Uplink Allocation Beyond Choke/Unchoke

or Why Divide Does Not Always Conquer Best

Nikolaos Laoutaris
Telefonica Research
nikos@tid.es

Damiano Carra
INRIA - Sophia Antipolis
dcarra@sophia.inria.fr

Pietro Michiardi
Eurecom
pietro.michiardi@eurecom.fr

ABSTRACT

Motivated by emerging cooperative P2P applications that go beyond file-sharing, we study new uplink allocation algorithms for substituting the rate-based choke/unchoke algorithm of BitTorrent which becomes inefficient in these cases. Our goal is to reduce further the download times. We do so by improving the uplink utilization when it is mostly challenged: in young torrents, and when there exist downlink and network bottlenecks. We develop a new family of uplink allocation algorithms which we call *BitMax*, to stress the fact that they allocate to each unchoked node the maximum rate it can sustain, instead of an $1/(k+1)$ equal share as done in the existing BitTorrent. BitMax computes in each interval the number of nodes to be unchoked, and the corresponding allocations, and thus does not need any empirically preset parameters like k. We demonstrate experimentally that BitMax can reduce significantly the download time in a typical reference scenario involving mostly ADSL nodes. We also consider scenarios involving network bottlenecks caused by filtering of P2P traffic at ISP peering points and show that BitMax retains its gains also in these cases.

Keywords

Peer-to-Peer, BitTorrent, uplink bandwidth allocation

1. INTRODUCTION

BitTorrent [1] (henceforth BT) has been enjoying phenomenal growth over the last few years and is now believed to amount for a substantial percentage of Internet's traffic [2]. Evident to its success is the fact that several ISPs have started rate-limiting BT-generated traffic at their peering points to prevent it from disrupting their economics [3]. As elaborated in Sect. 2, several innovative aspects allow BT to carry so much data, including its ability to perform perfect discovery of content, the adoption of parallel downloads from multiple nodes, and the enforced contribution during participation. The achieved download times depend largely on the uplink utilization level that the protocol can achieve for the participating nodes under the particular operating parameters of each torrent.

Uplink utilization in BitTorrent: Uplink utilization depends largely on the choke/unchoke algorithm used in BT for deciding which nodes get to have their requests for missing chunks honored. The algorithm operates as follows. Every 10 sec a node "unchokes" the $k = 4$ nodes[1] which have provided it with the highest download rates over the previous 20 sec interval. Each one of the unchoked nodes is given an equal share of the node's upload capacity which it can use for pulling missing chunks. Also, every 30 sec, a randomly picked choked node is unchoked for the next 30 sec interval, independently of the download rate it might have provided. These so called optimistic unchokes are used for discovering new nodes that can support high transfer rates. Several measurement [4, 5, 6] and simulation [7] works have reported that the choke/unchoke algorithm achieves above 90% utilization of upload bandwidth once it reaches steady-state, after an initial "flash crowd" phase in which utilization is low. Most studies have focused in torrents composed mainly of ADSL users with heavily asymmetric upload/download rates. In addition, analytic works [8, 9] have verified the above results by showing that simplified contact processes that capture some of the characteristics of BT are asymptotically efficient in steady-state in large networks.

Open questions: Despite this substantial body of work, there still exist some important questions on uplink utilization that have not gotten a definitive answer yet. First and foremost, there seems to be no clear justification behind the choice to unchoke $k = 4$ nodes at each unchoking interval. The aforementioned experimental studies have reported that 4 seems to be working well after the startup phase in the typical ADSL scenario. But is 4 a magic number that can guarantee high utilization independently of network size and upload/download capacities? Could a different value

[1] The value 4 is the default one given in [1] and used until recently in the reference *mainline* BT client, as well as in popular clients like *Azureus* and *μTorrent*. Different default values have appeared in some versions that the user can override. There does not exist,however, any kind of guideline on how to make a better choice, and thus users either stick to the default value, or experiment on a trial and error basis.

prove more efficient during the initial phase of a torrent when utilization is low, or for that matter, why should k be independent of the torrent's lifetime (or state)? Such questions has been posed in the past [7, 10], but were never answered.

Moving a little further, even if someone could make the case for a way to set k, it would still remain an open question whether splitting the uplink capacity equally is a good choice or not. One could for example ask: "if the objective is to optimize the social utility of the group, then why not give a higher uplink endowment to nodes that are themselves fast uploaders?". The answer the above questions depends on a huge parameter set that includes: the size of a swarm and of individual neighborhoods, the amount of asymmetry between uplink and downlink, the distribution of node capacities, and finally, the existence of additional network bottlenecks beyond the access links. The latter has become a timely consideration in view of recent news that several ISPs have started rate-limiting the amount of BT traffic that is allowed to cross their peering points [3] (previous works like [4, 7, 5, 6] that have reported high uplink utilization have assumed only access link bottlenecks).

Our contribution: In this paper we address the above questions by focusing specifically on the uplink utilization problem. Our target is to see if we can further decrease download times through improved uplink scheduling. This requirement stems from the fact that we are motivated by new applications in which, unlike standard P2P file-sharing, peers are cooperative. One such example is given in [11]. It involves edge devices like set-top-boxes, home gateways, etc., that belong to the same ISP or service provider, and confederate by pooling their excess resources in order to offer an alternative to capital-intensive/energy-consuming monolithic datacenters. Another example is the case of P2P clients participating in a private torrent that is set-up by a system administrator to disseminate a software update or a new application to all the computers of a large organization. In all these settings, the first and only priority is the maximization of the aggregate system performance and therefore, the choke/unchoke algorithm becomes redundant since it makes performance sacrifices to protect from selfishness, which is not an issue in these cases.

For this reason, we propose uplink allocation algorithms that go beyond choke/unchoke, and study them under standard and new operational settings that haven't been studied in previous works. Our main outcome is a generalized family of uplink allocation algorithms for deciding (1) how many nodes to unchoke at each unchoking interval, and (2) what percentage of the available upload capacity to allocate to each unchoked node. We label these new algorithms under the term – *BitMax* – and show experimentally that they can provide for a substantial reduction of download times, while also remaining nearly as simple as the standard choke/unchoke algorithm. BitMax uplink allocation is build around the following guidelines:

– *The uplink should be partitioned minimally:* When a node is unchoked, it should be allocated exactly the maximum rate it can sustain. The underlying principle is to saturate the uplink bandwidth of a node by "focusing" it to as few neighbors as possible. The idea is to give them a chunk as fast as possible, and then redirect the uplink to other nodes if necessary, while the original ones can start utilizing their own uplinks for pushing further the newly received chunk. Focusing the bandwidth is the exact opposite of what the "anti-social" BitTyrant client [6] is doing (over-partitioning the uplink to maximize the number of reciprocating peers).

