2010-10-15

On the Impact of Seed Scheduling in Peer-to-Peer Networks

On the Impact of Seed Scheduling in Peer-to-Peer Networks

Boston University, Computer Science Technical Report BUCS-TR-2010-034

Flavio Esposito
flavio@cs.bu.edu
Computer Science Department
Boston University
Boston, MA

Ibrahim Matta
matta@cs.bu.edu
Computer Science Department
Boston University
Boston, MA

Debajyoti Bera
dbera@iiitd.ac.in
IIIT-Delhi
New Delhi
India

Pietro Michiardi
pietro.michiardi@eurecom.fr
Eurecom Institute
Sophia Antipolis
France

***Abstract*—In a content distribution (file sharing) scenario, the initial phase is delicate due to the lack of global knowledge and the dynamics of the overlay. An unwise piece dissemination in this phase can cause delays in reaching steady state, thus increasing file download times. After showing that finding the scheduling strategy for optimal dissemination is computationally hard, even when the offline knowledge of the overlay is given, we devise a new class of scheduling algorithms at the seed (source peer with full content), based on a proportional fair approach, and we implement them on a real file sharing client. In addition to simulation results, we validated on our own file sharing client (BUTorrent) that our solution improves up to 25% the average downloading time of a standard file sharing protocol. Moreover, we give theoretical upper bounds on the improvements that our scheduling strategies may achieve.**

***Index Terms*—Peer-to-peer, seed scheduling, distributed protocols, BitTorrent.**

I. Introduction

In the last decade, content distribution has switched from the traditional client-server model to the peer-to-peer *swarming* model. Swarming, $i.e.$ parallel download of a file from multiple self-organizing peers with concurrent upload to other requesting peers, is one of the most efficient methods for multicasting bulk data. Such efficiency is in large part driven by cooperation in the ad-hoc network forming the swarm. This paradigm has recently been widely studied as it enables lot of potential applications (see $e.g.$, [1]–[4] and references therein), with particular focus on making content distribution protocols efficient and robust.

In particular, measurements [4], simulation [5], and analytical studies [6] on BitTorrent-like protocols have shown that, even though peers cooperate, leveraging each other's upload capacity, this operation is not done optimally in every possible scenario. Prior research in this area has therefore focused on both the analysis and the improvement of clients (or downloaders) in their strategies for selecting which neighbor (peer) to download from — including dealing with free riders using rational tit-for-tat exchanges [3], or reducing inter-ISP traffic using geographic or topological locality [7] — and which part of the content (piece) to download next [8].

Other studies [9], [10] have shown that there may be source bottlenecks due to a uneven piece distribution during initial phases and due to churns — transient phases characterized by a burst of simultaneous requests for a content. In fact, during churn downloaders' strategies are less effective since the connected (neighboring) peers have either no interesting content, or no content at all. As a consequence, the time to download increases.

We claim that the source (also called seed) can help through scheduling of pieces, in a better way than simply responding immediately in a FCFS manner, upon downloaders requests by providing a notion of fairness in distributing the content pieces.

We propose to provide piece-level fairness via proportional fair scheduling at the source of the content to ensure that different pieces are uniformly distributed over the peer-to-peer network. Specifically, we show when and how this policy enforced by the seed may result into more effective exchange of pieces and therefore into shorter average time to download.

The main contributions and the contents of the paper are summarized as follows: in Section II we give an overview of the BitTorrent protocol, defining some notions that we use in later sections. In Section III we describe why a content distribution protocol ala BitTorrent does not work well in dynamic scenarios, then go on to describe the problem of source scheduling of pieces to allow downloaders to finish as fast as possible which we prove to be NP-Hard.

Consequently, in Section IV, we describe our polynomial time seed scheduling algorithm for content distribution protocols and evaluate its performance in later sections.

In particular, in Section V we give an upper bound on the performance improvement that our seed scheduling algorithm may achieve, after showing why scheduling at the seed is important to reduce the time to download. Moreover, we adapt the analytical fluid model in [4] so it is valid even during the transient (initial and churn) phases, capturing the effect of seed scheduling, and we show that a smarter seed scheduling increases the chances that leechers will more often find their missing pieces among their neighbouring leechers.

This work has been partially supported by a number of National Science Foundation grants, including CISE/CCF Award #0820138, CISE/CSR Award #0720604, CISE/CNS Award #0524477, CNS/ITR Award #0205294, and CISE/EIA RI Award #0202067.

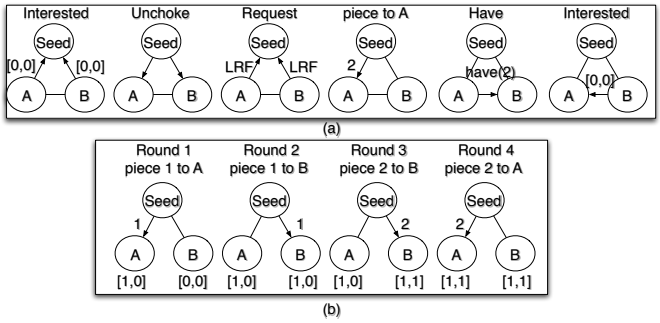


Fig. 1. (a) BitTorrent protocol: from left to right, the sequence of messages. (b) Global information leads to faster content distribution. Peers unaware of each other cannot use swarming.

Section VI experimentally validates our analysis of effective swarming using both a known event-driven peer-to-peer simulator [11], and an implementation of our solution on a real file sharing client (that we call BUtorrent [12]), confirming also on Planetlab [13] that in dynamic overlays, we improve up to 25% the average downloading time. Finally, in Section VII we discuss related work.

II. BitTorrent Overview

BitTorrent is a file sharing protocol for content dissemination. The content is divided into 256 KB pieces so that peers not having the whole content can speed up their download by exchanging pieces among themselves. A peer-to-peer system applying a swarming protocol labels the peers interested in downloading pieces *leechers*, and peers that act only as servers (i.e., only upload content) *seeders* or *seeds*. In this work, we consider a *torrent T* to be a set of at least one seed, at least two leechers, a tracker which is a special entity that helps bootstrapping the peer-to-peer network, and a file *F*, split in p pieces, to be distributed. Moreover, we use the following definitions:

Definition 1: (Neighborhood) The neighborhood or peerset of a peer v, denoted as $\Gamma(v)$, is defined as the set of peers directly connected to v. We denote its size $|\Gamma(v)|$ as k_v.

The BitTorrent protocol works as follows (see Figure 1(a) for an illustration): if a peer v_i, leecher or seed, has pieces that another connected leecher v_j does not have, an "interested" message is sent from v_j to v_i. Together with the interested message, a binary vector called *bitfield*, is sent. Every peer has a bitfield, of length equal to the number of pieces of F, and a bit is set to one if the peer has the corresponding piece. Through the bitfield dissemination, each peer v has information of the pieces present in $\Gamma(v)$ (only). Amongst all the interested leechers, only $Mleqk_v$ ($M = 4$ in the first version of the protocol) peers are given an opportunity to download data, using the *choke* algorithm, which is applied every re-choking interval T_{rc},[1]. For seeds, it is simply a round robin schedule among the peers in $\Gamma(s)$.

[1] T_{rc} is (mysteriously !) set to ten seconds in most file sharing implementations ala BitTorrent.

For leechers, choking is based instead on two mechanisms: (i) tit-for-tat and (ii) optimistic unchoke. The tit-for-tat peer selection algorithm captures the strategy of preferring to send pieces to leechers which had served them data in the past: upload bandwidth is exchanged for download bandwidth, encouraging fair trading. In the optimistic unchoke, one leecher (up to M) is chosen at random from $\Gamma(v)$. The objective is mainly to explore potentially faster peers; furthermore, this peer selection strategy is more effective than the tit-for-tat in the initial rounds because few leechers have interesting pieces to offer. In Figure 1(a) only two leechers (A and B) are in $\Gamma(s)$, so both are unchoked.

Unchoked leechers select one piece of the content they want to download using the Local Rarest First (LRF) algorithm. Thanks to the disseminated bitfield, a leecher v counts how many copies of each piece are present in $\Gamma(v)$, and then requests the rarest using LRF; any ties are broken at random. As soon as a peer gets a request it replies with the piece. In Figure 1(a) the piece with ID 2 goes to the leecher A. Upon reception of a piece, leechers inform all their neighbors with a "have" message. Now, if leecher B does not have piece with ID 2 yet, B sends an interested message to A.

Definition 2: (Initial Phase) Given a torrent *T*, we define the initial phase of *T* to be the time interval between the first scheduling decision made by the first seed, and the time the last piece has been completely uploaded to some leecher in the seed's peerset.

After the initial phase is complete, we say that the protocol enters the *steady state*.

For completeness of our protocol overview, we also introduce the last phase of the protocol. When a leecher v has only one piece left, BitTorrent changes its normal behavior: v requests for its last piece to all its k_v neighbors; as soon as some peer w replies with the piece, v sends to $\Gamma(v) \setminus w$ a "cancel" message to avoid waste of upload capacity. Formally we have:

Definition 3: (End-Game Mode) Given a torrent *T*, we say that a leecher v of *T* is in the end-game mode if it has completed the download of all but its last piece.

