)+|A_j|} \int_0^\infty v b_{u_j}^{(h)}(v, k) \mathrm{d}v + \omega \int_0^\infty v b(v) \mathrm{d}v,$$

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

[3] If $w = \hat{R}_j(k) + |A_j| > k$, peer j will be always willing to reciprocate with a new peer.

B Analysis of the BitTyrant Algorithm

In this section we investigate the performance of the BTyr algorithm in a homogeneous environment, i.e. when all nodes are BTyr. In particular we neglect the effect of content availability: we assume that each node has always interesting pieces to give to any other nodes. This allows us to focus on the interactions among nodes, in particular on the unchoked relationships.

To this aim, we design a simulator that run for each node the choking algorithm, considering, instead of the amount of data received and sent in the previous choking intervals, the bandwidth assigned to the connection in both ways. Before describing the simulator and the results, we first argue why the case of all BTyr client is of interest.

B.1 Game Theoretic Approach

In [5] the authors shortly discuss the case of spread adoption of BitTyrant: they suggest that performance degradation can actually occur. The intuition is that BTyr tries to minimize the contribution to each neighbor, thus when two BTyr nodes interact, they might start decreasing the mutual sending rate.

In order to solve this problem, the authors propose a different strategy: when two BTyr nodes start exchanging data, they should switch to a block based TFT strategy and, at the same time, they should try to increase the rate, instead of decreasing it.

We believe that such a behavior – using different strategies according to the client the node is interacting with – can never occur. The simple reason is that a BTyr client that adopts the same strategy independently from the type of the other clients (*static BTyr*) will gain with respect to a BTyr that adapts the strategy (*adaptive BTyr*). In fact, the adaptive BTyr tries to increase the bandwidth, while the static BTyr decrease it, obtaining a better performance with respect to the adaptive BTyr. In other words, there is no incentives to be adaptive, and each client behaves selfishly remaining static.

On the one hand, BTyr will be adopted instead of BT, since there is an incentive to adopt BTyr with respect to BT; on the other hand, the BTyr client will be necessarily static, i.e. they adopts the same behavior independently from the neighbor types, since there is no incentive to be adaptive. The final situation is a population with all (static) BTyr clients. For this reason it is correct to study the case with all BTyr, since it is the evolutionary path followed by clients.

B.2 Simulator Description and Settings

The simulator we developed focus on the unchoking relationships among nodes. At every choking interval, a BTyr node unchokes a subset of its neighbors, assigning to them a rate. Neglecting the influence of the content means assuming that during the choking interval each node sends data to each unchoked neighbor with the rate assigned to it during the last run of the choking algorithm (i.e. the rate is fully utilized). This condition represents an ideal situation that in practice is unlikely to happen: nevertheless, with such an approach, we are able to evaluate the dynamics of the relationships, and in particular the evolution of the rate assigned by the BTyr choking algorithm.

Once the content is neglected, the operations performed by each node are very simple. At each choking interval each node performs the BTyr choking algorithm, considering the ratio between the received rate and sent rate for each neighbor (instead of the ratio between the received data and sent data). Then it unchokes the neighbors with the best ratio until it has available upload bandwidth, as in the usual BTyr choking algorithm.

The remaining behavior of the system remains unchanged: all the nodes arrive uniformly at random in a interval of 10 seconds and the overlay is built with the help of a tracker. We assume that BTyr nodes has standard peer set size and nodes remain always online. All the other parameters – initial value of the rate assigned the first time to neighbors, choking interval, distribution of the upload capacities – are same as described in Sect. 4.

B.3 Results

The main output of the simulator is the rate assigned to each neighbors over time. Each node creates a matrix where the element e_{ij} represents the rate assigned to peer i at choking interval j. In order to show the basic behavior of BTyr choking algorithm, we select three representative nodes with upload capacity 80, 200 and 10000 respectively.

In Fig. 11 we show the matrix of the rates for a node with 10000 KB/s upload capacity. The value of the assigned rate is represented with colors: the darker the color, the higher the rate. At the beginning, the node assigns the same rate to all its neighbors[4]. Note that, with an initial rate of 15 KB/s assigned to each unchoked neighbors, the node has to unchoke *all* its neighbors; moreover, the total amount of assigned bandwidth is in any case much lower than the upload capacity (1500 KB/s).