To make the point more clear, consider the following simple example with 3 nodes. Node v has upload rate 1 and wants to upload a chunk of size 1 to nodes v_1 and v_2. Assigning rate 0.5 to each one gets the chunk to both nodes after 2 time slots. Alternatively, assigning the entire rate of 1 to v_1 for 1 time slot and then redirecting it to v_2 has the same end result – both v_1 and v_2 have the chunk at the end of time slot 2. The latter, however, more "focused" allocation, has the added advantage of permitting node v_1 to start contributing to the network by uploading the new chunk earlier, with the beginning of slot 2. This becomes very important during the initial phase of a torrent when most nodes suffer from low chunk availability and can thus underutilize their uplink severely, unless they are provided with new chunks as fast as possible.

– *The allocation should be constrained minimally:* Splitting the uplink into k equal shares works well for some scenarios but it is certainly not optimal for large parts of the parameter space. Therefore we propose a generalized allocation of bandwidth that does not depend on any fixed k or need to abide to equal-share endowments. We decide the allocation at the beginning of each unchoke period, attempting to fulfill the following objectives: (1) saturate the uplink of the node, (2) maximize its "focus", (3) give preference to nodes with high upload capacity and low availability of chunks as they run the highest risk of underutilization due to low chunk availability. We meet these objectives by modeling the allocation of uplink bandwidth as a fractional knapsack problem [12]. Computing an optimal solution to this problem is as complex as the standard unchoke algorithm. Applying the derived uplink allocation leads to significant performance benefits, as outlined below:

- On typical BT settings that we examined, involving mostly ADSL nodes and few fast peers, BitMax is able to reduce the median and the worst case download time by a factor of 2.
- Adding throttling of BT traffic at ISP peering points

hurts both BT and BitMax but the latter still outperforms the formed by a factor of 2.

- The performance gains are independent of whether leechers stay or leave after becoming seeds. This is due to the fact that under BitMax nodes complete their downloads not far apart in time.

Overall, BitMax outperforms the existing unchoke because it strives consciously towards the optimization of the social utility of the network, while at the same time caters to the download rates of individual nodes through natural load balancing rules. Therefore, it avoids getting trapped into an unnecessary campaign against selfishness through tit-for-tat that would hurt the social utility. BitMax uplink allocation is readily applicable in settings in which peers work towards a common goal, like the aforementioned private torrent or the system described in [11]. For standard P2P file-sharing that involves selfish peers, BitMax could be realized only through the adoption of "closed" clients that users download and use without being able to modify (similarly to the way *Skype* works, see Sect. 5 for details).

The remainder of the article is structured as follows. In Section 2 we briefly overview some of the properties of BT and present a simple example to demonstrate that partitioning the uplink slows down the speed with which new chunks spread in the network. In Section 3 we detail and justify the allocation of uplink under BitMax. Section 4 quantifies the advantages of BitMax through extensive performance evaluation based on simulation. Section 5 discusses implementation and adoption issues for clients using BitMax allocation. Section 6 reviews recent related work and Section 7 concludes the article.

2. BACKGROUND

In this section we first briefly introduce some of the key enablers behind BT's success (more details can be found in [1, 13]). We then motivate the use of "focused" bandwidth allocation by studying a simplified example that we can evaluate numerically without needing simulation.

Key innovations of BitTorrent: The following have contributed to BT's data dissemination efficiency.

– Perfect discovery: BT trackers know exactly which peers are downloading a file (called "leechers") and which already have it entirely (called "seeds"), and thus can directly point new comers to the appropriate peers. Compared to 1st generation flood-search approaches (Gnutella), the centralized directory kept at the trackers guarantees the best possible discovery, while minimizing the overhead of searching for content. Compared to 1st generation index-based discovery, either centralized (Napster) or distributed (KaZaA), BT achieves its scalability by providing at least one dedicated tracker to each file.

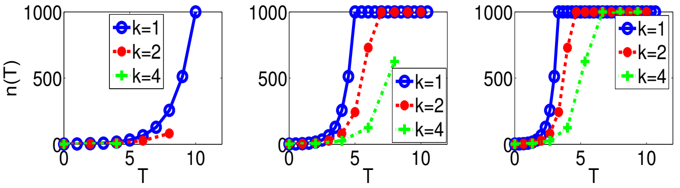
– Parallel downloads from multiple peers: In BT a file is cut into multiple 256 kBytes chunks, allowing a peer to download different chunks in parallel from multiple other peers in its "neighborhood". This increases the perceived download rates while improving the resilience against churn (unexpected disconnection of an uploader before its downloaders have finished receiving the file).

– Enforced contribution: In 1st generation P2P it was very common that nodes would free-ride [14] by disconnecting from the network immediately after they finished downloading the file. This meant that in systems in which nodes needed to have the entire file before they could assist in uploading, free-riders would contributed almost zero capacity. BT works around this problem by using small chunks and thus permitting nodes to start contributing actively early on, with the completion of their first chunk. In addition, the employed choke/unchoke algorithm, creates a tit-for-tat environment in which peers are incentivized to upload chunks to others in exchange for faster downloads. Thus, even if nodes disconnect immediately after receiving the entire file, they still have to contribute for at least as long as the duration of their participation.

In this paper we will show that alternative unchoke algorithms that utilize the above enablers are in position to provide significantly shorter download times than the standard one that uses an empirically set fixed k and equal-share allocations. We start with an idealized example in which we show that it is preferable to saturate the uplink using the minimum possible k (in the idealized case $k = 1$). The intuition from this example drives our generalized design in Sect. 3 whose performance is evaluated under various realistic settings in Sect. 4.

Motivating a "focused" uplink allocation: In the following we will demonstrate why it makes sense to allocate nodes the maximum rate they can download instead of allocating less and increasing the number of parallel unchokes. Consider n nodes, each with upload capacity u and download capacity $d : d \gg u$. Let c denote the size of a chunk. Since nodes split their upload capacity into k equal parts and unchoke k out of their w neighbors, it requires kc/u time to upload a complete chunk to a neighbor. We ask the following question: How many nodes can get a new chunk in T time units, starting at time 0 with a single seed?

We look at the idealized case in which nodes give priority to our tagged chunk over other missing ones (e.g., because it is rare or because it is new). Then, owing to the assumed common capacities, uploads will complete at discrete points in time (iterations), at $kc/u, 2kc/u, \ldots$ Let $n(i)$ denote the number of nodes that have the chunk at the beginning of the ith iteration ($n(0) = 1$). To write $n(i + 1)$ in terms of $n(i)$, we first compute the probability $p(x|n(i))$ that x out of the w neighbors of a node already have the chunk at iteration i, granted that



(a) $u = 1$ (b) $u = 2$ (c) $u = 3$

Figure 1: Number of nodes with the chunk at time T under different k for a network of $n = 1000$ nodes, chunk size $c = 1$ and different upload capacity u.