As in [5], we ignore the end-game mode as is it inconsequential to our study, having no effect on the steady state performance.

III. Protocol Weaknesses and Problem Definition

We consider the problem of a seed scheduling the piece whose injection helps reducing the download time for all leechers, as well as the time it takes for the entire content to be fully injected by the seed. We show how, especially in dynamic overlays, increasing the efficiency of swarming translates into reducing the requests to the seeds, that translates into reducing the first content uploading time. As a consequence, the average leechers downloading time is also reduced. To do so, we need the following definitions:

Definition 4: (Effectiveness) Given a peer-to-peer system, we define effectiveness of a file sharing η as the probability

that a leecher v has at least an interesting piece for its neighborhood $\Gamma(v)$.
We discuss the effectiveness in detail in Section V-B.

Definition 5: (Burstiness) Given a dynamic peer-to-peer system —peers join and leave the overlay — we define burstiness as the ratio of the peak rate of leechers' arrival to the average arrival rate during a given observation period.

Observe that the *peak rate* is defined as the ratio of the size of a churn (number of newly arriving leechers) to the length of the interval over which the churn occurs, and that the *average arrival rate* is defined as the ratio of the total number of leechers to the total considered time.

Definition 6: (Seed Utilization) We define seed utilization as the ratio of the number of uploads from a seed to the average number of uploads from all leechers in the system.

Definition 7: (Clustering Coefficient) Given a graph G=(V,E) with V vertices and E edges, and denoting with $E_i \subseteq E$ the subset of edges that includes vertex i, we define Clustering Coefficient for G [14]:

$$CC = \frac{1}{|V|} \sum_{i=1}^{|V|} \frac{E_i}{\begin{pmatrix} k_{v_i} + 1 \\ 2 \end{pmatrix}}. \quad (1)$$

Although BitTorrent has been widely studied by the research community and recognized as a robust protocol, it is far from being an ideal protocol in every scenario. In the presence of burstiness for example, $i.e.$, churn phases, peersets can be too small to significantly benefit from swarming [15]. Moreover, for overlays with low clustering coefficient, *seed utilization*, can be surprisingly high. In our work we focus on seeds rather than leechers.

To the best of our knowledge, only in [5] seeds have been taken into consideration, assuming significant changes to the whole BitTorrent protocol. In our approach, we only modify the scheduling at the seed, leaving the rest of the BitTorrent protocol intact.

Currently, the piece selection algorithm in BitTorrent-like protocols, is done uniquely by the leechers throughout the LRF algorithm; the seed simply replies to the leechers requests in a first come first serve (FCFS) fashion. The scheduling is crucial in the transient phases since the injection of too many duplicates diminishes the swarming efficiency and wastes seed upload bandwidth. During any transient phase, almost every piece has the same LRF rarity ranking, as almost no leecher has any piece, and since in LRF ties are broken at random, a leecher may end up asking the same piece that one current neighbor already asked in the current or previous rounds.

A. Global vs Local Information

Consider Figure 1 (b): in the first round the seed uploads piece 1 to leecher A which updates its bitfield vector. Since A is not connected to B, no "have" message is sent. In the second round, the seed uploads the **same** piece 1 to B. The same happens in the third and fourth round for piece 2. Peers unaware of each other cannot use swarming, which slows down their downloading time as they ask the seed for the same pieces. To use swarming, peers have to have neighbors with interesting pieces. Seeds are interesting by definition as they have the whole content; leechers may not be because, the LRF algorithm does not yield an equal distribution of pieces (and replicas), so it is less likely for a leecher to find its missing pieces at other neighboring leechers since the view leechers have is limited. Henceforth, we refer to the equal distribution of pieces as "piece fairness". The lack of piece fairness is especially more pronounced when the clustering coefficient of the overlay is low.

Without fairness, requests to the seed for the same pieces can occur more frequently before the whole content is injected. This can lead to torrent death if the seed leaves the system prematurely and leechers are left with incomplete content. In some other cases, uploading twice the same piece can be even costly. For example, if the seed is run using the *Amazon S3* [16] service, higher seed utilization results in higher monetary costs for using Amazon resources (bandwidth, CPU, etc.)

B. Dynamic Overlays

The problem we present here is related to the the dynamic nature of the overlays: lack of offline knowledge of which peer will be present at which instant of time, makes the piece distribution problem challenging. In fact, in case a seed or a leecher schedules a piece to a leecher j that later goes temporarily or definitively offline, the effectiveness of the overlay segment containing j is reduced inevitably and so performance decreases.

C. The Initial Phase Problem

A third problem occurs during the initial phase of a torrent. We generalize the initial phase concept to a *transient phase*, namely, any phase in which the system does not fully utilize its swarming properties due to the lack of pieces to trade. Accordingly, in the rest of the paper we interchangeably use transient and initial phase to mean any phase displaying a poor interest in unchoking leechers.

Recalling how the tit-for-tat mechanism works (Section II), when a new group of leechers joins the overlay (churn phase), even though swarming techniques are used, leechers have not yet downloaded many pieces, so every leecher will most probably (and not surely due to the rare optimistic unchoke events) ask the seed for its missing pieces, making the seed a bottleneck. Therefore, during any transient phase, the seed is one of the few interesting peers. Even though this congestion at the source (seed) is inevitable, and may not last for too long, this phase can be prolonged if the dissemination is inefficient.

We are not the first to observe that source bottlenecks occur due to inefficient piece dissemination during flash crowds and churn phases [17]. We consider any churn phase to be another initial phase, and so during *all* the initial phases of a torrent, a wise seed scheduling is crucial.

D. Example of Inefficiency of the Initial Phase

We illustrate the inefficiency of the initial phase through an example.

Definition 8: (Collision) We define a collision event at round i, C_i to be the event that two leechers ask the seed for the same piece at round i.

Note that if two connected leechers generate a collision at round i, and both are served, we have a suboptimal use of the seed upload capacity as one leecher could download the colliding piece using swarming.

Definition 9: (Earliest Content Uploading Time) We define earliest content uploading time, the time it takes for the seed to upload at least once every piece of the file.

A collision event slows down performance with respect to both the earliest content uploading time, and downloading time, since the seed's upload capacity is used to upload more than once the same piece. In particular, for every occurring collision, an extra round may be needed to inject the whole content.

Example of optimal seed scheduling: Let us consider one seed, six pieces, and three leechers A,B and C. Assume that, running LRF, the leechers end up asking the seed for pieces in the following order: $A = [1, 2, 3, 4, 5, 6]$, $B = [4, 5, 3, 1, 6, 2]$ and $C = [3, 6, 2, 5, 1, 4]$. Let us also assume the seed uploads three requests in each round. After the first seed scheduling round, pieces 1,4 and 3 are uploaded, and after the second round, all the content is injected. This is the optimal case: to inject six pieces the seed needs two rounds.

Collision example: If instead the LRF for B ends up with the permutation $B = [4, 2, 1, 3, 6, 5]$, we notice that in the second scheduling round, we have a collision between leecher A and B, both asking the seed for piece 2, and so two scheduling rounds are not enough anymore; to inject the whole content we need to wait three rounds.

Nowadays, BitTorrent-like protocols do not have policies for the seed to schedule pieces, since the piece selection is done by LRF. In the next sections we present an algorithm that attempts to overcome the lack of leechers' awareness of each other, improving piece fairness, and we show that scheduling is crucial for boosting swarming effects.

E. Complexity of Piece Dissemination

To motivate our attempt to devise a better scheduling at the seed, we show that finding the optimal piece dissemination is computationally hard, even if the offline knowledge is given about the overlay of the peers in the peerset. As described in Section III, one focus of the seed scheduling is to increase swarming efficiency — increasing the chances that neighbouring leechers have interesting pieces to trade — and therefore to reduce downloading time. As an additional benefit, such efficiency reduces uploads requests from the seed, whose upload capacity may turn out to be the bottleneck in the initial phases as shown in [9], [10].

We consider a simplified offline variant of the problem which distributes all the pieces among the leechers such that they can exchange pieces in the best possible way. Suppose that there is only one seed and a static set of leechers, whose topology is known to the seed. To compute the complexity of the problem, instead of the peers asking for pieces, we let, as in [5], the seed decide which piece to give to which peer. The seed is allowed to upload pieces to all of its peers in any round, but we restrict the seeds activity to only the first round – the seed neither uploads any piece nor plays any other role on all but the first round. Therefore, it becomes necessary to make one additional assumption that the number of pieces is upper bounded by the number of leechers. The objective of the seed is to minimize the time taken by all peers to download all the pieces.

Since the seed knows the topology, the number of rounds for this case is surely a lower bound on the number of rounds it would take if the leechers apply any piece selection algorithm ($e.g.$ LRF) to decide which pieces to request. We show that even in this simplified form, it is NP-hard for the seed to decide how to schedule its upload for the first and only round. Only leechers which are reachable from one another can effectively participate in swarming, hence, we assume that all leechers form a connected graph; our hardness result works independently for each connected component of the overlay.

