2009 Eighth IEEE International Symposium on Network Computing and Applications

# Seed Scheduling for Peer-to-Peer Networks

Flavio Esposito* Ibrahim Matta* Pietro Michiardi† Nobuyuki Mitsutake* Damiano Carra‡

*Computer Science Department
Boston University
Boston, MA
{flavio, matta, mnobu}@cs.bu.edu

†Eurecom Institute
Sophia Antipolis, France
pietro.michiardi@eurecom.fr

‡Computer Science Department
University of Verona
Verona, Italy
damiano.carra@univr.it

***Abstract*—The initial phase in a content distribution (file sharing) scenario is delicate due to the lack of global knowledge and the dynamics of the overlay. An unwise distribution of the pieces in this phase can cause delays in reaching steady state, thus increasing file download times. We devise a scheduling algorithm at the seed (source peer with full content), based on a proportional fair approach, and we implement it on a real file sharing client [1]. In dynamic overlays, our solution improves by up to 25% the average downloading time of a standard protocol ala BitTorrent.**

## I. Introduction

### A. Problem and Motivation

Swarming techniques have been widely studied recently due to the vast number of applications that a decentralized swarming network such as peer-to-peer systems enable. In particular, the purpose of these studies is to make content distribution protocols efficient and robust. See BitTorrent [2] and references therein.

However, most of the literature focus, even recently [3], has been both on the analysis, and on the improvement of downloaders in their strategies for selecting which neighbor (peer) to download from, and which part of the content (piece) to download next.

Measurements [4], simulation [5], and analytical studies [6] on BitTorrent-like protocols have also shown that, even though peers cooperate, leveraging each other's upload capacity, this operation is not done optimally in every possible scenario.

Other studies [7], [8] show that there are source bottlenecks due to a non smart piece distribution during initial phases and churns — a churn is a transient phase characterized by a burst of simultaneous requests for a content. In fact, during churns, 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 a piece-level fairness via a proportional fair scheduling at the source of the content to ensure that different pieces are uniformly distributed over the peer-to-peer network, and thus peers can effectively exchange pieces among themselves as quickly as possible. Specifically, when the overlay is dynamic, we show that this translates into shorter average time to download.

### B. Contributions

The main contributions of this work are summarized as follows:

- We devise and implement on a real file sharing client (we call our modified file sharing client BUtorrent [1]) a seed scheduling algorithm, that improves in dynamic overlays, by up to 25% the average downloading time.
- We adapt the analytical fluid model in [4] so it is valid even during initial and churn phases, capturing the effect of seed scheduling.
- We show that a smarter seed scheduling leads to higher probability for downloaders to make use of swarming to finish their downloads.

### C. Paper Organization

The rest of the paper is organized as follows: In Section II we give an overview of the BitTorrent protocol, defining notions that we will use for the rest of the paper. Section III describes the problem of source scheduling of pieces to allow downloaders to finish as fast as possible, and points out why and when a file sharing protocol ala BitTorrent does not work well in dynamic situations. In Section IV we describe our solution: a new seed scheduling algorithm for BitTorrent. In Section V we study analytically why scheduling at the seed is important to reduce the time to download and Section VI validates our analysis on swarming effectiveness experimentally. Section VII discusses related work and finally Section VIII concludes the paper.

## II. BitTorrent Overview

BitTorrent is a file sharing protocol for content dissemination. The content is divided into pieces (256 KB each) so that peers not having the whole content can speed up their download by exchanging pieces among themselves (swarming). A peer-to-peer system running a swarming protocol calls the peers interested in downloading pieces *leechers*, and peers that only act as servers (i.e., only upload content) *seeders* or *seeds*. In this work, we consider a *torrent* $T$ to be a set of at

This work has been partially supported by National Science Foundation awards: CISE/CCF #0820138, CISE/CSR #0720604, CISE/CNS #0524477, CNS/ITR #0205294, and CISE/EIA RI #0202067.

978-0-7695-3698-9/09 $25.00 © 2009 IEEE
DOI 10.1109/NCA.2009.15

least one seed, many leechers, a tracker, which is a special entity that helps bootstrap the peer-to-peer network, and a file *F*, split into $p$ pieces, which has to be distributed to all leechers.