$n(i)$ have it in total at i. If we assume that the $n(i)$ copies are randomly distributed at the various nodes (which is approximately true for standard trackers that return random neighbors) then $p(x|n(i))$ obeys the following Hyper-Geometric distribution [15],

$$p(x|n(i)) = \text{HyperGeo}(x, n-2, n(i)-2, w),$$

which gives the probability of selecting x "successes" when drawing randomly w samples from a pool of $n-2$ items, out of which $n(i)-2$ are "successes" (the -2 because the node that is uploading is one of the successes and the seed is a second one). Using $p(x, n(i))$, we can write $n(i+1)$ as follows:

$$\begin{aligned} n(i+1) = \quad & n(i) + \\ & n(i) \cdot \left(k \sum_{x=0}^{w-k} p(x|n(i)) + \sum_{x=w-k+1}^{w} (w-x) \cdot p(x|n(i)) \right) \end{aligned}$$

The formula captures the fact that each one of the $n(i)$ nodes with the chunk will spawn k new copies over the next iteration if it can, i.e., if at least k of its neighbors don't already have it. Otherwise, it will spawn as many as its neighborhood status permits. In Fig. 1 we plot the number of nodes with the chunk at time T for $n = 1000$ and $c = 1$ for different k. Figure 1(a) shows that for upload capacity $u = 1$, using $k = 1$ manages to get the chunk to all nodes at $T = 10$, at which point $k = 2$ and 4 are still[2] well below 100. Even for $u = 3$ (Fig. 1(c)), which amounts to quite high upload bandwidth, enough to transmit a 256 Kbytes chunk to 3 neighbors in one unchoke period (10 *sec*), $k = 1$ requires almost half of the time required by $k = 4$.

The above example shows that over-partitioning the uplink slows down the speed at which fresh chunks reach nodes in need of them. In one sense, requiring too much time to transmit a single chunk to a node goes against the very basic upload-while-download philosophy of 2nd generation P2P, pointing more to 1st generation systems in which you had to wait too long (the entire file to arrive) before being able to upload. Our new algorithms described in the following section try to focus the bandwidth and thus increase the torrent's efficiency, especially during its early stage.

3. BITMAX UPLINK ALLOCATION

In this section we present *BitMax*, a family of algorithms for optimizing the allocation of uplink bandwidth in swarm-based systems.

3.1 Knapsack formulation

Consider node v with nominal uplink capacity $u(v)$. Let $V(v)$ denote the set of v's neighbors that are interested on at least one of its chunks and let $u(v, v_i)$ denote the maximum upload rate from v to its neighbor $v_i \in V(v)$. $u(v, v_i)$ depends on: (1) the nominal rate of all links on the IP path from v to v_i, and (2) the cross-traffic on these links. Thus it captures all of the following: the uplink and downlink capacities of the end nodes, potential network bottlenecks due to congestion or filtering of P2P traffic at ISP peering points, the current download saturation of the target node v_i due to participation in the current or other torrents. Define $U(v_i, h)$ to be the utility to the network if v uploads at full rate $u(v, v_i)$ to v_i for a duration ("horizon") h. The utility $U(v_i, h)$ will be defined precisely on Sect. 3.2.

The allocation of uplink rate in BitMax is decided based on the solution of a *fractional knapsack problem* [12] involving $|V(v)|$ "items" v_i, each with utility $U(v_i, h)$ and weight $W(v_i) = u(v, v_i)$, and a "knapsack" with capacity $u(v)$. Selecting items greedily according to decreasing normalized utility $U(v_i, h)/W(v_i)$ is known to provide an optimal solution to such a knapsack. Let $S \subseteq V(u)$ denote the set of nodes that make it into the knapsack in an optimal solution. All nodes in S are allocated their maximum attainable rate $u(v, v_i)$, with the possible exception of the last (partially fitting) one that is allocated the remaining upload capacity up to the saturation[3] of $u(v)$. In the general case this is less than the maximum attainable rate from v. A selected node v_i uses the allocated rate $u(v, v_i)$ to download chunks at full rate from v *but looses its allocation if upon the completion of a chunk fails to request a new one.* In such cases, the unutilized bandwidth is re-allocated to nodes in $V(v) \backslash S$ according again to their normalized utility (i.e., it goes to the next most valuable nodes that didn't make it into the knapsack in the first draft).

[2] The time for an iteration depends on k and thus the tics of different curves do not align. A direct consequence is that the last tic of some curves with $k = 2$ or 4 is before $T = 10$ due to a last iteration that does not complete before $T = 10$.

[3] To use the full nominal rate of the uplink, one must also consider some underutilization introduced by TCP's congestion control mechanism. In [16] it has been shown that a small number of parallel TCP sessions (2-3) is enough for capturing most of the capacity of a link.

3.2 Utility functions

We examine two families of utility functions with different information requirements.

Bandwidth-based: In this case the utility is defined based solely on bandwidth quantities as follows:

$$U(v_i,h)=\begin{cases} h\cdot u(v,v_i) & \text{if } h\cdot u(v,v_i)\leq c \\ h\cdot u(v,v_i)+ & \\ \quad\left(h-\dfrac{c}{u(v,v_i))}\right)(u(v_i)-\lambda(v_i)) & \text{otherwise} \end{cases}$$

The definition requires some explanation. The first part is just the maximum amount of new information that can be put from v to v_i during the horizon h (we set h to 10 seconds to match the duration of unchoke intervals in BT). The second part materializes when at least one complete chunk can be put into v_i during h. In that case, the utility is incremented by $(h-c/u(v,v_i))\cdot(u(v_i)-\lambda(v_i))$. The first parenthesis is the time during which v_i will surely have at least one new chunk to serve to its own neighbors, assuming that this first chunk from v to v_i is also of interest to enough of v_i's neighbors to saturate its uplink. The second parenthesis captures the maximum rate at which v_i can push to its neighbors new information that it gets from v – it is approximated by taking the difference between v_i's nominal upload capacity $u(v_i)$, and its aggregate incoming rate $\lambda(v_i)$ from all other nodes but v (negative values are ok as explained shortly).

