Institut Eurécom
Department of Corporate Communications
2229, route des Crètes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-06-184

Modeling and Analysis of Seed Scheduling Strategies in a BitTorrent Network

Pietro Michiardi, Krishna Ramachandran and Biplab Sikdar

Tel : (+33) 4 93 00 26 26
Fax : (+33) 4 93 00 26 27
Email : {Pietro.Michiardi}@eurecom.fr, {ramak,sikdab}@rpi.edu

[1]Institut Eurécom's research is partially supported by its industrial members: Bouygues Téécom, France Téécom, Hitachi Europe, SFR, Sharp, ST Microelectronics, Swisscom, Texas Instruments, Thales.

Abstract

BitTorrent has gained momentum in recent years as an effective means of distributing digital content in the Internet. Despite the remarkable scalability and efficiency properties that characterize BitTorrent in the long haul, several studies identify the source of the content as the main culprit for the poor performance of the system in a transient regime where user requests for a popular content swamp the source and in case of high node churn. Our work models the scheduling decisions made at the source (called the *seed*) for selecting which pieces of the content to inject in the system through a stochastic optimization process and provides an analytical framework to compare different strategies. We define a new piece selection algorithm (called proportional fair scheduling, PFS) that incorporates the seed's limited vision of the system dynamics in terms of user requests so as to ensure a better content distribution among the users. We prove convergence of PFS and compare its short and long term performance against the mainline BitTorrent implementation and the "smart seed" technique recently introduced in [9]. Our results show that PFS induces substantial improvements on both system performance, by decreasing the download time at the users, and system robustness against peer dynamics, by quickly reacting to sudden changes in the request patterns of the users.

1 Introduction

Peer-to-peer (P2P) networks provide a paradigm shift from the traditional client server model of most networking applications by allowing all users to act as both clients and servers. The primary use of such networks so far has been to swap media files within a local network or over the Internet as a whole. Among current solutions deployed in the Internet, BitTorrent (BT) has received a lot of attention from the research community because of its scalability properties and its ability to handle the so called *flash crowd* scenario, a transient phase characterized by a sudden burst of concurrent requests for a popular content. However, recent results [1–3, 9, 11] have revealed some inefficiencies of BT that translate into a prolonged transient phase, indicating the source of the content (called the *seed*) as the main cause of a disproportionate distribution of the content among the downloaders. In this paper, we motivate the need to incorporate intelligence into scheduling file pieces at the *seed* and develop an analytic framework wherein the impact of the chosen strategies can be studied for a BT-like P2P network. We propose a novel scheduling policy called Proportional Fair Scheduling (PFS) that improves the content distribution process based both on past scheduling decisions and on the actual distribution of content requests as seen by the seed. Using the proposed analytical framework we compare our scheduling policy with the one used in the *mainline* BT implementation and with the best known scheduling improvement called "smart seed" [9]. Through numerical evaluation we show that PFS outperforms previous policies in the short term. For the long term analysis we built a BT simulator and show that our scheduling algorithm achieves a fair content distribution, and reduces the time needed for the seed to inject the content in the system. To summarize, our contributions in the current work can be stated as follows:

- Present an analytic framework wherein different scheduling policies can be modeled and their behavior analyzed.
- Propose a new algorithm, called Proportional Fair Scheduling (PFS) for piece distribution that performs better than the current proposed scheduling modification for the seed.

1.1 BitTorrent overview

Before proceeding further, we provide a brief system overview. BT is a P2P application that replicates the content by leveraging the upload bandwidth of the peers involved in the download process. Each unique content in the system is associated with a $.torrent$ file, and is independent of the remaining torrents in the system. What this implies is that a peer's view of the BT system is confined to a subset, termed the *peer set*, of all the hosts associated with a specific torrent. Peers wishing to download a particular content obtain the corresponding $.torrent$ file from a web server and use a centralized entity called the tracker to collect a random subset of hosts currently active in the torrent. Peers involved in a torrent

cooperate to replicate the file among each other using swarming techniques. BitTorrent achieves scalable and efficient content replication by employing the choke and rarest first algorithms. The former is used for peer selection, i.e. which peer to upload to, while the latter for selecting the file part scheduled to be transfered. Finally, a peer in BitTorrent exists in two states: *seed* state wherein it has the entire content or *leecher* state wherein it is in the process of downloading the file. Note that we have limited our description to details relevant to the current work and have glossed over several technicalities of the BT protocol, which may be found in [7].