We prove the NP-hardness by considering the following decision version of a graph theoretic model of the above problem.

Definition 10: (Complete Piece Dissemination) Given an undirected connected graph $G(V, E)$, an integer $k \leq |V|$ and an integer $p \leq |V|$, we want to decide if there is a way to assign one piece $p_v \in \{1, \ldots, p\}$ to every vertex v such that the following holds:

1) Before the beginning of round 1, vertex v is given piece p_v
2) In each subsequent round, v collects all the pieces in its neighborhood
3) By the end of round k, v has collected all the pieces $1, \ldots, p$.

We now show that the Complete Piece Dissemination problem is NP-complete by reducing from another graph problem, the *Generalized Domatic Partition (GDM)*, which is a generalization of a known NP-complete problem - *Domatic Partition* [18]. In order to do that, we generalize the Definition 1 of neighborhood to k hops.

Definition 11: (k-neighborhood) Let $G(V, E)$ denote an undirected graph. Given a subset $S \subseteq V$ of vertices, we define the neighborhood of S as $\Gamma(S) = \bigcup_{v \in S} \Gamma(v)$. The k-neighborhood of a vertex v is inductively defined as the set of all vertices that are reachable from v by a path of length k or less. Formally:

$$\Gamma^k(v) = \begin{cases} \{v\} & \text{if k = 0} \\ \Gamma^{k-1}(v) \cup \Gamma(\Gamma^{k-1}(v)) & \text{if } k > 0 \end{cases} \quad (2)$$

Furthermore, the k-neighborhood of a set of vertices is defined as: $\Gamma^k(S) = \bigcup_{v \in S} \Gamma^k(v)$.

Definition 12: (Dominating Set) Given an undirected connected graph G, a dominating set S of G is a set of vertices

such that each vertex either belongs to S or it is a neighbor of some vertex in S *i.e.*, every vertex $v \in \Gamma^1(S)$.

Definition 13: (Domatic Partition) Given an undirected connected graph G, a domatic partition of G is a partition of the vertices such that each partition is a dominating set. The size of a domatic partition is the number of partitions. The *domatic number* of a graph is the maximim size of any domatic partition.

For any connected graph, its domatic number is at least 2 and at most $\delta + 1$ where δ is the minimum degree of any vertex in the graph. It is well known that finding the domatic number of any graph is NP-hard [19]; in other words, the following problem is NP-complete: Given a graph G and an integer $p \leq |V|$, is there a domatic partition of size p?

We generalize the above definitions to include the concept of k-neighborhood.

Definition 14: (Generalized Dominating Set) Given an undirected connected graph $G(V, E)$, a k-th order dominating set is a set of vertices S such that for every vertex $v \in V$, $v \in \Gamma^k(S)$.

Definition 15: (Generalized Domatic Partition) Given an undirected connected graph $G(V, E)$, an integer $k \leq |V|$, and an integer $p \leq |V|$, does there exist a partition of V into $V_1, \ldots, V_p$ such that each V_i is a k-th order dominating set?

Lemma 1: The Generalized Domatic Partition problem is NP-complete.

Proof: Observe that the original problem of dominating set (and domatic partition) is a special case of generalized dominating set (and generalized domatic partition) with $k = 1$. It is easy to see that generalized domatic partition is in NP, since given any partition, it can be verified in polynomial time if each of the partitions indeed forms a k-th order dominating set. Therefore, the generalized domatic partition problem is also NP-complete. ∎

Theorem 2: The Complete Piece Dissemination problem is NP complete.

Proof: The complete piece dissemination and the generalized domatic partition problems are equivalent: the vertices initially given piece i in the complete piece dissemination problem form the partition V_i in a generalized domatic partition and vice-versa. Since the generalized domatic partition problem is NP-complete, therefore so is the complete piece dissemination problem. ∎

We end this section by discussing our assumptions, pointing out how the Complete Piece Dissemination problem, whose complexity we deduced to be NP-complete, might differ from real systems, with respect to peer-to-peer protocols ala BitTorrent. In general, the scenario to analyze is far more complex as overlays may have multiple connected (but uncoordinated) seeds, thereby allowing more pieces in the network in a shorter time. Moreover, these protocols have a limit on the number of TCP connections that any leecher may keep open (in the first version of BitTorrent for example, the maximum degree was set to eighty), and a limit on the number of simultaneous uploads by a seed or a leecher, thereby limiting the overlay size and therefore the rate of piece exchange. A significant difference is the dynamic nature of a real overlay, where leechers join and leave, seeds arrive and depart from the system, resulting in a constantly changing overlay structure. Theoretical analysis of such dynamic systems, as well as design of protocols that would also orchestrate cooperation among the seeds (besides the usual cooperation among leechers) are beyond the aims of this paper and are left as interesting research directions.

IV. Seed Scheduling

In this section we provide a detailed explanation of how we propose to modify any swarming protocol ala BitTorrent. Our solution, based on the Proportional Fair Scheduling (PFS) algorithm [20], [21], exploits and improves the method conceived by Bharambe $et\ al.$ [5], where memory about pieces scheduled in the past was first introduced. Furthermore, we formally present the scheduling algorithm that we applied at the seed.

A. Detailed Idea of our PFS Scheduling at the Seed

Smartseed [5] inserts an intelligence into the seed by actively injecting the pieces least uploaded in the past, changing the nature of the BitTorrent protocol from pull (on demand) to push (proactive). Our seed strategy instead, consists of unchoking every possible leecher, so that all the requests are taken into account, and uploading the M (*four*) pieces that are both requested the most in the current round and uploaded the least in the past rounds, without changing the way BitTorrent-like protocols work. The piece requested the most in the current scheduling round represents the instantaneous need, namely, the present, while the least uploaded piece represents the past scheduling decisions. We can view Smartseed as considering only the past. The present need gives the seed a rough view of which pieces leechers currently need, perhaps because neighboring leechers do not have these pieces. While past uploads gives the seed the ability to inject the piece that is least replicated from its vantage point. By equalizing the number of piece replicas, the probability that leechers have pieces that are of interest to their neighboring leechers increases.

Formally, if r_i is the number of the requests for piece i at the current round, and τ_i is the throughput vector for piece i, namely, the number of times piece i has been uploaded in all the previous rounds, PFS chooses the piece i^* that maximizes the ratio $r_i[t]/\tau_i[t-1]$, namely:

$$i^*[t] = \arg\max_i \left\{ \frac{r_i[t]}{\tau_i[t-1] + \epsilon} \right\} \tag{3}$$

where ϵ is just a small positive constant introduced to avoid division by zero. The r_i is decreased every time a piece message is sent and τ_i is increased every time a piece has been completely uploaded. The BitTorrent protocol splits the 256 KB piece further into sub-pieces. We ignored this further fragmentation in our simulations but not in our system implementation (BUtorrent [12]).

B. History has to be forgotten

When peersets' peer (neighborhoods) are dynamic, keeping track of the pieces uploaded since the beginning of the torrent is a bad idea, since some leechers may have left, others may have arrived. The intuition is that the weights that τ_i brings into the scheduling decision has to decrease over time. For this reason, every time the seed uploads a piece, we update τ_i using exponential weighted moving average (*EWMA*), namely:

$$\tau_i[t+1] = \beta \cdot I_i[t] + (1-\beta) \cdot \tau_i[t] \quad (4)$$

where $I_i[t]$ is an indicator function such that:

$$I_i[t] = \begin{cases} 1 & \text{if piece } i \text{ was uploaded at round (time) t} \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

From power series analysis [22], we can express the smoothing factor β in terms of N time periods (scheduling decisions) as: $\beta = \frac{2}{N+1}$. This means that an upload decision is forgotten after N rounds.

What is the best choice for N then? Let us assume that a considered seed S has, at most Γ connections (in BitTorrent, $\Gamma = 80$). Let us also assume a worse-case scenario, where the seed's neighborhood is made of leechers that do not serve each other. When the peerset is static (peers neither arrive nor depart), then the seed can schedule the same piece i at most Γ times, one for each leecher, since peers do not ask again for a piece they already have.

If the seed happens again to receive a request for piece i, it means the request is coming from a new leecher which joined the overlay after the first time i was uploaded by the seed. So it is wise to upload that piece again. In other words, if a seed has received Γ requests for the same piece, the $(\Gamma+1)^{th}$ request should be counted as *fresh* and so, we set $\beta = \frac{2}{\Gamma+1}$ so the oldest upload decision for this piece gets to be forgotten.

As shown in our simulation study, the value of β should depend on the level of burstiness of the system. The challenge here is that the scheduling is an online process. If the seed does not know in advance how dynamic the peerset is, a static value of β may only work for certain cases. For classic BitTorrent overlays, where burstiness values are not extremely high [10], a value of $\beta = \frac{2}{80+1} \cong 0.0247$ is a good heuristic.