**Definition (Neighborhood)**: *given a peer $v$, we define as neighborhood or peerset of $v$, $N_v$, the set of peers directly connected to $v$, whose size $|N_v| = k_v$.*

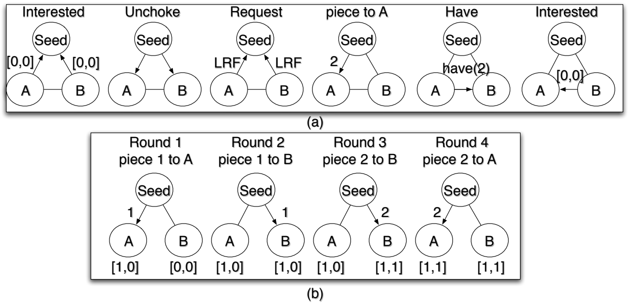

Fig. 1. (a) BitTorrent protocol: from left to right, the sequence of messages. (b) Global information leads to faster content distribution. Peers unaware of each other cannot use swarming.

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 $N_v$ (only). Amongst all the interested leechers, only *four* peers are given an opportunity to download data, using the *choke* algorithm. The choke algorithm, applied every re-choking interval $T_{rc}$,[1] is simply a round robin among the peers in $N_S$, in case a seed $S$ is running it, while choking is based on two mechanisms for leechers: (i) tit-for-tat and (ii) optimistic unchoke. The tit-for-tat peer selection algorithm adopted by leechers, 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. Because at the beginning, no leecher has pieces to upload and because leechers prefer to unchoke faster peers so they can download and collaborate faster, one leecher (potentially up to four leechers initially) is selected at random to explore potentially faster leechers (optimistic unchoke). In Figure 1(a) only two leechers (A and B) are in $N_S$ so both are unchoked.

Unchoked leechers select one piece of the content they want to download using the Local Rarest First ($LRF$) algorithm. A leecher $v$ applying $LRF$ counts, thanks to the bitfield dissemination, how many copies of each piece are present in $N_v$ and requests the rarest. 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 leecher $A$. Upon reception of a piece, leechers inform all their neighbors with a "have" message. Now, if leecher $B$ does not have piece 2 yet, $B$ sends an interested message to $A$.

**Definition (initial phase)**: *Given a torrent* T*, and a file* F *split into* p *pieces, we define the initial phase of* T *for* F *to be the time interval between the first scheduling decision made by a seed, to the time all distinct $p$ pieces of* F *have been completely uploaded.*

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

## III. Protocol Weaknesses and Problems

**We consider the problem of seed scheduling the piece whose injection guarantees the maximum benefits for the overlay.** In this section we analyze why this is a challenge, and when this is an important factor. To do so, we need the following definitions:

**Definition (Effectiveness):** *Given a peer-to-peer system, we define effectiveness of a file sharing $\eta$ as the probability that a leecher $v$ has at least an interesting piece for its neighborhood $N_v$.*

We discuss this effectiveness metric in detail in Section V-B.

**Definition (Burstiness):** *Given a dynamic peer-to-peer system, we define burstiness as the ratio of the peak rate of leechers' arrival to the average arrival rate during a period of observation.*

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 (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 (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 the Clustering Coefficient for G as* [9]:

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

Although BitTorrent has been widely studied and recognized as a robust protocol, it is far from being ideal in every scenario. In the presence of burstiness for example, $i.e.$, churn phases, the effectiveness of swarming can be too low [10]. Moreover, for overlays with low clustering coefficient, *seed utilization* can be surprisingly high. Therefore it is natural to think about improvement by adding intelligence to the seed as opposed to investigating techniques that modify leechers' strategies ($e.g.$, [3]).

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

[1] $T_{rc} = 10$ seconds in most implementations.

the scheduling at the seed, leaving the rest of the BitTorrent protocol intact.

### 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* [11] service, higher seed utilization results in higher monetary costs for using Amazon resources (bandwidth, CPU, etc.)