The rest of the paper is organized as follows: in Section 2 we survey related literature, while in Section 3 we discuss on the rationale and motivations of our work. In Section 4 we present our analytical model that emulates the various content scheduling strategies for a seed, Section 4.1 provides an analytical dissection and addresses issues such as stability and convergence of the scheduling strategies. We present our results in Section IV and draw relevant inferences from them and finally summarize the work in Section 7.

2 Related Work

In recent times BitTorrent has received substantial interest from the research community, with several modeling as well as simulation studies aiming at improving its performance. Mathematical models for BT are presented in [3–5]. In [4] a fluid model is used to characterize the performance of BitTorrent like networks in terms of the average number of downloads and download times. The authors in [5] propose to improve upon the aforementioned modeling work using a stochastic differential equation approach, by incorporating more realistic BT network behavior in their study. A Markovian model of a BT network was studied in [3], wherein the authors propose a novel peer selection strategy to improve download times. Along similar lines is another modeling work, [10], wherein a branching process based Markovian model was formulated to study BitTorrent like networks.

Simulation based studies are the focus of the works presented in [1, 2, 6, 8, 9]. In [1], the authors investigate the efficacy of the rarest first and the choke algorithms while [2] documents the impact of various system parameters on the networks performance. Along similar lines, [8] presents the dissection of the performance of the mechanisms and algorithms used by BT over a five month period. In [6], the authors make the case for a network coding scheme to improve content replication, while in [9], the authors study the performance of BT by employing metrics such as file download time, link utilization and fairness.

A common feature shared by the literature surveyed thus far is the attempt at modeling the BT system in its entirety. As a result, not all facets pertaining to efficient content distribution are explored. For instance, the first step in this direction is to ensure that the initial seed is able to inject the $entire$ content among the leechers at the earliest and this calls for specialized scheduling algorithms. Unfor-

tunately, with a wholistic approach, this is difficult to accomplish. In this current work we restrict our attention to the $seeds$, and study the impact of scheduling decisions at their end on the effectiveness of content distribution in the system. This is elaborated further in the following section.

3 Rationale and motivation

Typically when content first appears in a BT network, it is stored at a single host, i.e. there is a single seed. From here on, the lifetime of a torrent can be broadly classified into three stages: the initial flash crowd or transient phase where the seed experiences a huge volume of concurrent requests for the content followed by the steady state phase where the system dynamics (especially the arrival of requests for content) are regular and finally the "dying" out phase which marks the point where a substantial portion of the leechers complete downloading the content and leave the system. Note that, it is not binding for one stage to necessarily succeed the other. For, instance a torrent could witness multiple iterations of the flash crowd and steady state phases before eventually dying out.

The motivation for the current work stems from the findings of various simulation studies [2, 9, 11] revealing an inefficiency in the performance of the protocol during the flash crowd phase of a torrent arising from a disproportionate distribution of content among the leechers. It was found that in the flash crowd scenario, often the distribution from the seed becomes a bottleneck in the replication process. In such a scenario, a lack of intelligence during the upload process at the seed could result in some of the pieces not being replicated at all. This phenomenon is termed *starvation* and can adversely impact the torrent's performance in the following manner: consider the scenario where after a certain time (say t), the seed decides to go offline. At such time, if there are certain parts of the file that have not yet been replicated among any of the leechers, then the torrent would eventually die out since none of the leechers would be able to complete the download. Even otherwise, a disproportionate distribution of the parts would result in a prolonged flash crowd scenario since the leechers have nowhere else to request the parts from. In other words the seed and the leechers hosting the rarer parts would be swamped with a huge volume of upload requests. This problem if further magnified if the seed is bandwidth constrained. Thus, an improved distribution of content at the seed's end would serve to improve the performance of the torrent by decreasing the download time of the leechers, since there is a bigger pool of leechers with the same piece.