Notice that $0 \leq \beta \leq 1$ and that, the difference between BitTorrent and $PFS_{\beta=1}$ (no memory of the history at all), is that $PFS_{\beta=1}$ serves the pieces requested the most in each round, considering *all* its connections (peerset), while seeds in BitTorrent apply *FCFS* and do round robin over the leechers in its peerset. Moreover, if the peerset of the seed is static, $\beta \to 0$, *i.e.* remembering the whole history, is the best choice.

A more elaborate solution would be to use a control approach to estimate β. We leave this approach for future work.

C. Our PFS Algorithm

In this subsection we present formally the *Proportional Fair Seed Scheduling*. We have implemented this algorithm in a new file sharing client that we call BUTorrent [12].

Input: Γ (seed connections), T_{rc} (re-choking interval), T (timeout for collecting requests).
Output: Set of M pieces to schedule per round.
while *Seed S has connected leechers* **do**
  if *Rechoking interval time T_{rc} expires* **then**
    foreach *Leecher i in peerset of S* **do**
      unchoke i;
    end
    while *S receives requests for pieces within T* **do**
      update vector of requests $R = \{r_1, \ldots, r_p\}$
      foreach *q = 1, ..., M* **do**
        S sends piece q (seed applies FCFS);
        update R and τ ;
      end
    end
    if *collected requests $> M$* **then**
      $\beta \leftarrow \frac{2}{\Gamma+1}$
      foreach *k = 1, ..., M* **do**
        $i_k^* \leftarrow \arg\max_i \left\{ \frac{r_i}{\tau_i + \epsilon} \right\}$ (break ties at random)
        $r_{i_k^*} \leftarrow r_{i_k^*} - 1$,
        foreach $j = 1, \ldots, p$ **do**
          if $j = i_k^*$ **then**
            $I_j = 1$;
          else
            $I_j = 0$;
          end
          $\tau_j \leftarrow \beta\ I_j$ + (1-β) τ_j ;
        end
      end
      S chokes the $N - M$ peers whose request were not scheduled ;
    end
    Keep serving leechers which requested $i = i_k^*$;
    Update R and τ as above every uploaded piece;
  end
end

Algorithm 1: Seed Scheduling in BUtorrent [12].

In order to have a global view, the seed unchokes every leecher in its peerset, allowing them to request their LRF. Then a timer T is started. The first M requests (in BitTorrent and in our experiments $M = 4$) are served by the seed as BitTorrent does (FCFS). This is because the collection of requests may take some time due to congestion in the underlay network, and we do not want to waste seed uploading capacity. Moreover, some of the Γ requests the seed is expecting may never arrive. That is why we need the timer T: leechers send their "interested" messages to all their neighborhood peerset, so it is possible that in the time between the "interested" message arrives at the seed and the unchoke message is sent by the seed, a leecher may have started downloading its remaining pieces from other peers, having nothing else to request from the seed.

When the timer T expires, PFS is run on the collected

requests. Since the seed upload capacity has to be divided by M, if the collected requests are less than M there is no choice to make and so no reason to apply PFS. After choosing the best PFS pieces, up to M other uploads begin, namely, the seed does not interrupt the upload of the first M pieces requested during the time T. Lastly, the $N - M$ requests that were not selected, are freed by choking the requesting peers again. After the PFS decision, for a time T_{rc} the M peers whose requests were selected are kept unchoked and so they may request more pieces. So M PFS decisions are made every T_{rc}. [2]

V. Analytical Results

In this section we give theoretical evidence that introducing smarter scheduling at the seed may be highly effective in terms of performance. We also show how the effectiveness of file sharing η (defined in 4), that our algorithm improves reducing downloading time, can capture the effect of seed scheduling using the fluid model in [4].

A. Collisions during the initial phases

In Section III we have defined the initial phase problem (Section III-C) showing with an example (Section III-D) how serving the same piece in a round to two different connected leechers may result in a suboptimal use of the seed uploading capacity. We have thus devised PFS (Section IV-C) to determine which requests the seed should satisfy, limiting the negative effects of what we have defined collisions (*Definition* 8). In this subsection we show theoretical bounds on the advantage of using PFS over a FCFS scheduling approach, in reducing both these collisions and therefore the earliest content uploading time (*Definition* 9).

Leechers follow the LRF piece selection algorithm, $i.e.$ in each scheduling round, every leecher makes a request for a piece randomly chosen among its equally rarest missing pieces in its direct neighbors. The seed scheduling thus depends heavily on the current leechers' connectivity, and, as we have shown earlier in Section III-E, solving it optimally in terms of the earliest conent uploading time is NP-hard, even when the overlay is static and known to the seed.

We assume the worse case scenario of only one seed and to simplify our analysis, we consider, as in [4], uniform upload and download capacity among the M concurrently active connections. We denote the total number of leechers as L. We also assume a fully connected overlay, so that swarming is most effective. Under this last assumption, a piece downloaded by one leecher is not requested again from the seed by other leechers.

Under the assumptions given above, we can quantify the expected number of collisions in any particular round by using a variant of the Birthday Paradox adapted to our scenario.

[2]The choice of T_{rc} is not trivial and it should not be static as overlays are not. We briefly show this point in our simulations, though leaving an extensive exploration of this parameter as an open question. BitTorrent implementation has it set to 10 seconds.

Lemma 3: If L leechers have to choose a piece uniformly at random from p pieces, then the expected number of pairs of leechers requesting the same piece is $\binom{L}{2}\frac{1}{p}$.

Proof: Consider n^2 indicator random variables X_{ij} for $i, j = 1, \ldots, n$, where $X_{ij} = 1$ if leecher i and j request the same piece and 0 otherwise. Since every leecher chooses a piece at random, $\Pr[X_{ij} = 1] = 1/p$ and therefore, $E[X_{ij}] = 1/p$.

Consider the random variable $X = \sum_{i<j} X_{ij}$ estimating the total number of collisions. Then, the expected number of total collisions is given by $E[X] = \sum_{i<j} E[X_{ij}] = \binom{L}{2}1/p$. ■

Let us consider an initial phase of a torrent with one seed and L fully connected leechers, trying to download a file with p pieces. [3] Let P^l denote the pieces yet to be uploaded at the beginning of round l; note how, for $l = 1$ we have $P^1 = \{1, \ldots, p\}$, and $P^{l+1} \subset P^l$ constantly reduces in size until the initial phase is over at round r when $P^r = \emptyset$. According to our assumptions, once the seed uploads a piece to some leecher, other leechers do not request it from the seed but obtain it from one of the other leechers. Thus, every leecher requests from the seed a piece chosen uniformly at random from P^l. Therefore, according to the Lemma 3, as l increases, the size of the set P^l decreases, and so the expected probability of having a collision is higher in later rounds. Note how the number of collisions even in the earlier rounds increases as the number of pieces p becomes comparable to the number of leechers L. In particular, the expected number of collisions are larger than zero even in the first round when $p < \binom{L}{2}$.

Let us now show the two improvement bounds induced by the use of our proportional fair heuristic: first, on the number of wasted upload slots due to collision events that may arise, and second, on the earliest content uploading time (Definition 9). Note that the performance in terms of both collisions and earliest content uploading time of PFS are never worse than FCFS. In FCFS in fact, the M pieces requested and served may or may not have duplicates depending upon the nature of the overlay and arrival order of requests. Therefore, we may have the same of fewer collisions when using PFS, but not more. From a theoretical point of view this implies that, without complete knowledge of the overlay, we are only able to show an upper bound on the improvement.

Proposition 4: In a peer-to-peer network with one seed and homogeneous bandwidth, the PFS piece selection algorithm decreases, with respect to FCFS, the total number of collisions as well as the earliest content uploading time by a factor of at most M, where M is the number of simultaneously active connections.

Proof: See appendix. ■

B. Effectiveness during initial phases

In this subsection we extend the Qiu-Srikant model [4] to capture the initial phase of a torrent, modeling the seed

[3]Any small-world network, such as those exhibited by leechers as evident in recent experiments [23], would show a similar connectivity.

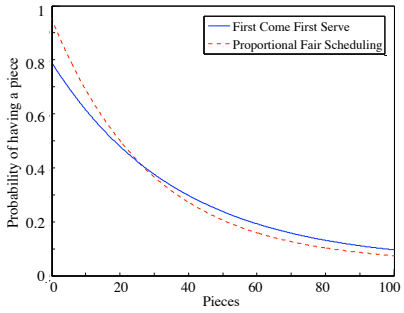


Fig. 2. The number of pieces each downloaders has in the initial phase of a torrent follows a power law distribution. In particular, the skewness α of the Zipf distribution is $\alpha = 1.4$ for a first come first serve ala BitTorrent and $\alpha = 1.9$ for our PFS.

scheduling effect through one of its crucial parameter, the effectiveness of file sharing η, defined in Section III. In particular, in [4] there is the assumption that each leecher has a number of pieces uniformly distributed in $\{0, \ldots, p-1\}$, where p is the number of pieces of the served file.