Bandwidth/Interest-based: In this case, the utility function factors in also the amount of interest from neighbors of v_i. The basic idea is that if $v_i, v_j \in V(v)$ have equal download rates from v and also equal upload capacities, then v will choose whichever one is least interesting to its own neighbors – this is the node that has a more immediate need for new chunks to avoid underutilizing its upload capacity. We denote $\eta(v_i)$ the amount of interest to v_i from its neighbors $V(v_i)$ and quantify it as follows:

$$\eta(v_i)=\frac{1}{|V(v_i)|}\sum_{v_j\in V(v_i)}\left|C_{v_i}\setminus\left(C_{v_i}\cap C_{v_j}\right)\right|$$

where C_{v_i} denotes the set of chunks held by v_i, also known as its *bitfield*. We use the interest to normalize the second part of the previous bandwidth-centric utility function.

$$U(v_i,h)=\begin{cases} h\cdot u(v,v_i) & \text{if } h\cdot u(v,v_i)\leq c \\ h\cdot u(v,v_i)+ & \\ \quad\dfrac{\left(h-\frac{c}{u(v,v_i))}\right)(u(v_i)-\lambda(v_i))}{\eta(v_i)} & \text{otherwise} \end{cases}$$

Justification: By unchoking nodes $S\subseteq V(v)$ that maximize $\sum_{v_i\in S}U(v_i,h)$ node v attempts to contribute to the maximization of the social utility of the network. The approach is distributed, as each node has control only of its own uplink, and of course heuristic, as it is impossible to write precisely the contribution of v's actions to the social utility of the network – the extent of this contribution jointly depends on unknown present and future decisions of other nodes. Nevertheless, our utility functions try to take actions towards the general "right direction": Neighbors that can download fast are preferred and are allocated exactly the maximum rate they can sustain under the current uplink, downlink and network bottlenecks. Thus the uplink of a node is saturated using the minimum possible amount of partitioning, as justified earlier in Sect. 2.

Besides being sensitive to how fast v_i can download from v, the allocation jointly factors in how fast v_i can in turn forward to nodes in $V(v_i)$ the chunks it gets from v. This rate is estimated by subtracting from v_i's nominal upload capacity $u(v_i)$, its current aggregate reception rate $\lambda(v_i)$ from nodes other than v. A low or negative value indicates that v_i is already receiving new chunks fast enough from other nodes and thus should not be allocated additional bandwidth from v. When $\eta(v_i)$ is also known, normalizing by it improves our estimate of which node is in the biggest need for new chunks. With these simple load-balancing heuristics our allocations attempt to make sure that in the process of optimizing the social utility of the network, no node is left too much behind in terms of reception rate.

In terms of computational requirements, BitMax requires sorting the neighbors according to normalized value once per unchoking interval and, therefore, is as complex as the standard choke/unchoke that requires sorting according to reception rate. The additional remote information required for executing BitMax ($u(v_i)$'s and $\lambda(v_i)$'s) can either be reported by nodes, or be inferred, depending on the assumed setting, as elaborated later in Sect. 5.

4. EVALUATION

4.1 Preliminaries

All our evaluations are based on flow-level discrete event simulation conducted using the publicly available peer-to-peer simulator GPS [17]. We patched the BT protocol implemented in GPS with minor bug fixes, and implemented the two versions of BitMax uplink allocation described in Sect. 3 (labelled MAX1 and MAX2 respectively). We also implemented a standard tracker that bootstraps new peers by providing them with randomly generated lists of 50 neighbors.

Peer model: Like most previous work [7, 18], we focus on peer arrivals that correspond to a *flash crowd*, and thus all of our peers are assumed to be arriving uniformly at random within a small initial interval of

10 seconds. Regarding peer departures, we considered two cases: peers that stay connected until the end of the simulation, and peers that disconnect immediately after becoming seeds. In our experiments we explored the following settings: a total number of peers in the set $n = \{50, 100, 500\}$, and a standard file sizes $FS = 500$ MB. If not otherwise stated, in the following we present results for $n = 100$ peers, and a *single initial seed*. We chose this torrent size because it appears frequently on real measured torrents [13]. Furthermore, the torrent size is large enough to avoid having all the peers neighboring the initial seed, which is the typical case for very small torrents. It has been shown in [5], and also discussed later, that BT fares very well in that case.

Physical network model: The underlying physical network is modeled at the IP-level using synthetically generated "transit-stub topologies" from BRITE [19]. The core of the network consists of 5 transit nodes (representing different ISPs) forming a complete graph. Each transit node has attached to it 200 stub nodes, representing residential ADSL connections. We placed our n peers randomly on these stubs and assumed uplink capacities that are in line with previous works, e.g., [7, 6]. Specifically, we used the same uplink capacity distribution of [6], that has been obtained through real-life measurements over a large set of torrents. Such a distribution corresponds to a commonly assumed population of mostly slow and fast ADSL mixing with few very fast peers connected through corporate and university networks. In all simulations we used a fluid model of TCP in which different competing flows receive equal shares of the link's capacity.

Bottleneck scenarios: In the least constrained scenario (*caseU*), we consider only uplink bottlenecks distributed as above. Downlink capacities are assumed to be infinite in this case. We complement this basic scenario with two additional ones. In the first (*case UD*), we constrain the downlink capacity of a peer to be a multiple of the uplink capacity (in our experiments we assume a multiplicative factor of 3.5, see [7, 6]). In the second (*case UDN*), we introduce additional bottlenecks at the core of the network in order to model the effects of P2P traffic-throttling at ISP peering points. We implement these as follows: Connections between peers belonging to the same ISP behave as in *case UD*, whereas connections that cross ISP boundaries are subjected to throttling which we simulate by constraining the total available bandwidth to them at the peering point. We explore different throttling intensities, representing 10%, 20% and 40% of the average peak-traffic that would be required under *case UD*. We obtained such peak demand by simulating *case UD* and computing the traffic matrices of the different ISPs (using multiple runs to remove noise). From these matrices we estimated the peak demand at each transit link. Taking 10%, 20% and 40% percentages of this peek gave us the nominal rates of 2 Mbps, 4 Mbps and 8 Mbps, which we used as bottleneck capacities of the transit links in order to simulate the effect of throttling. Bindal et al. [2] have used similar values in their work on locality-aware tracker design.

Metrics: We now define our evaluation metrics.

- **Download time**: This is the most natural metric for assessing the effectiveness of a torrent. We report the cumulative distribution function of the time required by different nodes for downloading the entire file. To ease the comparison between different policies, we also present side by side the most important percentiles.