A relevant doubt at this stage would be to question the rationale behind distinguishing between scheduling decisions at a seed and those at a leecher. In other words, *why would not a common scheduling algorithm work for both ?* The answer to this lies in the difference between the view of the torrent as seen by a leecher and a seed. While the leecher has complete information on the part distribution among the peers in it's peer set, this knowledge is hidden from the seeds. This is primarily

due to a mechanism used to reduce the control message overhead named the HAVE suppression technique. HAVE messages are used to disseminate information on the piece distribution among leechers: each time a leecher finishes to download a piece, she will inform all peers in her peer set about the new piece availability. The HAVE suppression technique inhibits the transmission of HAVE messages to those peers that currently have a replica of the announced piece. The consequence is that seeds will have no information on the piece distribution in her peer set. In fact, in the current *mainline* implementation of the BT protocol, a seed simply replies to piece request originated at the leechers without any scheduling decision (hence the name random scheduling (RS) used hereafter). Thus, lack of a global snapshot constrains a seed to base scheduling decisions on it's own past history in order to improve content distribution and hence the motivation behind the current work.

The endeavor in the current work is arrive at a mathematical framework generic in nature so as to facilitate the performance quantification of various scheduling strategies that could be implemented at the seed. In this paper we try and address the following problem: *How best can a seed incorporate the limited view of the BT system into its scheduling decisions so as to ensure better content distribution among the downloaders?*

To this end, as a part of their simulation study of BT, the authors in [9] propose the local rarest first (LRF) policy, termed "smart seed" scheduling policy, as an improvement over the current scheduling scheme. However, the proposed scheme is not receptive to the system dynamics, i.e. leechers entering and leaving the torrent, and further, the optimality of such a strategy is not guaranteed. In this paper, we provide a theoretical grounding for the problem through a framework based on stochastic approximation algorithms. In particular, we compare the performance of our scheduling strategy, the proportional fairness scheme (PFS), with the current $proposed$ modification, local rarest first (LRF), and the $existing$ policy, random scheduling (RS), currently used in the *mainline* BT client.

4 Analytical Framework

In this section, we present our analytic framework based on stochastic approximation to study the performance of piece scheduling decisions made at the seed. While the framework is generic in nature and applicable to study a large class of scheduling policies, for illustrative purposes we focus our discussion on characterizing the proportional fairness (PFS) and the LRF schemes. In the current section we present a detailed overview of incorporating the PFS scheme into the framework while in Section 4.2 we outline the modeling of the LRF scheme. The gist of the two schemes is presented below:

- LRF: In this policy users are served on a first come first serve basis. Leechers request the seed for a *set* of parts (RB) and the seed uploads the least served piece amongst RB.

- Proportional Fairness Scheme (PFS): In this scheme, the seed takes into account the requests coming in for each part and the corresponding past throughput and uploads the piece with the maximum ratio of the two.

Note that the *existing* scheduling algorithm (RS) is purely random in nature hence we do not model it in the current work.

Before proceeding with the description of the model, we outline our assumptions: The content to be replicated is divided into p equal parts and is stored at a $single$ seed. The seed is modeled by a single server queue with no buffer space. Time is slotted in intervals with the granularity of each round chosen to accommodate the transfer of a single file part. For the sake of simplicity, in the current work we allow peers to upload to 1 other randomly selected peer, as opposed to the fully fledged implementation wherein 4 peers are selected using the choke algorithm. In particular, the seed serves only one part in a round, with the decision on the piece to be uploaded in the next round made based on the requests that arrive during the *current* time slot. The peer satisfying the scheduling criteria is served in the next slot while the rest of the requests are dropped. The above assumptions are a reasonable mapping to a bandwidth constrained seed where it makes sense to dedicate the entire bandwidth to serve a particular request instead of increasing the latency by dividing it.