This assumption is invalid in the initial phase of a torrent. Consider Figure 2: 350 leechers join at once a torrent for downloading a file of $400MB$ with only one seed. We stop the simulation before leechers can possibly reach the steady state. We notice that (i) the number of pieces each downloaders has in the initial phase of a torrent follows a power law distribution and (ii) using the proportional fair scheduling, the skewness α of the Zipf distribution is higher ($\alpha = 1.4$ for a first come first serve ala BitTorrent and $\alpha = 1.9$ for our PFS). The higher is alpha, the higher is the probability that the a peer has interesting pieces for its neighbors, and so the higher are the swarming effects. Notice how for $400MB$, with a piece size of 256 we have 1600 pieces, but the probability of having more than 100 is less than 0.1. More precisely, the simulation is being stopped after 1600 seed scheduling decisions, *i.e.*, the time needed from an ideal round robin seed scheduling to inject the whole content.

The Qiu-Srikant fluid model captures the evolution of seeds $y(t)$ and leechers $x(t)$ in the overlay. Let λ be the new leechers arrival rate, c and μ the download and upload bandwidth of all leechers, respectively, θ the rate at which downloaders abort the download, γ as the seed abandon rate and η as effectiveness. From [4] we have:

$$\left\{ \begin{array}{l} \frac{dx}{dt} = \lambda - \theta x(t) - min\{cx(t), \mu(\eta x(t) + y(t)), \\ \frac{dy}{dt} = min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t). \end{array} \right\} \quad (6)$$

Our experiments revealed that the number of pieces that a leecher has in initial and churn phases is *not* uniformly distributed but follows a power law distribution. For lack of space, we do not show such experiments. So we have:

$$\eta = 1 - P \left\{ \begin{array}{c} \textit{leecher } i \textit{ has no} \\ \textit{piece useful for its peerset} \end{array} \right\},$$

and so for two leechers i and j:

$$\eta = 1 - P \left\{ \begin{array}{c} \textit{leecher } j \textit{ needs no} \\ \textit{piece from leecher } i \end{array} \right\}^k = 1 - P^k,$$

where:

$$P = P \left\{ \textit{ leecher } j \textit{ has all pieces of leecher } i \right\}.$$

Thus:

$$P = \frac{1}{c^2} \sum_{n_j=1}^{p-1} \sum_{n_i=0}^{n_j} \frac{1}{(n_i\ n_j)^\alpha} \cdot \frac{\left(\begin{array}{c} p - n_i \\ n_j - n_i \end{array} \right)}{\left(\begin{array}{c} p \\ n_j \end{array} \right)}; \quad (7)$$

where $c = \sum_{i=1}^{p} i^{-\alpha}$ is the normalization constant of the Zipf distribution. Therefore we obtain:

$$\left(\begin{array}{c} p - n_i \\ n_j - n_i \end{array} \right) = \left(\begin{array}{c} p - n_i \\ p - n_i - n_j + n_i \end{array} \right) = \left(\begin{array}{c} p - n_i \\ p - n_j \end{array} \right);$$

hence,

$$P = \frac{1}{c^2} \sum_{n_j=1}^{p-1} \sum_{n_i=0}^{n_j} \frac{1}{(n_i\ n_j)^\alpha} \cdot \frac{(p-n_i)!\ n_j!}{(n_j - n_i)!\ p!} \quad (8)$$

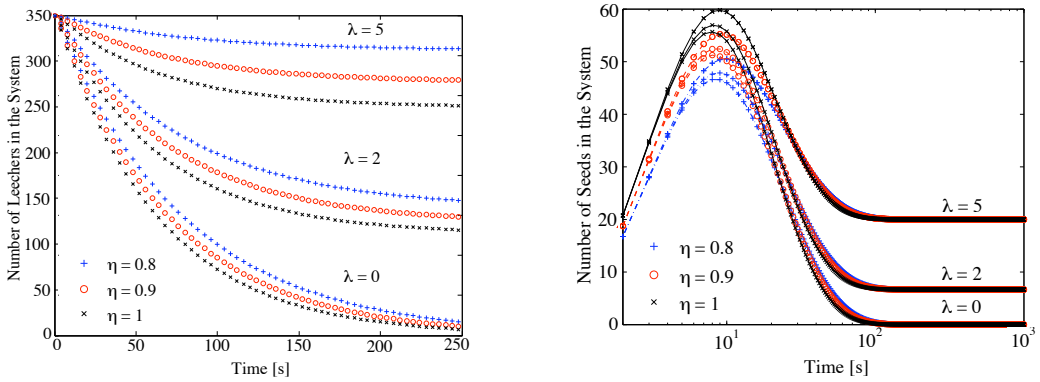
and so:

$$\eta = 1 - \left(\frac{1}{c^2 p!} \sum_{n_j=1}^{p-1} \frac{n_j!}{n_j^\alpha} \cdot \sum_{n_i=0}^{n_j} \frac{(p-n_i)!}{n_i^\alpha (n_j - n_i)!} \right)^k. \quad (9)$$

1) Leechers: In Figure 3(a), we present the evolution of the number of leechers ($x(t)$ in Equation 6). Admissible values of η depend on the typical values of skewness $\alpha \in [1, 4]$, plugged into Equation (9). Results in this section are obtained under the stability conditions described in [4]; in particular, $\theta = 0.001$, $\gamma = 0.2$, $c = 1$, $\mu = 1$ and $\lambda = \{0, 2, 5\}$ representing different levels of burstiness of the arrival process of new peers. Starting with one seed and 350 leechers, *i.e.* $x(0) = 350$ and $y(0) = 1$, set as in our later simulations, we plot the number of leechers in the system to study the impact of λ and the effectiveness η of file sharing.

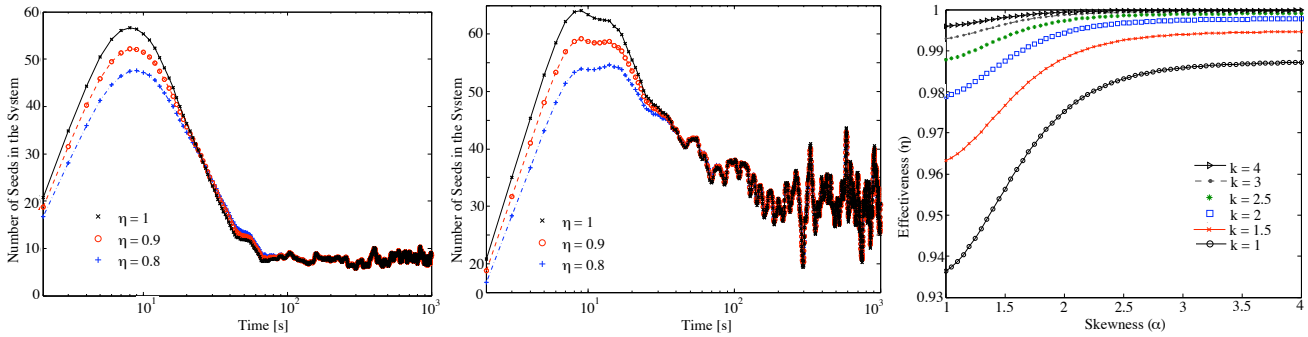
The analytical model provides the insight that for static peersets ($\lambda = 0$), the improvement we can achieve is limited even if we bring effectiveness to one through judicious seed scheduling. When burstiness increases, even a small improvement in η is significant in terms of reduction in total time to download. Note that shortest downloading time implies a smaller number of leechers in the system.

2) Seeds: In Figure 3(b), 4(a) and 4(b) we show, the impact of the effectiveness on the evolution of the number of seeds in the system ($y(t)$ in Equation 6). In Figure 3(a) three different constant leechers' arrival rate are represented. In Figure 4(a), we tested the impact of effectiveness for a random leechers arrival rate with average $\lambda = 10$ and in Figure 4(b) we show the seed evolution with random leechers' arrival rate and average $\lambda = 30$. The common results is that, independently from the level of peers arrival (burstiness), at the beginning of the torrent — the initial phase — higher values of effectiveness



(*a*) Leechers evolution (*b*) Seed evolution

Fig. 3. Burstiness Impact on Effectiveness (a) Leechers in the system decreasing faster over time indicates a shorter downloading time. We study here the impact of different rate of arrival λ on the effectiveness η of file sharing. For high arrival rate, higher effectiveness has a higher impact on the completion time of the initial 350 leechers. (b) Impact of effectiveness η on the evolution of the number of seeds in the system. Leechers departs when their download is complete. Constant leechers' arrival rate $\lambda = \{0, 2, 5\}$.



(*a*) $\lambda = 10$ (*b*) $\lambda = 30$ (*c*) outdegree k as parameter

Fig. 4. Skewness impact on effectiveness: (a) Impact of effectiveness η on the evolution of the number of seeds in the system. Leechers departs when their download is complete. Random leechers' arrival rate with average $\lambda = 10$. (b) Impact of effectiveness η on the evolution of the number of seeds in the system. Leechers departs when their download is complete. $\lambda = 30$. (c) For highly connected peersets (bottom curve only one neighbor, top curve 101 neighbors), the skewness α has less impact on the effectiveness (η) when the number of average active connection increase. The maximum number of connection is kept to 80 as in the original version of the BitTorrent protocol.

increment the number of seeds present at the same time in the system, namely, more leechers complete faster their download for higher effectiveness.