### B. Dynamic Overlays

Another problem is 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 (defined in Section II). During the initial phase of a torrent, the seed is one of the few interesting peers. Recalling how the tit-for-tat mechanism works (Section II), we realize that in this phase, even though swarming techniques are used, leechers have not yet downloaded many pieces, so every leecher will most probably ask the seed for its missing pieces, making the seed a bottleneck. 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. Moreover, when a new group of leechers joins the overlay (churn phase), no leechers is willing to unchoke peers with no pieces due to the tit-for-tat mechanism, with the seeds being the exception, and so another initial phase effectively takes place. We are not the first to claim that source bottlenecks occur due to inefficient piece dissemination during flash crowds and churn phases [12]. So during *all* the intial 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 (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 $i$.*

Notice 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 (wasted upload), because one leecher could have downloaded the colliding piece using swarming.

**Definition (Earliest Content Uploading Time):** *We define earliest content uploading time, to be 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. If a collision occurs, 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 (pieces) 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 a seed scheduling algorithm that attempts to overcome the lack of leechers' awareness of each other, improving piece fairness, and we show that seed scheduling is crucial for boosting swarming effects.

## 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 algorithm [13], [14], 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 (*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 $\tau_i$ 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 $r_i[t]/\tau_i[t-1]$:

$$i^*[t] = \max_i \left\{ \frac{r_i[t]}{\tau_i[t-1] + \epsilon} \right\} \tag{2}$$

where $\epsilon$ 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 $\tau_i$ 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 [1]).

### B. History has to be forgotten

When peersets (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 $\tau_i$ brings into the scheduling decision has to decrease over time. For this reason, every time the seed uploads a piece, we update $\tau_i$ using exponential weighted moving average (*EWMA*), namely:

$$\tau_i[t+1] = \beta \cdot I_i[t] + (1-\beta) \cdot \tau_i[t] \tag{3}$$

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

$$I_i[t] = \begin{cases} 1 & \text{if piece } i \text{ was uploaded at round (time) } t \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

From power series analysis [15], we can express the smoothing factor $\beta$ 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 $\Gamma$ 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 are not coming and leaving) then the seed can schedule the same piece $i$ at most $\Gamma$ 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 $i$, this means that 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 fine to upload that piece again. In other words, if a seed has received $\Gamma$ 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.

We will see from our simulations that the value of $\beta$ 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 $\beta$ may only work for certain cases. For typical BitTorrent overlays, where burstiness values are not extremely high [8], 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 a seed in BitTorrent applies *FCFS* on requests coming from four leechers selected in a round robin fashion. Moreover, if the peerset of the seed is static, $\beta \rightarrow 0$, $i.e.$ remembering the whole history, is the best choice.

A more elaborate solution would be to use a control approach to adapt $\beta$. 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 [1].

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 $\Gamma$ 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

```
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 $\tau$ ;
            end
        end
        if collected requests $> M$ then
            $\beta \leftarrow \frac{2}{\Gamma+1}$
            foreach $k$ = 1, ..., $M$ do
                $i_k^* \leftarrow max_i \left\{\frac{r_i}{\tau_i+\epsilon}\right\}$ (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-$\beta$) $\tau_j$ ;
                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 $\tau$ as above every uploaded piece;
    end
end
```

**Algorithm 1**: Seed Scheduling in BUtorrent [1].

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 show analytically that scheduling in the initial phase of a torrent is crucial to boosting swarming effects. Starting from the fluid model in [4], we also show how seed scheduling can be captured during the initial phase by the effectiveness of file sharing ($\eta$), defined in Section III, which our algorithm improves, therefore reducing downloading time.

### A. Expected Number of Wasted Uploads in initial phase

Under our assumptions, we will show that there is, on average, a wasted upload per leecher in the seed neighborhood.

During the initial phase of the torrent, we assume, because of the tit-for-tat mechanism, that leechers do not serve each other, because the probability that they are interested in one another is very low. Since during initial phases, leechers are unlikey to find interesting pieces at other leechers if optimistically unchoked by those leechers, we also ignore optimistic unchockes in this analysis. Of course these assumptions become invalid as the torrent approaches steady state since leechers are more and more likely to be interested in each other.