Let the request vector at the end of slot n (start of slot $n+1$) be represented as $\mathcal{R}(n+1) = [r_{1,n+1}, r_{2,n+1}, \cdots, r_{p,n+1}]$, where $r_{i,n+1}$ denotes the number of times part i was requested for in round n. In other words, each entry in $\mathcal{R}(n+1)$ represents the number of leechers requesting for that particular part during the previous round, i.e. round n. Let the throughput vector be denoted as $\mathcal{T}(n) = [t_{1,n}, t_{2,n}, \cdots, t_{p,n}]$, where $t_{i,n}$ represents the number of times part i was served in n rounds. Similarly, let $\theta(n) = [\theta_{1,n}, \theta_{2,n}, \cdots, \theta_{p,n}]$ denote the vector of sum of requests for the different parts, each time it was served, averaged over the past n rounds. The average throughput and request rate for part i after n rounds are defined as follows:

$$\mathcal{T}_{i,n} = \frac{\sum_{k=1}^{n} \mathbb{I}_{i,k}}{n} \qquad \theta_{i,n} = \frac{\sum_{k=1}^{n} r_{i,k}\mathbb{I}_{i,k}}{n}$$

where $\mathbb{I}_{i,k}$ is an indicator variable equal to 1 if part i is scheduled in round k and 0 otherwise. Thus, at the end of each round, each entry in vectors θ and $\mathcal{T}$ can be updated as follows:

$$\theta_{i,n+1} = \theta_{i,n} + \epsilon_n[\mathbb{I}_{i,n+1}r_{i,n+1} - \theta_{i,n}] \quad (1)$$

$$\mathcal{T}_{i,n+1} = \mathcal{T}_{i,n} + \epsilon_n[\mathbb{I}_{i,n+1} - \mathcal{T}_{i,n}] \quad (2)$$

with $\mathbb{I}_{i,n+1}$ as explained above and $\epsilon_n = \frac{1}{n+1}$. Given the above system parameters, the seed scheduling algorithm we propose (PFS) can be summarized as follows:

- Among the non-zero request entries that arrive in a round, select that part maximizing the following ratio:

$$\arg\max_{i} \left\{ \frac{r_{i,n+1}}{\theta_{i,n} + d} \right\} \tag{3}$$

  If there are multiple parts satisfying the above criterion, break ties arbitrarily. Here, d is a constant arbitrarily close to zero and is chosen to avoid the divide by zero error in the initial stages of the torrent when the throughputs for nearly all the parts are close to or equal to zero.

- Upload the chosen part from the previous step to the requesting peer. Again, break ties arbitrarily

It is quite natural to question the soundness, be it theoretical or practical, of a formulation as in Equation (3). The proposed format can be justified if the content replication process were to be viewed, from a seed's perspective, as a variant of the utility maximization problem. Note that in a BT system, the onus is primarily on the seed to ensure the spread of content among the peers in the system. Thus, a seed seeks to maximize the replicas of each piece among the leechers and therefore it is reasonable to assume that the utility function chosen is concave in nature. In this context consider the utility function to be the sum of the logarithm of average number of requests of the individual pieces, i.e.

$$U(\theta) = \sum_{i=1}^{p} \log(\theta_i + d) \tag{4}$$

Then it can be shown [13] that for this particular choice of utility maximization, the policy outlined in Equation (3) yields optimal results. We further note that the seed is not constrained to choose the policy of Equation (3). Any reasonable representative concave function can be chosen as the utility function and the scheduling policy appropriately tailored to obtain optimal results.