3) Skewness Impact on Effectiveness: As we showed in Figure 2, during the initial phase of a torrent the number of pieces a leecher has, follows a power law distribution. As the torrent approaches steady state, the skewness of the Zipf distribution grows until, it became reasonable to assume a uniform distribution among the pieces a leecher has. In Figure 4(c) we show the impact of the skewness on the effectinvess, having the average outdegree of a node k as parameter. k is measuring the number of active connections (downloads) a leecher maintains during its lifetime in the torrent. The first observation coming from Figure 4(c) is that, fixing k, the effectiveness grows with the skewness; this confirm that during a life time of a torrent, the skewness grows. The second observation is that the effectiveness grows with the average number of active connection a leecher has, namely, when a leecher on average is unchoked by more peers, it has more chance to get more pieces, therefore the effectiveness increases.

VI. Experimental Results

We performed a series of experiments to assess the performance of our Proportional Fair Seed Scheduling, both using the GPS simulator [11], creating our own random physical topologies, and over PlanetLab [13]. Overlay topologies are created by the tracker, which randomly connects together a maximum of eighty peers. In all the simulation experiments, we set a homogeneous bandwidth of 2 Mbps. For our PlanetLab testing we have implemented PFS on a real client [12], starting from the *mainline instrumented client* [24], and we left unconstrained the link bandwidth and measured, on average, about 1.5 Mbps. In all our experiments, there is only one seed and leechers leave when they are done. This choice of one seed was made to show results in the worse case scenario. In the following plots, 95% confidence intervals are shown.

A. Simulation Experiment 1: Average Downloading Time

Figure 5(a) shows the average downloading time of a 400 MB file for an overlay of 350 peers for different values of Burstiness (defined in Section III). For PFS, we set the forgetting factor β to 0.02 as explained in Section IV. We also show performance of Global Rarest First (GRF), where each peer is connected to every other peer. GRF serves as lower bound on the average download time. We observe that PFS improves up to 25% the leecher average downloading time, depending on the level of burstiness in arrival process.

To obtain a value of burstiness of 200 for example, we have used a peak rate of 10 meaning that, 10 leechers join the overlay in 1 second, for a simulation period of 7000 seconds. In this way we have an average arrival rate of 350 peers / 7000 s = 0.05 peers/s. With a peak rate of 10 and average arrival rate of 0.05, we obtain a burstiness level of $B = \frac{peak}{average} = \frac{10}{0.05} = 200$. When the burstiness level is high, leechers have too small peersets to make good use of swarming. When instead the burstiness is too low for the chosen value of β, then forgetting too fast has the same effect as not remembering at all (as BitTorrent does). Even for situations where there is no significant gain in time to download (the first point of Figure 5(a) shows $burstiness = 1$), Section VI-C shows that it is still beneficial to use PFS to reduce seed utilization.

B. Simulation Experiment 2: Seed with Global View

In Figure 5(b) we show results for a slight variation of the PFS algorithm — the seed is connected with, and so unchokes, all the leechers in the network. We see that the average time to download for different level of burstiness, with exactly the same simulation setting of the experiment described in Figure 5(a), has improved slighly.

A second observation is that, for unitary value of burstiness, the average downloading time does not change significantly regardless the scheduling algorithm. Section VI-C shows why it is useful to adopt a proportional fair scheduling strategy at the seed even for low level of burstiness.

C. Simulation Experiment 3: Seed Utilization

Figure 6 shows the utilization (defined in Section III) of the unique seed present for different file sizes and unitary burstiness value for case (a) and 25 in case (b). We note how: (i) with PFS, seeds are less congested by leechers' requests, and (ii) since leechers leave when they are done, a lot more requests are made to the seed at the end of the torrent when fewer connections are active.

Leechers find themselves alone at the end thus they all ask the seed being the only one left with interesting content. So seeds using PFS receive less requests due to better piece distribution. Moreover, lower seed utilization means higher effectiveness and so delay in reaching the phase where leechers are alone with the seed, (iii) the seed utilization is decreasing monotonically with the file size. When file size increases, there is more time for leechers to use swarming. Overall, requests to the seed are less if seed scheduling is smarter. For static peerset (every peer joins at the same simulation time), the seed is the only source and so it gets many requests, but many less if the scheduling algorithm is smarter (Figure 6(a)). For an higher value of burstiness instead (Figure 6(b)), we note smaller improvement of the seed utilization because smaller groups of peers at the time are downloading the file. In fact, we also notice fewer requests to the seed.

We conclude that, even for static cases or small values of burtiness that do not lead to a significant improvement in downloading time, a better seed scheduling guarantees a better seed utilization. Viceversa, we observed that, when the burstiness is high, we have less improvement in seed utilization but the average time to download is smaller due to the more effective piece distribution.

D. Fairness Index

In Figure 6(c) we show some fairness related results during the download of a short file (2 Mb) from a fairly small peerset (10 leechers). We use a standard measure of fairness, the Jain's fairness index, which is defined as:

$$J(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2} \tag{10}$$

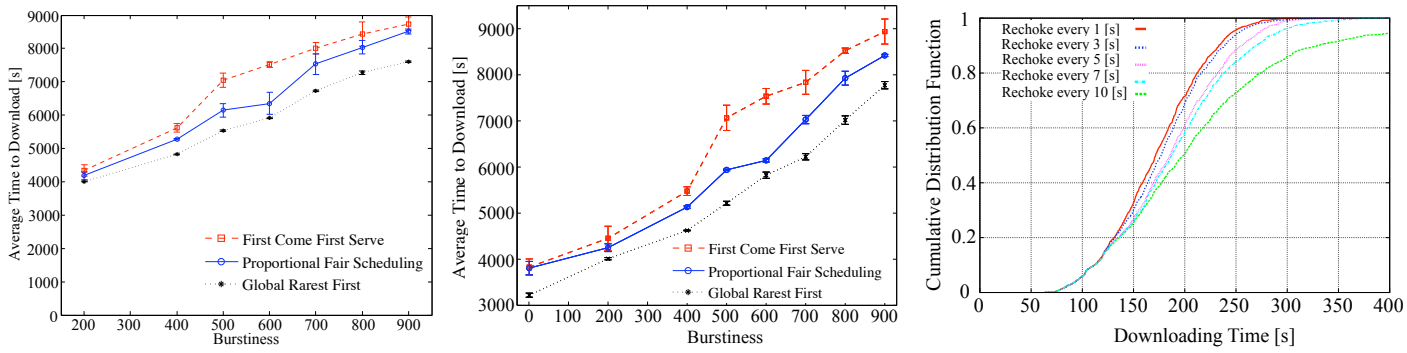
where x_i is the number of copies of piece i. The result ranges from $\frac{1}{n}$ (worst case) to 1 (best case). As always, we stopped the simulation when every leechers have completed the download, and we show the Jain's fairness index over time for both cases, using a first come first serve and a proportional fair scheduling at the seed.

The first result to note is that when some intelligence is applied at the seed, $i.e.$, when using a proportional fair scheduling, the piece distribution is more equalized and in fact, the fairness index is always higher. As we note in the first part of the graph, this implies that when the piece fairness is higher, the download is faster—when the Jain index reaches a unitary level, it means that all the considered peers have the same amount of pieces, and so they have completed the download. Using a first-come first-serve approach, the first 10 seconds are not enough for the first set of unchoked leechers to complete their download, while in less than 5 seconds the fairness index reaches its maximum value when the proportional fair scheduling is applied.

The second interesting results can be observed after the first rechecking interval, set to 10 seconds. Only M leechers are allowed to make a request in the first unchoking interval, and they quickly download all the pieces. After the first 10 seconds though, a new set of leechers's requests lower the Jain's fairness index, till again the steady state of the piece distribution is reached, because also the new set of unchoked leechers has the opportunity to download.