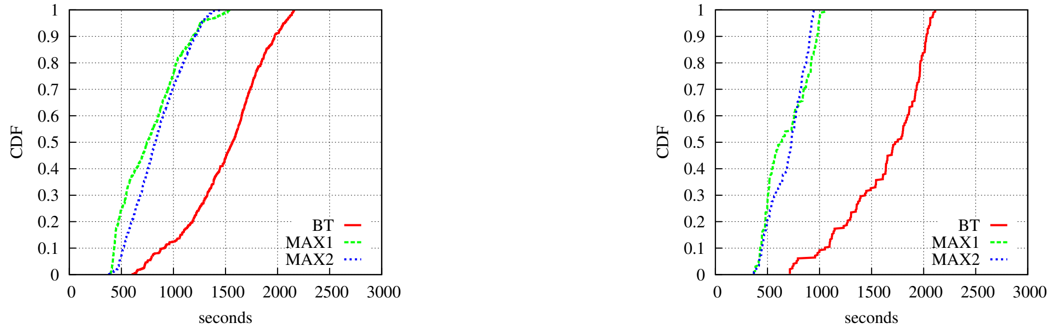
- **Service capacity**: This metric captures the utilization of the aggregate uplink capacity. In this work, we focus on absolute values: We compute the service capacity by counting the number of chunks (of size 256 KB) uploaded in total in the network over a time window of 10 seconds (*i.e.*, in the duration of a standard unchoke interval) and express it in Mbps.

- **Average number of unchoked peers**: We compute the time-series of the average number of unchoked peers to study the effects of the dynamic uplink allocation policies of BitMax. We partitioned the set of peers in an experiment to distinguish *slow* peers from *fast peers* and compute the average number of unchoked peers every second[4].

All the results reported in Sections 4.2, 4.3 on the above metrics were obtained by repeating each experiment 10 times. In each execution, peer uplink capacities were drawn randomly from the aforementioned uplink bandwidth distribution. The initial *seed* is assumed to be *well provisioned* with an uplink capacity of 10 Mbps. As we are primarily interested on cooperative scenarios, if not otherwise stated, we consider that leechers stay in the system after they finish downloading.

As a final note before presenting our results we would like to point out that in this work we are primarily interested in extending the *design space* of uplink allocation policies so we cannot, for many practical reasons, examine the *parameter space* as exhaustively as previous works [7, 5] that have focused specifically on the second. Therefore, we do not report results for underprovisioned seeds, or for seeds that depart before all peers are served, or file sizes other than 500 MB. Although we do not show them here, we have sampled points across this parameter space as well, and consistently see substantial improvement through BitMax. Of course, we

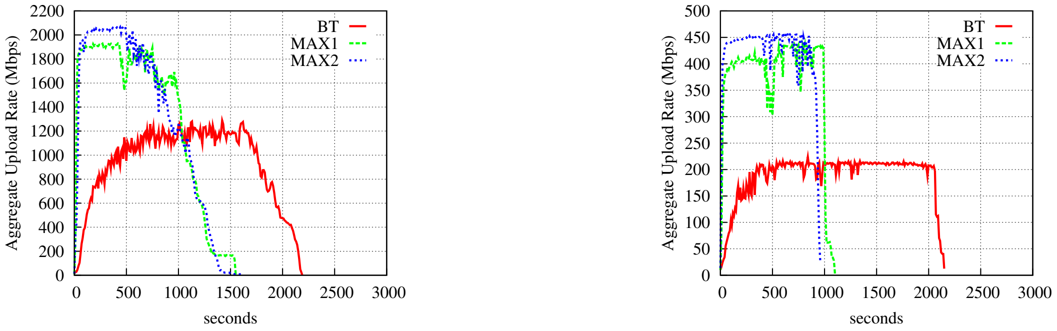
[4] We did not use a choke interval period to compute this metric as peers are not synchronized.



(a) $n = 500$ peers

(b) $n = 100$ peers

Figure 2: ***Case U:*** **CDF of download times for BT and for the two BitMax uplink allocation algorithms ($FS = 500$ MB). In the median case download times are halved with both versions of BitMax as compared to BT; worst-case download times are also in favor of BitMax.**



(a) $n = 500$ peers

(b) $n = 100$ peers

Figure 3: ***Case U:*** **Aggregate service capacity, absolute value ($FS = 500$ MB). A considerable gap during the steady state phase appears, as shown by the peak aggregate upload capacity.**

do not claim any kind of general optimality since BitMax is itself a heuristic.

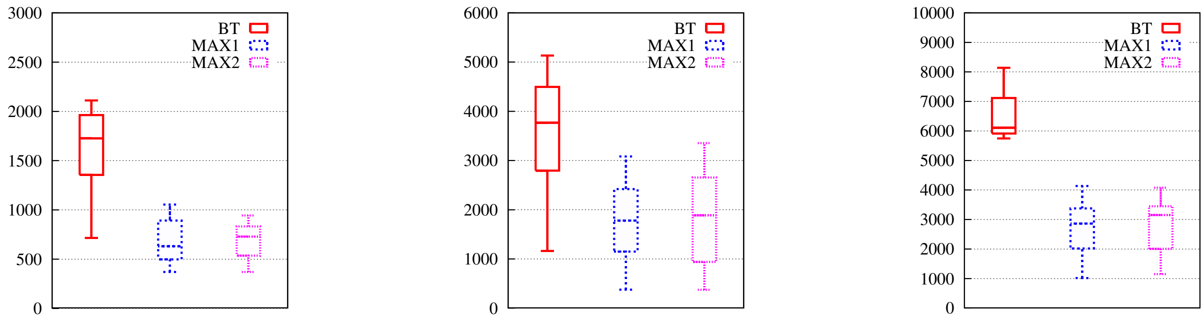
4.2 Bottleneck only at the uplink

We start with the case in which node uplinks represent the only bottlenecks of the system ($caseU$).

Fig. 2(a) and 2(b) illustrate the CDF of the download time across different peers, for $n = 500$ and $n = 100$ peers respectively. Both versions of BitMax yield substantial performance improvements over BT. When $n = 500$ peers, MAX1 and MAX2 have a median download time of 740 sec. and 760 sec. respectively, whereas BT has a median of roughly 1550 sec. The worst case download time for MAX1 and MAX2 is below 1500 sec. while in BT the worst case download time is roughly 2200 sec. The gap widens even more for a smaller system: when $n = 100$ peers, the median and the worst case download time is halved for MAX1 and MAX2 with respect to BT.

The faster downloads are due to BitMax's superior *overall* utilization of uplink capacity. To support this claim, we plot in Fig. 3(a) and 3(b) the time series of service capacity for the different policies, expressed in terms of absolute upload rate. We observe the typical three phases [13] that characterize the lifetime of a swarm-based system: an initial flash crowd phase generally characterized by low utilization, an intermediate phase during which peers reach the maximum attainable rate under the employed uplink allocation policy, and a final phase during which the last few receiving peers approach the completion of the download. With BitMax, the duration of the flash crowd phase is very short. Indeed, the system reaches in a very short time an efficient steady-state regime in which aggregate upload rate is close to the maximum attainable rate (which is the sum of *nominal* uplink bandwidth across all peers) and remains so constantly for a substantial amount of time. In contrast, BT requires much more time to reach a substantially lower peak value of service capacity and the system spends a fairly large amount of time in this sub-optimal state.