4.1 Convergence Analysis

The formulation of Equations (1) and (2) is in the framework of stochastic approximation algorithms [12]. Notably, under certain assumptions, which can be shown to be valid in a BitTorrent scenario, it can be shown that the stochastic approximation algorithm in Equation (2) can be described by a $deterministic$ mean field ordinary differential equation (ODE) system. This enables us to characterize the behavior of the proposed algorithm and is also a useful tool to study the asymptotic properties such as the long term throughput of the respective file pieces. An important consequence of the convergence proof is that concerning the stability of the system. For example, a scheduling policy that converges asymptotically also

characterizes a stable system. We now outline the assumptions required for the ODE convergence:

- Stationarity of the request distribution: $\{\mathcal{R}(n), n < \infty\}$. Note that in a BT system, the requests generated by leechers for the missing pieces depend only on the current distribution of the parts among each other. For instance, if a system snapshot at time t were to be translated to a different instant, say t_1, the pattern of requests generated would be similar. Define the stationary expectation w.r.t. the request distribution for part i as

$$\hat{h}_i(\theta) = E[\mathbb{I}_{\{\frac{r_i}{\theta_i+d_i} \geq \frac{r_j}{\theta_j+d_j}\},\forall j \neq i}] \tag{5}$$

- Lipschitz continuity of $\hat{h}_i(.), 1 \leq i \leq p$. We demonstrate this with the help of a simple case where the file consists of two parts and the joint probability density is given by $p(r_1, r_2)$. Then, for part 1, Equation (5) can then be simplified as

$$\hat{h}_1(\theta) = \int \mathbb{I}_{\{\frac{r_1}{r_2} \geq w\}} p(r_1, r_2) dr_1 dr_2 \tag{6}$$

  where $w = (\theta_1 + d)/(\theta_2 + d)$. Note that in the above equation we have used a continuous density function for the request generation process, which is in fact discrete. This is because, it has been shown in [14], that the requests for the parts can be approximated by a Gaussian distribution which is continuous. In the current work, we employ the same approximation and hence the formulation of Equation (6). Now, Eqn. (6) is Lipschitz continuous with respect to w, since the area of the region where the indicator function is not zero is a differentiable function of w [13]. Similar is the case for $\hat{h}_2(\theta)$. Further, the derivatives of $\hat{h}_1(\theta)$ and $\hat{h}_2(\theta)$ will be continuous if $p(r_1, r_2)$ is bounded and continuous.

- Bounded density of $\mathcal{R}(n)$. This is trivially satisfied since the number of users in a BT system is finite thus ensuring that the requests generated during each round of time remain bounded.

Under the above assumptions, the stochastic approximation algorithm of Equation (2) can be approximated by the ODE given by:

$$\dot{\mathcal{T}}_i^{PFS} = E[\mathbb{I}_{\{\frac{r_i}{\theta_i+d_i} \geq \frac{r_j}{\theta_j+d_j}\}\forall j \neq i}] - \mathcal{T}_i^{PFS} \tag{7}$$

4.2 Modeling other policies

The analytic framework provides a generic setting wherein a wide class of scheduling policies can be modeled and quantified. We illustrate the robustness of the framework by modeling the LRF scheme in [9] as follows:

- For each piece i in the request block (RB) set $r_{i,n+1} = 1$
- Choose piece i such that: arg $\max_{i \in RB} \left\{ \frac{1}{\theta_{i,n}+d_i} \right\}$; break ties arbitrarily
- Upload the piece from the previous step

The corresponding throughput formulation for part i, $\mathcal{T}_i^{LRF}$, is then given by:

$$\mathcal{T}_{i,n+1}^{LRF} = \mathcal{T}_{i,n}^{LRF} + \epsilon_n [\mathbb{I}_{\{\frac{1}{\theta_i+d_i} \geq \frac{1}{\theta_j+d_j}\} \forall j \neq i} - \mathcal{T}_{i,n}^{LRF}] \tag{8}$$

and the equivalent ODE by:

$$\dot{\mathcal{T}}_i^{LRF} = E[\mathbb{I}_{\{\frac{1}{\theta_i+d_i} \geq \frac{1}{\theta_j+d_j}\} \forall j \neq i}] - \mathcal{T}_i^{LRF} \tag{9}$$

5 Implementation details

In this section we provide some more details on the BT protocol and discuss on practical implementation issues that may arise when implementing the PFS algorithm in a real BT client.