It is possible to distinguish two different neighbor subsets: some of them have a rate with a downward trend; others have a rate with an initially upward trend, then periodic trend. The risk of neighbors with downward trend was implicitly foreseen by authors of [5] – for this reason they argue about adaptive BTyr nodes. The interesting behavior is represented by the upward and periodic trend. This behavior can be explained considering that the node has spare upload capacity and thus it continues to unchokes *all* its neighbors. On the other hand, its neighbors may have limited capacity, thus they choked the node we are considering. Since the node is choked, it applies the BTyr algorithm and it increases the rate. This behavior is visible in the first part, up to 20-30 rounds.

At this point the rate is so high that there are two effects: first, the node starts choking some of the neighbors, since it does not have enough capacity for all of them; second, the neighbors start unchoking the node we are considering. These two effects concur in creating the periodic behavior. In order to fully understand the reason of this behavior, we have to consider the nodes with low upload capacity (that represent the majority of the neighbors of a high upload capacity node). Figure 12 shows the matrix of the rates for a node with 200 KB/s upload capacity and a node with 80 KB/s upload capacity.

Here we see that the prevalent trend is the periodic one. The period in all cases is equal to three rounds. This value is a consequence of the period used

[4] Here we fix the maximum number of neighbors equal to 100

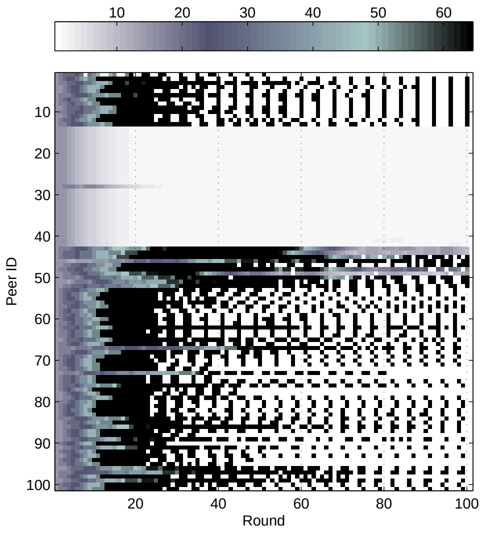
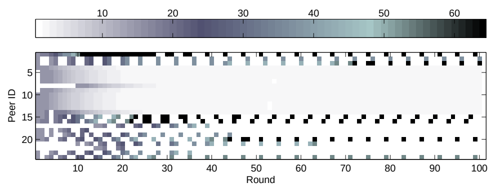


Fig. 11. Uploaded neighbors by a single fast BiTyrant client (with all BTyr clients).

by BTyr (and BT) when evaluating the received/sent rate – two rounds: the algorithm evaluates the data received and sent in the last two rounds. For instance, assume node n_1 that performs the choking algorithm at time t. If node n_1 has not sent any data to a node n_2, and node n_2 has sent even an infinitesimal amount of data to node n_1, the ratio between the received data and the sent data for n_1 will be infinite, independently on the received data. This means that node n_2 will be unchoked every three rounds for just one round, since the third round node n_1 loses memory of the given data but still receive something from n_2. This can be applied to all nodes, with a final periodic trend for most of the nodes.

An interesting observation related to Fig. 12 is the explored peer set: the node continues to deal with the same 20-25 neighbors, without exploring the other neighbors. This is an undesirable behavior, since nodes are not able to discover better neighbors. We used the same simulator to analyze the behavior of BT, and we observed that, as the time goes by, more and more neighbors are unchoked at least for three rounds (thanks to the optimistic unchoking). Thus the number of rows of matrices (such as the ones shown for BTyr) tends to becomes equal to the maximum number of neighbors. The fact that the BTyr algorithm is not able to properly discover better neighbors represent another contribution to the bad performances of BTyr when all the other nodes are BTyr.



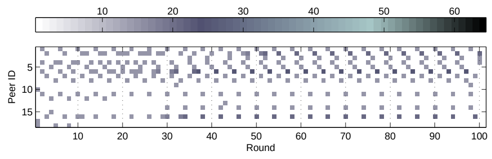


Fig. 12. Uploaded neighbors by a single medium and slow BiTyrant client (with all BTyr clients).