We now compute the expected number of times a seed uses its upload capacity to upload the same piece in a torrent with $L$ leechers, $p$ pieces and one seed. During the initial phase, almost all pieces are equally rare, so for the $LRF$ algorithm it is just a random selection of pieces. We consider a discrete-time scenario[3], where the time is given by the current scheduling round. If we define a random variable $X_i = \{0, 1\}$, where $X_i = 1$ if and only if a pair of leechers have requested the same piece at round $i$, the expected number of collisions for a given leecher during its download of all $p$ pieces from the seed is given by:

$$E\left[\sum_{i=1}^{p} X_i\right] = \sum_{i=1}^{p} E[X_i] =$$

$$\sum_{i=1}^{p} (0 \cdot P(X_i = 0) + 1 \cdot P(X_i = 1)) = \sum_{i=1}^{p} \frac{1}{p} = p \cdot \frac{1}{p} = 1$$

For simplicity, we assume here that requests are independent and uniformly distributed. This is in general not true, since if a peer got a piece at round $i$, at round $i + 1$, the peer will have one less piece to request. Relaxing this, the number of collisions will be even higher as one can imagine. In our technical report [1] we prove the following:

**Theorem (Collision bounds)**: *Given a peer-to-peer overlay with one seed and L leechers all connected at least to the seed and wishing to download a p-piece file, the expected number of collisions in the first* $p$ *rounds is at least* $\binom{L}{2} - \binom{L}{2}^2 \frac{1}{p}$.

As discussed in Section III, every collision is a potential waste of one slot of seed upload capacity. If the leechers requesting the same piece serve each other, they could have exchanged the piece, maximizing the swarming effect and so minimizing the time to download the whole file.

### B. Effectiveness during initial phases

In this section we adapt the Qiu-Srikant model [4] to capture the initial phase of a torrent, modeling the seed scheduling effect through one of its crucial parameter, the effectiveness of file sharing $\eta$, 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

[2] The choice of $T_{rc}$ is not trivial and it should not be static as overlays are not. We leave the exploration of this parameter for future work. BitTorrent implementation has it set to 10 seconds.

[3] This model of discrete time is important since otherwise the probability of having the same request at the same instant goes to zero.

number of pieces of the served file. This assumption is invalid in the initial phase of a torrent as we show in our technical report [1].

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

$$\begin{cases} \frac{dx}{dt} = \lambda - \theta x(t) - min\{cx(t), \mu(\eta x(t) + y(t))\}, \\ \frac{dy}{dt} = min\{cx(t), \mu(\eta x(t) + y(t))\} - \gamma y(t). \end{cases} \tag{5}$$

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. In particular, the probability that a peer has fewer pieces is higher. For lack of space, we do not show such experiments. [4]

To capture this different behavior in the initial phase, we recompute the effectiveness of a file sharing introduced in [4] as follows:

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

and so for two leechers $i$ and $j$:

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

where:

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

Thus:

$$P = \frac{1}{d^2} \sum_{n_j=0}^{p-1} \sum_{n_i=0}^{n_j} \frac{1}{(n_i+1)^\alpha (n_j+1)^\alpha} \cdot \frac{\binom{p-n_i}{n_j-n_i}}{\binom{p}{n_j}}, \tag{6}$$

where $d = \sum_{i=0}^{p-1}(i+1)^{-\alpha}$ is the normalization constant of the Zipf distribution. Hence we obtain:

$$\eta = 1 - \left( \frac{1}{d^2 p!} \sum_{n_j=0}^{p-1} \frac{n_j!}{(n_j+1)^\alpha} \cdot \sum_{n_i=0}^{n_j} \frac{(p-n_i)!}{(n_i+1)^\alpha (n_j-n_i)!} \right)^k \tag{7}$$

*1) Leechers:* In Figure 2, we present the evolution of the number of leechers ($x(t)$ in Equation 5). Admissible values of $\eta$ depend on the typical values of skewness $\alpha \in [1,4]$, plugged into Equation (7). 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 $\lambda$ and the effectiveness $\eta$ of file sharing.