These results corroborate recent observations from the literature stating that BT is suboptimal in several occasions. This was first documented by Legout et al.



(a) case U

(b) case UD

(c) case UDN

Figure 4: A comprehensive overview of results: Boxplot of download times when more and more stringent bottlenecks are introduced in experiments (n=100 peers, $FS = 500$ MB, for case UDN there is a 2 Mbps network bottleneck). The two BitMax algorithms maintain their gain in performance over BT across all scenarios, and they are less sensitive to severe network conditions. *Note* the different scale of the boxplots: whereas BitMax remains superior to BT in all the scenario we considered, downlink and network bottlenecks increase the time it takes to distribute the content to all peers.

in [13], but without providing a clear explanation. In a follow up work, Hamra et al. [18] attributed it to clustering phenomena among the first peers that arrive in a torrent. Such "early comers" have the privilege of having the initial seed in their neighborhood. As shown in the aforementioned work, for a period of time the early comers confine piece exchange among themselves, condemning late comers to low utilization. It is only once the initial clique starts breaking that the aggregate system capacity increases by involving more nodes in the active exchange of chunks.

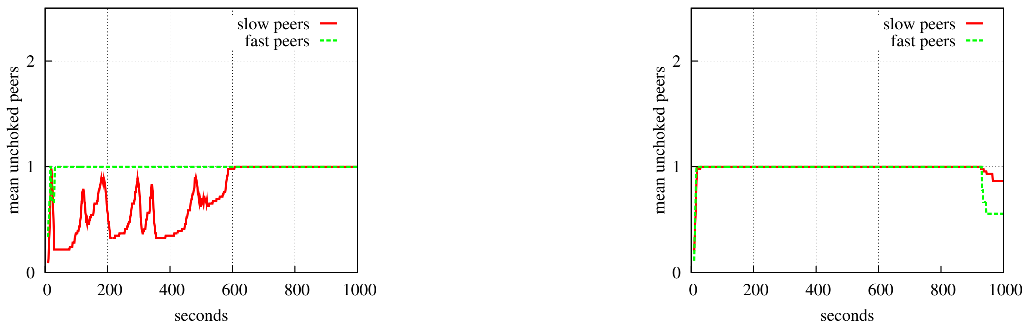
It's quite evident from the previous plots that the service capacity of BT is far from being optimal, especially if one looks at the absolute value. Based on this observation, one could conclude that these results contradict previous works reporting almost optimal uplink utilization for BT [7, 5]. We explain why such a contradiction does not exist. Bharambe et al. [7] report utilization levels that exceed 90%, but they confine their analysis to the steady-state phase of a torrent's lifetime, excluding the initial phase of low utilization. We on the hand, plot the evolution of service capacity throughout the life time of the torrent. When focusing on the steady-state, we see a utilization of around 60% under the uplink capacity distribution that we got from [6], which is different from the synthetic one of Bharambe et al. [7]. We repeated the experiment with this synthetic distribution and verified that indeed BT achieves 90% utilization in this case. The previous, is just a confirmation of our main thesis in this study, which is that the effectiveness of the standard pre-parameterized unchoke algorithm depend on operating parameters and that more elaborate heuristics like BitMax are expected to perform much better in several scenarios.

Similarly, there's no contradiction with Legout et al. [5] that report almost 100% utilization based on real measurements. The key to understanding why, is to notice that their experiment is conducted on PlanetLab and, thus, involves a small number of nodes (around 40). In such small torrents, peers form a fully connected mesh and thus there's no clustering effect to hurt the performance. Almost as important, all peers neighbor the well provisioned initial seed which makes things easier.

4.3 Adding downlink and network bottlenecks

We now study the effects of adding downlink and network bottlenecks in the system. We are interested in observing the impact that the added constraints have on the absolute download times, as well as on the relative performance between BitMax and BT. For the downlink bottleneck case we constrain the downlink capacity to be a multiple of the uplink capacity, as discussed in Sect. 4. We then move to a more elaborate scenario in which network bottlenecks are introduced to simulate the effect of bandwidth throttling at ISP interconnection points. We employ the methodology described in Sect. 4 to decide the bottleneck values (caps) that we introduce at the AS interconnection points of our synthetic topology.

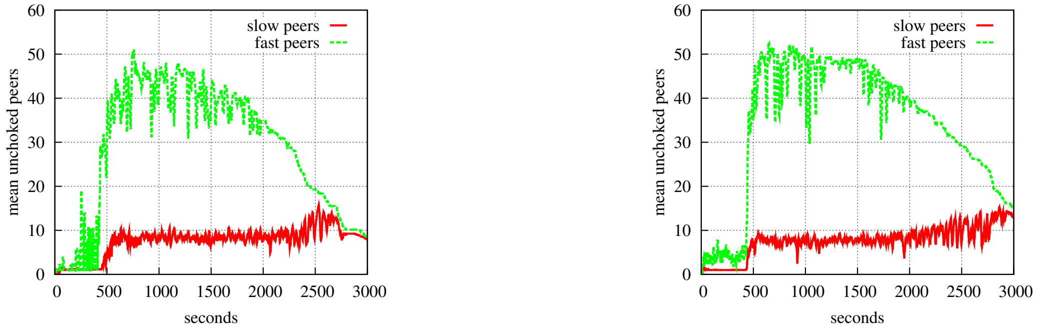
Figure 4 provides a global snapshot of download times under BT and the two versions of BitMax, as we progressively go from the least constrained scenario, to the most constrained one, in the following order: uplink bottlenecks ($caseU$), uplink and downlink bottlenecks ($caseUD$), uplink, downlink and network bottlenecks ($caseUDN$). A glance at Fig. 4, in which we show the case for a 2 Mbps bottleneck, reveals that the addition of extra bottlenecks affects negatively all uplink allocation policies in a similar way: download times raise by a factor of roughly 2.5 when introducing downlink bot-



(a) BitMax 1

(b) BitMax 2

Figure 5: *case U:* Dynamic uplink allocation ($n = 100$, $FS = 500$ MB).



(a) BitMax 1

(b) BitMax 2

Figure 6: *case UD:* Dynamic uplink allocation ($n = 100$, $FS = 500$ MB).