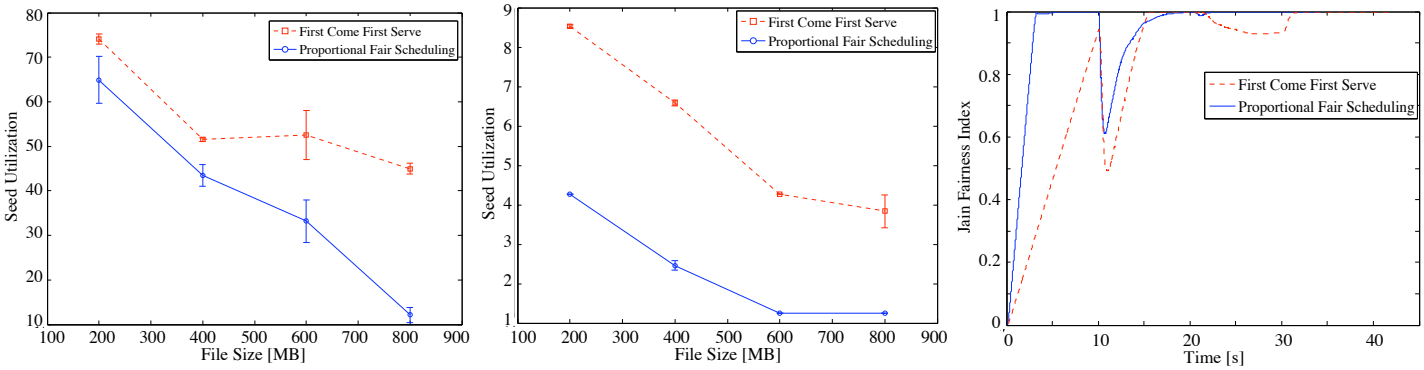
This observation has an important consequence it the design of peer-to-peer protocols: in particular, the rechocking interval should not be fixed to 10 seconds but it should be adoptive with the size of the peerset and with their capacity.

The third results, related with the previous two observations, concerns the total downloading time. Although using our



(*a*) ADT (*b*) ADT (seed connected to every leecher) (*c*) Peer selection frequency

Fig. 5. (a) Average Download Time (ADT) for different burstiness values: 350 peers are sharing a 400 MB file. Leechers always depart when they are done with their download. The proportional fair scheduling guarantees up to 25% of improvement over the First Come First Serve approach used by the BitTorrent protocol. (b) Average download time for different burstiness values and seed with full view over the peerset. 350 peers sharing a 400 MB file. Leechers always depart when they are done with their download. The proportional fair scheduling guarantees up to 25% of improvement over the First Come First Serve approach used by the BitTorrent protocol. (c) Applying the peer selection algorithm more often helps the average download time.



(*a*) Seed Utilization with Burstiness B = 1 (*b*) Seed Utilization with B = 25 (*c*) Jain Fairness Index

Fig. 6. Seed Utilization with 350 peers. Seed applying PFS is less congested due to better piece distribution. (a) Burstiness = 1. (b) Burstiness = 25. Note the different scale. (c) Jain's Fairness index for a static peerset in the initial phase of a torrent. PFS guarantees a more equal piece distribution. After seed unchoking interval, PFS recovers fairness faster.

PFS algorithm, it is barely visible that the total downloading time of the all the peers (or the slowest peer) ends (about 5 seconds) earlier, it is more important to note how, when using PFS, there are periods during the simulation in the which no leechers downloads any piece only because the rechoking interval protocol parameter is set to too high values. This means that no other leechers except the first (M=4 in BitTorrent) is allowed to request any piece.

This observation led us to analyze even the effect of tuning the rechoking interval parameter, result that we report in the next subsection.

E. Rechoking Interval

As a last simulation result, we show in Figure 5(c) the impact of the rechoking interval — the time period after which every peer applies the peer selection algorithm again — on the average downloading time.

We include this result to suggest an interesting research direction more than an actual performance analysis of PFS. In most of the file sharing protocols ala-BitTorrent, the rechoking interval period has been (mysteriously) set to 10 seconds. Although for the rest of our simulations we kept invariant this choice to analyze separately each contribution, we note how it can impact on the average download time.

We have simulated an overlay of 350 peers among which only one is a seed, all interested in downloading a file of 20 MB, with homogeneous bandwidth. As we can see, even if the peer arrives all at the beginning, and no seed is leaving the peerset when it has completed its download, applying the peer selection algorithm with different frequency has a non-negligible impact on the average downloading time. It is important to notice that the simulation takes into account the overhead of the control message. The take home message here is that, refreshing the peers view by increasing the frequency at which the best leechers are selected, helps the average download time. Although it is intuitive that this parameter should be adopted dynamically, depending on the peers rate of arrival and departure, we can see how, even in static cases, an more frequently updated view of the system helps the downloading speed.

PlanetLab Results			
Protocol	Burstiness	ADT [s]	Improvement
BitTorrent	No	839	–
	Low	1328	–
	High	2183	–
BUtorrent	No	733	**12.6 %**
	Low	1120	15.6 %
	High	2171	0.5 %
BUtorrent ($\beta = 0.02$)	No	809	3.5 %
	Low	1031	**22.3 %**
	High	1924	**11.8 %**

TABLE I
PLANETLAB EXPERIMENT WITH A PEERSET OF 350 ACTIVE LEECHERS AND A UNIQUE INITIAL SEED. IN DYNAMIC OVERLAYS THE FORGETTING FACTOR β BRINGS IMPROVEMENT IN ADT.

We therefore suggest as interesting research direction the design of adaptive protocols that dynamically change seed parameters given the network conditions.

F. Planetlab Experiment

To validate our results, we tested our BUtorrent on PlanetLab. We run simultaneously the scheduling algorithms we wish to compare to minimize the difference in bandwidth available to different experiments.

We ran a set of experiments with 350 PlanetLab nodes sharing a 400 MB file and we report the results in table VI-F. We found that BUtorrent improves the average download time by 11.8% over BitTorrent in static overlays (no burstiness), 22.3% improvement for low burstiness ($B = 400$) and 12.6% improvement for high burstiness ($B = 800$).

We show the cumulative distribution function of downloading time for static — Figure 7 (a) — and dynamic peersets — Figure 7(b). In Figure 7(a), a file of 400 MB is shared among the 350 leechers in a static peerset case. As always in our experiments, leechers departs after their download is complete. Note how the forgetting factor β decrease performance, and also that there is not much to improvement for the group of leechers, with lower download capacity, that are rarely unchoked by other leechers.

In Figure 7(b) we introduce a burstiness level of 400. We can see how that the forgetting factor plays a pretty crucial rule in improving the downloading time.

For such dynamic scenarios, the effect of the slow peers have less impact because in burstiness cases, smaller group of peers must collaborate using swarming to finish as fast as possible. Namely, when there is a small group of peers to download from, leechers do not have much choice when they apply the peer selection algorithm.

Notice also that the average downloading time is higher when the peerset is dynamic because many leechers arrive later in the torrent.

VII. RELATED WORK

Swarming techniques have received strong interest by the research community, peer-to-peer traffic being a consistent amount of the whole internet traffic [25]. *"Between 50 and 65% of all download traffic is P2P related. Between 75 and 90% of all upload traffic is P2P related"* [26]. The most popular protocol using swarming is BitTorrent [1].

A. Improving peer strategies

The BitTorrent protocol has established swarming as one of the most fresh and promising ideas in contemporary networking research and thus has kicked-started a (tidal) wave of research articles in the area. Many proposals on how to improve the BitTorrent protocol, by modifying the behavior of leechers, have therefore appeared in literature. Recent work [27] argues for greedy strategies, we cite BitTyrant [2] as an example. Other work [3] considered game theoretical approaches. Our focus is only on the seed without modifying leechers.

Smartseed [5], [21] applies some strategies at the seed to boost performance. However, Smartseed is not backwards compatible with the BitTorrent protocol, as opposed to our seed scheduling proposal that keeps every other fundamental algorithm of BitTorrent intact. Moreover, Smartseed does not take into account dynamic scenarios, one of the key aspects of our results. Another unpublished approach involving the seed is given by the *superseed* mode of Bittornado [28]. The goal of *superseed* is to enable under provisioned seeds to effectively inject content. An experimental study of its performance can be found in [29].

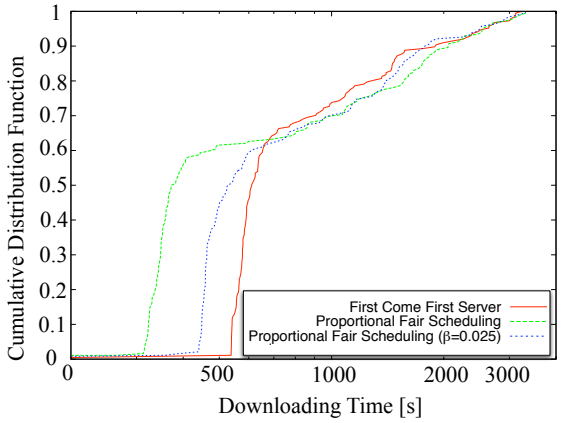
A recent solution whose main goal is to reduce the seed load has been discussed in [30]. The authors propose to estimate the popularity of each of the pieces — using information from its directly-connected neighbors and by probing additional peers — to later decide which piece to advertise as available to its neighbors. Although this policy is effective in limiting the seed uploads, it has to deal with the trade-off of extending the peer download time up to 3 times.