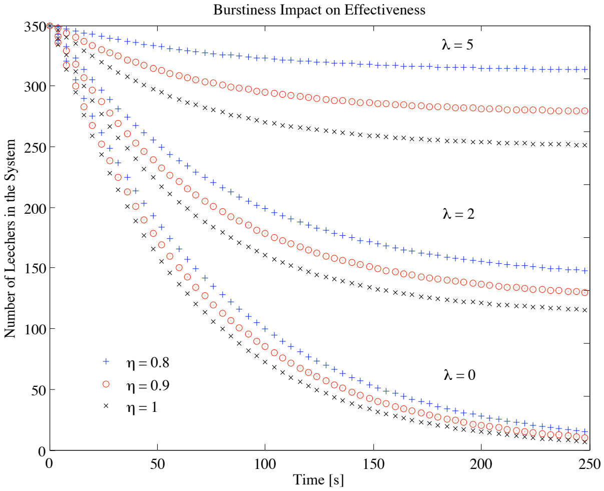

Fig. 2. Number of leechers in the system decreasing faster over time indicates a shorter downloading time. We study here the impact of different rate of arrival $\lambda$ on the effectiveness $\eta$ of file sharing. We notice how for high arrival rate, higher effectiveness has a higher impact on the completion time of the initial 350 leechers.

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 (*i.e.*, higher $\lambda$), even a small improvement in $\eta$ 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. Thus, we expect that $PFS$ scheduling at the seed would be most effective in dynamic overlays.

## VI. Experimental Results

We performed a series of experiments to assess the performance of our Proportional Fair Seed Scheduling, both using the GPS simulator [16], creating our own random physical topologies, and over PlanetLab [17]. Control messages are considered in the simulations as well. Overlay topologies are created by the tracker, which randomly connects a newly arriving peer to 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 [1], starting from the *mainline instrumented client* [18], 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.

[4] In [4] there is also the assumption that, the $n_i$ pieces that peer $i$ has, are chosen randomly from the set of all pieces of the file, *i.e.*, any permutation of size $n_i$ is equally likely to occur. This assumption is still valid in the initial phase, since in in the original BT implementation, (i) pieces equally rare are chosen at random and, (ii) the first few pieces are randomly requested without using LRF.

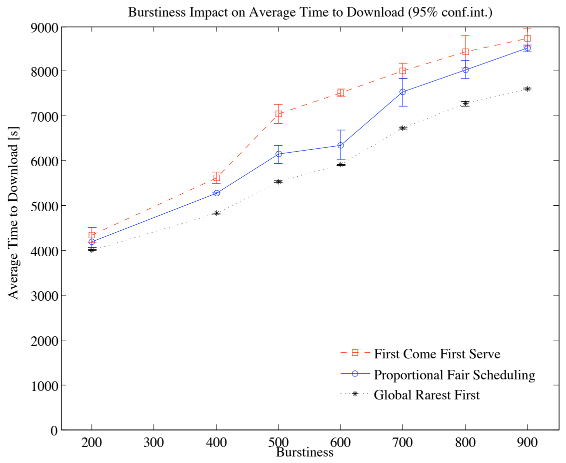

Fig. 3. Average download time for different burstiness values: 350 peers are sharing a 400 MB file. Leechers always depart when they are done with their download. The proportional fair scheduling guarantees up to 25% of improvement over the First Come First Serve approach used by the BitTorrent protocol.

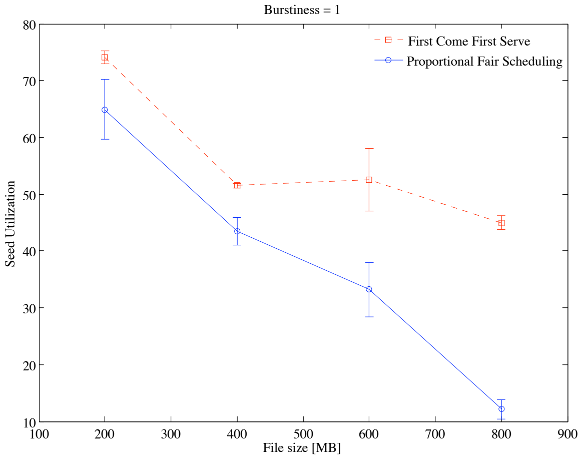

Fig. 4. Seed Utilization for unitary burstiness and 350 peers. Seed applying PFS is less congested due to better piece distribution.