The protocol that governs the piece exchange between peers in BitTorrent can be trivially described as follows:

- Any peer wishing to download a part of the file unicast a control message called `INTERESTED` message, to announce the willingness to download a part from a remote peer;
- A remote peer schedules one or more upload opportunities (based on the peer selection algorithm that we will not detail in this paper) and informs the selected peer through a control message called `UNCHOKE`;
- The unchoked peer selects a piece to download (based on the piece selection algorithm) and unicasts a `REQUEST` message to the remote peer;
- Finally, the remote peer uploads the part to the requesting peer.

Peer scheduling decisions, *i.e.* `UNCHOKE` messages are sent, are made every 10 seconds with one exception, as described for example in [1].

The alternative scheduling policy for the seed proposed in this paper can be incorporated into present BT clients with the minimal of changes and incurring minimal overhead. Note that leechers in a BT system can distinguish between seeds and non-seeds, i.e. a leecher knows the seeds in it's peer set through the initial handshake procedure wherein BT clients exchange a digest of their piece availability the first time a connection is established. Thus, when an `INTERESTED` message is sent to a seed, the leecher appends the piece identifier it is looking for, in a similar way it is done for a regular `REQUEST` message. Once the seed has collected

a sufficient number of requests in a round, it executes the above algorithm and unchokes the leecher that satisfies both the piece scheduling and the peer scheduling criterion. Note that the change is made *only* to the `INTERESTED` messages sent to the seed, the format of other `INTERESTED` messages (sent to leechers) remaining unaltered. Thus, the size of each `INTERESTED` message to a seed increases by a byte and while the message complexity remains invariant, the byte overhead increases, albeit minimally.

6 Results

In this section we present results comparing the efficiency of the PFS scheme against LRF. To prove the robustness of the proposed framework, we quantify the performance gains obtained in the short term as well as in the long run. For the short term analysis we perform a numerical evaluation of the PFS scheduling using the stochastic approximation algorithm as described in Section 4. On the other hand, we perform the long term evaluation using a custom simulator of the BT system. The rationale behind this choice lies in the lack of a *realistic* characterization of the piece request rate $\mathcal{R}(n) = [r_{1,n}, r_{2,n}, \cdots, r_{p,n}]$ to be used in the analytical evaluation presented in Section 4.1. Our implementation, which is outlined in Section 6.2, also provides a global perspective of the system, as opposed to the seed's perspective offered by the analytical model.