B. Scheduling, churn and transient phases

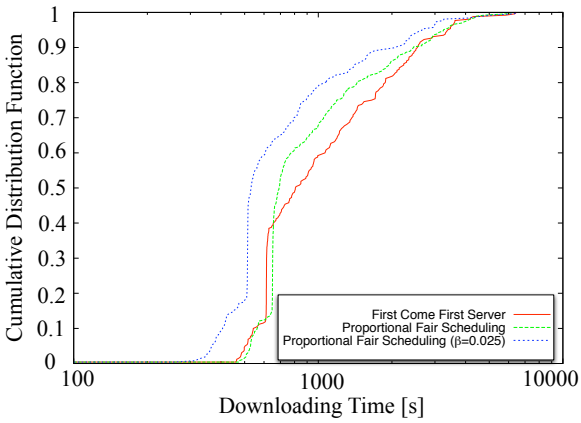
More generally, the authors in [31] show strategies for selecting resources that minimize unwanted effects induced by churn phases. In our case the resources are the pieces that the seed has to schedule.

Scheduling for BitTorrent is also discussed by Mathieu and Reynier in [9], which analyzes starvation in flash-crowd phases. Using optimization theory, an uplink-sharing fluid model whose goal is to find the optimal scheduling (piece selection) for peer-to-peer networks has been investigated in [8] and used in [32] and [33]. These approaches focus either on the end game mode, where leechers are missing their last pieces, or on the analysis of the minimum distribution time. Our system-oriented approach targets the minimization of the average downloading time by exclusively minimizing the transient phases.

Measurement studies were also carried out, with focus on BitTorrent transient phases [17]. The goal is to understand, given a peer-to-peer system, how quickly the full service capacity can be reached after a burst of demands for a particular content, namely, how long the system stays in the



(*a*) Planetlab with static peerset



(*b*) Planetlab with dynamic peerset

Fig. 7. (a) Cumulative Distribution Function (CDF) of the downloading time for 350 leechers downloading a file of size 400 MB and joining the overlay at once (no burstiness). Leecher depart when they finish their download. (b) CDF of the downloading time for 350 leechers downloading a file of size 400 MB. Burstiness value of 400. Leechers depart when they finish their download.

transient phase. We studied initial phases using the fluid model adopted by Qiu and Srikant in [4]. We fixed their notion of effectiveness to capture seed scheduling effects during initial phases.

C. Fairness

Even though the concept has been applied on the peer and not on the piece selection, a recent solution that shows how considering fairness may help speeding up the leechers downloading time was discussed in FairTorrent [34].

VIII. Conclusion

In this work we considered the piece selection of content distribution protocols ala BitTorrent. We studied analytically and we provided simulation and real evidence that improving the scheduling algorithm at the seed can be crucial to shorten the initial phase of a torrent therefore reducing the average downloading time of the leechers. Our idea is to give seeds a more global view of the system supporting and not substituting the Local Rarest First piece selection algorithm used by BitTorrent like protocols. We devised our seed scheduling algorithm, inspired by the Proportional Fair Scheduling [20], implementing it into a real file sharing client that we call BUtorrent [12]. We found in simulation and PlanetLab experiments that BUtorrent, in dynamic overlays, increases the effectiveness of file sharing, reduces the congestion of requests at the seed improving up to 25% the average downloading time over BitTorrent.

Appendix - Proof of Proposition 4

Let us consider the initial phase of a torrent with one seed and L fully connected leechers, trying to download a file with p pieces. We consider *Case A*, where the seed is using FCFS to choose which requests to satisfy and *Case B* where the seed is using PFS. We compute the expected number of collisions in the initial phase for each case and so the claim.

Case A: In the worse case the first M requests to the seed could turn out to be for the same piece. This would result in the most possible collisions in each round, namely, the seed may end up uploading the same piece for all its M upload slots i.e. $r = p$ and $|P^{l+1}| = |P^l| - 1$ for all l. The expected number of total collisions in the initial phase would then be:

$$\begin{aligned}\binom{L}{2}\sum_{i=1}^{p}\frac{1}{|S^i|} &= \binom{L}{2}\sum_{i=1}^{p}\frac{1}{p-(i-1)} \\ &= \binom{L}{2}\sum_{j=1}^{p}\frac{1}{j} \approx \binom{L}{2}\ln p\end{aligned}$$

Case B: However, if the seed is following PFS, it will be able to upload M distinct pieces in its M upload slots in most rounds (in the final few rounds there will be less than M distinct pieces to upload and collisions can't be avoided). Therefore, for this case, the number of rounds in the initial phase reduces to p/M and the expected number of collisions will reduce by a factor of M since:

$$\begin{aligned}\binom{L}{2}\sum_{i=1}^{p/M}\frac{1}{|S^i|} &= \binom{L}{2}\sum_{i=1}^{p/M}\frac{1}{p-M(i-1)} \\ = \binom{L}{2}\sum_{i=1}^{p/M}\frac{1}{M}\frac{1}{p/M-(i-1)} &\approx \binom{L}{2}\frac{1}{M}\ln p/M\end{aligned}$$

■

Acknowledgment

We are grateful to professor Steve Homer, Ilir Capuni, Alessandro Duminuco and Heiko Röglin for their valuable comments, and to Damiano Carra and Nobuyuki Mitsutake for their previous work on this project.

References

[1] B. Cohen, "Incentives build robustness in bittorrent," bittorrent.org, Tech. Rep., 2003.

[2] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *Proceedings of USENIX NSDI 2007*. USENIX, April 2007.

[3] D. Levin, K. Lacurts, N. Spring, and B. Bhattacharjee, "BitTorrent is an auction: analyzing and improving BitTorrent's incentives," in *SIGCOMM*, vol. 38, no. 4, New York, NY, 2008, pp. 243–254.

[4] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *SIGCOMM*. New York, NY: ACM, 2004, pp. 367–378.

[5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *INFOCOM*, 2006.

[6] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-to-peer file dissemination," *J. of Scheduling*, vol. 11, no. 2, pp. 105–120, 2008.

[7] OpenP4P. [Online]. Available: http://www.openp4p.net

[8] J. Mundinger, R. Weber, and G. Weiss, "Optimal scheduling of peer-topeer file dissemination," *J. Scheduling*, 2006.

[9] F. Mathieu and J. Reynier, "Missing piece issue and upload strategies in flashcrowds and p2p-assisted filesharing," in *Telecommunications, 2006. AICT-ICIW.*, 2006, p. 112.

[10] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Proceedings of the 5th Passive and Active Measurement Workshop*, 2004.

[11] W. Yang and N. Abu-Ghazaleh, "Gps: a general peer-to-peer simulator and its use for modeling bittorrent," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005. 13th IEEE International Symposium on*, 2005, pp. 425–432.

[12] [Online]. Available: http://csr.bu.edu/butorrent

[13] B. N. Chun, D. E. Culler, T. Roscoe, A. C. Bavier, L. L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[14] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, pp. 440–442, 1998.

[15] 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 *ICDCS*, 2006.

[16] [Online]. Available: http://aws.amazon.com/s3/

[17] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," in *INFOCOM*, 2004.

[18] E. Cockayne and S. Hedetniemi, "Optimal domination in graphs," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 11, pp. 855–857, Nov 1975.

[19] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[20] H. Kushner and P. Whiting, "Convergence of proportional-fair sharing algorithms under general conditions," *Wireless Communications, IEEE Transactions on*, vol. 3, 2004.

[21] P. Michiardi, K. Ramachandran, and B. Sikdar, "Modeling seed scheduling strategies in BitTorrent," in *Networking 2007, 6th IFIP international conference on Networking, May 14 -18, 2007, Atlanta, USA — Also published as LNCS Volume 4479*, May 2007.

[22] J. C. B. and H. Burkill, *A Second Course in Mathematical Analysis*. Cambridge University Press, 2002.

[23] C. Dale, J. Liu, J. Peters, and B. Li, "Evolution and enhancement of bittorrent network topologies," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, 2008, pp. 1–10.

[24] http://www-sop.inria.fr/planete/Arnaud.Legout/Projects/.

[25] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *IEEE Network*, vol. 17, pp. 6–16, 2003.

[26] "http://torrentfreak.com/peer-to-peer-traffic-statistics/. 2010."

[27] D. Carra, G. Neglia, and P. Michiardi, "On the impact of greedy strategies in bittorrent networks: the case of bittyrant," in *IEEE P2P*, 2008.

[28] [Online]. Available: http://bittornado.com/docs/superseed.txt

[29] Z. Chen, Y. Chen, C. Lin, V. Nivargi, and P. Cao, "Experimental analysis of super-seeding in bittorrent," in *Communications, 2008. ICC '08. IEEE International Conference on*, 19-23 2008, pp. 65 –69.

[30] B. Sanderson and D. Zappala, "Reducing source load in bittorrent," *Computer Communications and Networks, International Conference on*, vol. 0, pp. 1–6, 2009.

[31] B. P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 147–158.

[32] R. Kumar and K. W. Ross, "Peer-assisted file distribution: The minimum distribution time," in *in HOTWEB, Boston, USA*, 2006.

[33] V. B. A. Sweha, Raymond; Ishakian, "Angels in the cloud – a peer-assisted bulk-synchronous content distribution service,," Boston University, TR 2010-024, Tech. Rep., Aug 2010.

[34] A. Sherman, J. Nieh, and C. Stein, "Fairtorrent: bringing fairness to peer-to-peer systems," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*. New York, NY, USA: ACM, 2009, pp. 133–144.