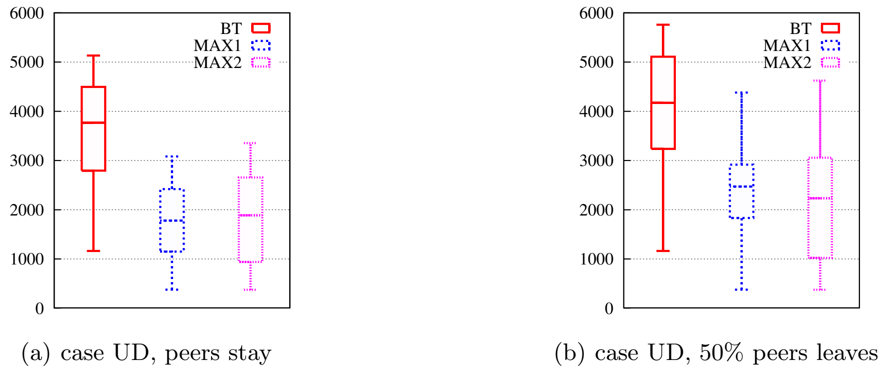
tlenecks and by a factor of roughly 4 when introducing also network bottlenecks. However, BitMax preserves its advantage both in median and worst-case download times, which are halved with respect to BT.

4.4 Looking deeper into BitMax

Dynamic allocation of the uplink: Downlink and network bottlenecks make it more difficult to sustain a high uplink utilization. Their impact, however, is more severe on allocation policies that have to adhere to empirically preset operational parameters. For example, under BT's standard unchoke algorithm there can be underutilization when allocating $1/(k + 1)$ of a fast node's capacity to nodes that are either slow, or sit behind peering points that are being throttled. In such cases, it is required to unchoke more nodes and provide them with smaller bandwidth shares. BitMax achieves exactly that by adapting the allocation dynamically, in response to the current state of the torrent and the physical network. For that purpose, we analyzed the time series of the average number of unchoked peers (or equivalently the number of "items" that are selected in an optimal solution of the knapsack problem of Sect. 3).

Fig. 5 and 6, report the average number of unchoked peers for *case U* and *case UD* for both MAX1 and MAX2, differentiating between slow and fast peers. When there are only uplink bottlenecks, Fig. 5 clearly indicates that a peer using BitMax unchokes less than 4 remote peers, as opposed to what a peer obeying to BT would do; hence, uplink capacity is more "focused". We also observe that for the MAX1 case, slow peers are not always able to unchoke a remote peer because of data un-availability, whereas for the MAX2 case, both slow and fast peers arrive at focusing their uplink capacity to peers that need it most.

When downlink bottlenecks are present, our results indicate that in order to avoid underutilization due to downlink and network bottlenecks, peers using one of the BitMax policies adjust the number of unchoked peers across a wide range of values beyond 4. Fig. 6 shows that once fast peers establish a high aggregate input rate for themselves (up to 500 sec.), their attention turns to slower peers that have started falling behind. MAX1 and MAX2 behave similarly, except during the initial phase in which MAX2 unchokes more peers due to piece (in)availability, as explained in Sec 4.2. When the cross-ISP traffic is subjected to bandwidth throttling, the response of BitMax is similar to the downlink bottleneck case – fast peers unchoke a larger number of peers in order to saturate their uplink bandwidth. How-



(a) case UD, peers stay

(b) case UD, 50% peers leaves

Figure 7: Impact of peer departures: Boxplot of download times for *case UD* when all peers stay in the system and for *case UD* when 50% of peers leave the system once they finish downloading (n=100 peers, $FS = 500$ MB). The two BitMax algorithms maintain their gain in performance over BT across all scenarios, and they are less sensitive to peer departures, although we notice an increased variance in the distribution times with peer departures.

ever, in this case, the maximum number of unchokes is smaller because peers at other ISPs are not unchoked unless they can sustain a minimal rate. This is required for avoiding pathological cases in which a chunk may delay too much due to a very small allocated rate.

Peer departures: We now examine the performance of BitMax and BT when peers leave the system as soon as they finish downloading the content. Fig. 7(a) and 7(b) summarizes the results for download times in *caseUD* with 50% of the peers leaving the system as soon as they finish to download. As expected, we notice a global increase of the median and worst case download time for both BT and BitMax. Both versions of BitMax still achieve better download times than BT. The higher variance of download times owes to the added randomness introduced during the selection of the 50% of peers that get to leave the torrent after becoming seeds.

5. DISCUSSION

In this section we first comment on implementation requirements for integrating BitMax allocation in existing BT clients and then discuss the possibility of using BitMax in non-cooperative environments (e.g., typical file-sharing).

Implementation requirements: BitMax is a new uplink allocation mechanism not a different protocol, so integrating it to existing clients requires rather limited amount of modification. The required information for executing BitMax can be obtained as follows. For unchoked nodes, $u(v, v_i)$ can be updated by monitoring the actual data transfers: if the ongoing transfer is slower than expected then the estimate is adjusted downwards; occasionally additional epsilons of bandwidth can be used for detecting increases in the maximum attainable rate. For choked nodes, $u(v, v_i)$ can be estimated through optimistic unchokes: a sensible strategy for the "probing" node v can be to start with a small initial rate and increase it progressively on subsequent optimistic unchokes (e.g., double it) until it reaches the maximum rate that the probed node v_i can support. In terms of remote information, each node v needs to know $u(v_i)$ and $\lambda(v_i)$ for each neighbor $v_i \in V(v)$. This information can be piggybacked on HAVE messages. Of course the reporting clients need to be trusted, which is not a problem in the cooperative case. Next we discuss the possibility of using BitMax in settings where users might be selfish.

BitMax and selish peers: To use BitMax in this case it would require to have a "closed client". This is the approach taken by *Skype* and other P2P systems that users download and use as they are. The promise of this approach is that if the system is well designed, then users will adopt it because of its superior performance. In the case of BitMax a closed client would first of all guarantee that all nodes follow faithfully the protocol and report truthfully their information (nominal upload rate and current reception rate). If nodes upload as dictated by BitMax, then as we showed in Sect. 4, all nodes will enjoy high download rates. Since the client is closed, a free-riding user that wants to preserve its uplink, would have to use a third program (a filter) to drop the uploading chunks from its client before they get transmitted on the network. Such attempts are rather too elaborate for the average user and can also be disclosed by having the client issue digitally signed reports with the amount of data it has sent and the exact recipients. A node receiving such a report and realizing that it didn't get the actual data can identify the filtering and disclose the free-rider.

6. RELATED WORK

In Cohen's original paper [1], it was suggested that k should be set equal to 4, so as to "allow TCP's built-in congestion control to reliably saturate the upload capacity". Such empirical setting works indeed well when there is high chunk availability (i.e., after the initial phase) and when there are no losses in utilization of the uplink due to congested receiving peers (which pretty much implies that the peers should be highly homogeneous, the uplink and downlink rates be highly asymmetric, and the network be imposing no bottlenecks). As shown in Sect. 4, by adapting the allocation on a per unchoke basis we can both improve the utilization during the startup phase, as well as handle arbitrary settings in terms of peer rates, amount of uplink/downlink asymmetry, and even network bottlenecks.