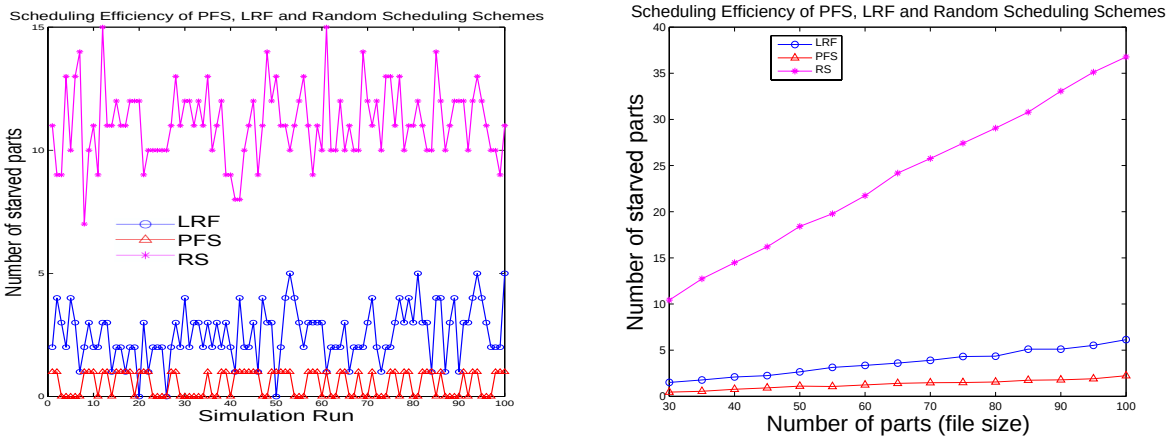
6.1 Short term behavior

Since the primary objective in the initial stages of a torrent is to minimize starvation of pieces, a natural benchmark for comparing the policies would be to measure the number of starved pieces at a certain point of time under each policy. Here, we choose to make the comparison after p rounds, where p denotes the number of pieces the content is divided into. The rationale behind this is as follows: since we assume that the seed schedules one piece per round, in the ideal case it would require p rounds to ensure that the file in it's entirety is present among the leechers. Figure 1 graphs the performance of the various policies in the flash crowd stage. In Figure 1(a), the number of starved parts of a 30 part file are plotted for each policy over 100 runs of our algorithm while Figure 1(b) quantifies the impact of the file size on the number of starved parts. Each point on the graph of Fig. 1(b) is an average of 100 runs. As seen from the plots, the proportional fair scheme offers significant gains over the other two policies. Even with increasing file sizes, the performance degradation is not very substantial. In fact, for a file consisting of 100 parts, the ratio of starved pieces in the "flash crowd" phase is about 1:3 for PFS and LRF, while it is around 1:18 when comparing PFS and the RS schemes. We believe the better performance of the algorithm could be attributed to the following factors:

- The seed makes a scheduling decision taking into account all the requests

that are made in a particular round, unlike LRF and RS where users are served in a first come first served manner. For instance, if a large number of leechers request for a particular piece there is a higher probability of it being a rare piece as compared the rarity of a piece requested by a single user.

- In an open BT system the local rarest piece need not reflect reality, from the seed's perspective, due to leechers entering and leaving the system. Thus, when a seed bases its scheduling decisions only on its past history like in the LRF case, due to peers' dynamics a seed may have a stale vision of what is rare and what is not in the system. The PFS scheme accounts for this by using the number of requests for a piece as the system's indicator of rarity and makes the scheduling decision accordingly.



(a) (b)

Figure 1: Performance evaluation in the flash crowd phase

6.2 Long term behavior

As a final validation of our theoretical formulation presented in Eqn (3), we present a simulation comparison of the proposed PFS algorithm against the LRF scheme, especially the behavior over long time periods. Since we only modify the seed scheduling algorithm, it only makes sense to quantify the impact within the seed's peer set and not globally. The main objective in the long term is to prevent a high variance in the number of replicas of each part, i.e. prevent a disproportionate piece replication in the peer set since it is the root cause of all problems. In other words the scheduling process should be "fair" to the individual pieces. The intuition behind this is that ensuring a balanced replication of the pieces can help improve download times since there is a higher level of redundancy and also distribute the load more evenly among the leechers. As a measure of the degree of

fairness, we employ the Max-Min Fairness Index [15] given by $\frac{min_{\forall i}(x_i)}{max_{\forall i}(x_i)}$, where x_i denotes the number of replicas of part i at the end of a round *in the seed's peer set*. Before discussing the long term results, we provide a brief description of the custom simulator we designed.

6.2.1 BitTorrent Simulator

We developed a synchronous simulator working in rounds wherein we implemented both seed and leecher algorithms following the BT specification. We then implemented two scheduling policies at the seed side, the PFS and the LRF. The only limitation we imposed on the simulator follows the one of the analytical model: only one peer is unchoked in each round. The peer set size for a peer is set to the default value of the mainline BT client, that is 80 peers. To quantify the impact of the scheduling decisions, we assume that leechers that finish downloading leave the torrent, i.e. there is a single seed in the system at all times.