### A. Simulation Experiment 1: Average Downloading Time

Figure 3 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 $\beta$ 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 by up to 25% the leecher average downloading time, depending on the level of burstiness in the arrival process.

To obtain a value of burstiness of 200 for example, we have used a peak rate of 10 where 10 (new) leechers join the overlay in 1 second, every 200 seconds 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 $\beta$, 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 3 when $burstiness = 200$), Section VI-B shows that it is still beneficial to use $PFS$ to reduce seed utilization.

### B. Simulation Experiment 2: Seed Utilization

Figure 4 shows the utilization (defined in Section III) of the only seed present for different file sizes and unitary burstiness value. We point out few observations: (i) with $PFS$, seeds are less congested by leechers' requests, (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 (we do not show the plots of seed utilization over time for lack of space). 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 higher value of burstiness (plots in the technical report [1]) we have noticed smaller improvement of the seed utilization because of smaller peersets and hence fewer requests to the seed.

### C. 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 experiments with 350 PlanetLab nodes sharing a 400 MB file. 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 ($B = 800$).

## VII. Related Work

Swarming techniques have received strong interest as peer-to-peer traffic has become a significant amount of the whole internet traffic. *"Between 50 and 65% of all download traffic is P2P related. Between 75 and 90% of all upload traffic is P2P related"* [19]. The most popular protocol which makes use of swarming is BitTorrent [2].

### A. Improving BitTorrent

Many proposals on how to improve the BitTorrent protocol, by modifying the behavior of leechers, have appeared in the literature. Recent work [20] argues for greedy strategies, BitTyrant [21] is an example. Other work [3] considered game theoretical approaches. Our focus is only on the seeds, without modifying leechers.

Smartseed [5], [14] 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 [22]. Superseed alters the behavior of the main-line BitTorrent seed by masquerading as a normal leecher, while in PFS, a seed "probes" the torrent status to serve the best pieces.

### B. Churn and transient phases

More generally, [23] shows 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 [7], which analyzes starvation in flash-crowd phases. They focus on the end-game mode, where leechers are missing their last pieces. We take a more system-oriented approach that analyzes initial phases.

Measurement studies were also carried out, with focus on BitTorrent transient phases [12]. 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 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.

## VIII. Conclusion

In this work we considered the piece selection aspect 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 but 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 [13], and implemented it into a real file sharing client that we call BUtorrent [1]. 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 by up to $25\%$ the average downloading time over BitTorrent.

## IX. Acknowledgment

We are grateful to Debajyoti Bera, Ilir Capuni and Heiko Röglin for their valuable comments.

## References

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

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

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

[4] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: 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] F. Mathieu and J. Reynier, "Missing piece issue and upload strategies in flashcrowds and p2p-assisted filesharing," in *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, 2006, p. 112.

[8] 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.

[9] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, vol. 393, no. 6684, pp. 440–442, June 1998.

[10] 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 '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, 2006.

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

[12] X. Yang and G. de Veciana, "Service capacity of peer to peer networks," in *Proceedings of 23th IEEE International Conference on Computer Communications (INFOCOM 2004)*, 2004.

[13] H. Kushner and P. Whiting, "Convergence of proportional-fair sharing algorithms under general conditions," *Wireless Communications, IEEE Transactions on*, vol. 3, no. 4, pp. 1250–1259, July 2004.

[14] 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.

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

[16] 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.

[17] 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.

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

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

[20] D. Carra, G. Neglia, and P. Michiardi, "On the impact of greedy strategies in bittorrent networks: the case of bittyrant," in *In Proc. of IEEE P2P, 8th International Conference on Peer-to-Peer Computing, RWTH Aachen University*, SEPTEMBER 8th-11th, 2008.

[21] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*. USENIX, April 2007.

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

[23] 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.