Following the initial BitTorrent paper, some works such as Bharambe et al. [7], and very recently again Mol et al. [10], have asked the question of how much should the uplink be partitioned, but provided no answer. We have addressed such question through our dynamic BitMax allocation based on the solution of a knapsack problem. However, instead of being trapped into a discussion of selfishness and tit-for-tat, we chose to explore the full potential of uplink utilization, which can only be realized with utility functions that target the improvement of the social, instead of individual welfare. We took care, nevertheless, to include in our objective functions, natural load balancing heuristics that make sure that no node is "sacrificed" in the process of optimizing the social welfare. Our BitMax policy is obviously applicable in networks of peers working for a common goal. In networks including potentially selfish peers, we believe there are ways to realize BitMax through "closed" clients, but we have left the development of a prototype system to future work.

Diametrically opposite to our approach is the recent work of Piatek et al. [6] on the BitTyrant client. They also show how to allocate the uplink dynamically on a per unchoke basis, but with a different objective. Their intention is to maximize the number of reciprocating peers, so they strive to saturate the uplink by partitioning it in as many pieces as possible, while taking care to make each piece just large enough to guarantee reciprocation from the receiving peer. As shown by the authors, this becomes detrimental to the social utility as the percentage of BitTyrant clients increases. Contrary to that, our allocation of uplink works towards the improvement of the social welfare and is trying to focus the allocation of bandwidth. For these properties, we could think of BitMax as being the opposite of BitTyrant.

Several simulation[7] and measurement-based works [4, 20, 13, 5] have shown that BitTorrent is quite effective during the intermediate phase in typical ADSL scenarios where constraints are mostly on the uplink of nodes. Our results are aligned with this previous works and complement them by looking more closely at the low utilization problem in the initial phase, and at addressing additional bottlenecks, especially at the core of the network, caused by throttling of P2P traffic at ISP interconnection points. Works that substitute BitTorrent's piece selection policy with network coding [21] are also based on setting the number of parallel uploads empirically and thus can benefit from our work.

Analytic works have shown that BitTorrent works efficiently once it reaches steady-state in large networks. For example Qiu and Srikant [8] have shown that its "efficiency" becomes $1 - \left(\frac{\log n}{n}\right)^{w}$ in such case (w denotes the neighborhood size). This result is derived assuming peers with homogeneous upload capacity and is oblivious to the way that this capacity is split. It also refers to torrents that have reached high chunk availability, thus it ignores the initial phase in a torrent's life in which utilization is low. Massoulie and Vojnovic [9] have presented a more refined analysis of steady-state dynamics in which they used a contact process to model more faithfully the exchanges of "coupons" (chunks) among interacting peers (in [8] the specifics of the coupon exchange process were not modeled). Still they need to make several simplifying assumptions as before, e.g., they consider that $k = 1$, and thus their analysis is not able to discriminate among the different uplink allocation policies studied here.

7. CONCLUSIONS

In this work we set off to examine whether it is possible to improve the uplink utilization of BitTorrent, and thus help in reducing further the download times for the benefit of the users. In trying to meet such a challenge, we had to question the efficiency of certain design choices in the currently employed choke/unchoke algorithm, which up to now, had not be justified sufficiently. We did so through the development of our so called *BitMax* allocation policy. Contrary to the existing empirically parameterized algorithm, BitMax allocates the uplink dynamically in response to the current operating conditions. Among other things, it tries to focus instead of over-curve the uplink. It also avoids underutilizing it by unchoking receivers that cannot fully absorb the allocated rate (due to downlink or network bottlenecks). To reduce the download times below what an implicit coordination through tit-for-tat can achieve, we built BitMax around objective functions that consistently promote the social welfare of the torrent without, however, sacrificing the download quality of any individual peer in the process. BitMax is fundamentally as simple as the standard unchoke algorithm, and has certain concrete advantages over it: it fixes the low utilization problem during the initial phase, it is better or at least as good during the steady state, and it reacts better to network bottlenecks. With these properties we have demonstrated that it is in position to produce significant performance gains in several settings.

8. REFERENCES

[1] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. of First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, Jun 2003.

[2] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in BitTorrent via biased neighbor selection," in *Proc. of IEEE ICDCS '06*, Lisbon, Portugal, 2006.

[3] Slashdot, "Comcast Hinders BitTorrent Traffic," Aug 2007.

[4] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems." in *Proc. of ACM IMC'05*, Berkeley, CA, 2005.

[5] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *Proc. of ACM SIGMETRICS '07*, 2007.

[6] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?" in *Proc. of NSDI'07*, Cambridge, MA, 2007.

[7] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *Proc. of IEEE INFOCOM '06*, Barcelona, Spain, 2006.

[8] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. of ACM SIGCOMM '04*, 2004, pp. 367–378.

[9] L. Massoulie and M. Vojnovic, "Coupon replication systems," in *Proc. of ACM SIGMETRICS '05*, Banff, Alberta, Canada, 2005, pp. 2–13.

[10] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips, "Give-to-get: An algorithm for P2P video-on-demand," in *Proc. of SPIE/ACM MMCN '08*, San Jose, California, Jan 2008.

[11] N. Laoutaris, P. Rodriguez, and L. Massoulie, "ECHOS: edge capacity hosting overlays of nano data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, pp. 51–54, 2008.

[12] C. H. Papadimitriou and K. Steiglitz, "Combinatorial optimization: Algorithms and complexity." New York: Dover Publications, 1998.

[13] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proc. of ACM IMC '06*, Rio de Janeriro, Brazil, 2006.

[14] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proc. of ACM EC '04*, New York, NY, USA, 2004.

[15] W. Feller, *An Introduction to Probability Theory and Its Applications*. New York: Wiley, 1968.

[16] E. Altman, D. Barman, B. Tuffin, and M. Vojnovic, "Parallel tcp sockets: Simple model, throughput and validation," in *Proc. of IEEE INFOCOM '06*, Barcelona, Spain, 2006.

[17] W. Yang and N. B. Abu-Ghazaleh, "GPS: A general Peer-to-Peer simulator and its use for modeling BitTorrent," in *in Proc. of MASCOTS'05*, Atlanta, GA, 2005, pp. 425–434.

[18] A. A. Hamra, A. Legout, and C. Barakat, "Understanding the properties of the Bittorrent Overlay," Inria, Technical Report 00162088, version 1, July 2007.

[19] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *Proc. of MASCOTS '01*, Cincinnati, OH, Aug 2001, pp. 346–354.

[20] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "The BitTorrent P2P file-sharing system: Measurements and analysis," in *Proc. of IPTPS'05*, Ithaca, NY, 2005.

[21] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. of IEEE INFOCOM '05*, Miami, FL, USA, 2005.