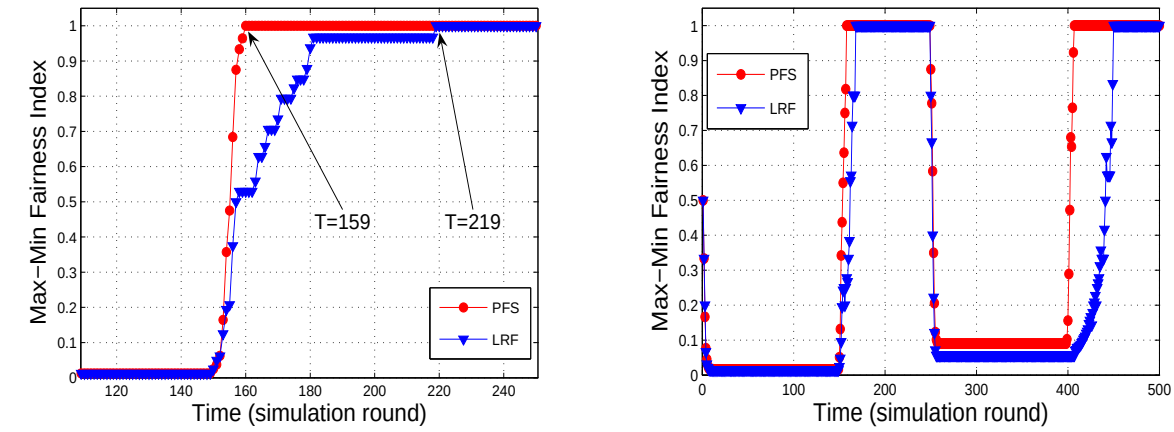
It is worthwhile noticing from the discussion in Section 5 that, as compared to the LRF scheduling policy which requires modifying both the seed and the leecher side of BT as well part of the protocol specification, PFS scheduling can be seamlessly integrated with a simple modification at the seed side only.

6.2.2 Simulation results

We compare the LRF and the PFS scheduling algorithms assuming the content to be split in $p = 150$ pieces. We simulate the presence of one seed only in the system and study two representative and realistic scenarios: the first where the torrent experiences a heavy flash crowd and the second indicative of a torrent with a high churn rate.

To simulate the flash crowd setting, 160 peers are injected into the system in the first round, after which no further joins are allowed. The objective here is to study the algorithm's sensitivity toward achieving a balanced replication in the wake of a huge volume of requests. Note that the Max-Min Fairness plots can also be used to infer and compare the download times experienced by the leechers. Since we assume that leechers with the entire content depart, the time T when the graph reaches one also denotes the instant when all the leechers in the system have finished downloading. Therefore, the faster the graph peaks to one, the better it is in terms of fairness as well as download times. In Figure 2(a) we plot the Max-Min Fairness index versus time (in simulation rounds) for the flash crowd scenario described above. When using PFS scheduling, $T = 159$ while for the LRF case $T = 219$. A similar trend was observed over multiple repetitions of the experiment, showing an improvement of the total time to download the content in favor of PFS whereas this improvement was even more pronounced in the case of smaller files. Further, as shown in Figure (3), the time required for all the 160 nodes in the flash crowd scenario to finish downloading the content grows linearly with the increase in the file size for PFS, while for LRF the behavior was quite erratic with a high

variance in the download times.



(a) (b)

Figure 2: Simulation results for the long term analysis.

In the second simulation study, we focus on the responsiveness of scheduling decisions at the seed when substantial variations in the population of peers downloading the content arise, i.e. a system with high churn. In particular, we consider 80 peers joining the system at round 1, then 30 randomly chosen peers leaving the system at round 150, and finally 80 new peers joining the system at round 250. Although both PFS and LRF scheduling reach the highest fairness index, Figure 2(b) clearly shows that PFS reacts consistently faster to peer dynamics as compared to LRF. Similar results (not reported due to lack of space) have been obtained for different runs of the same scenario.

7 Conclusion and Future Work

In this work, we motivated the need for improved scheduling algorithms at the seed in a BT system and quantified the performance gains obtained thus. A generic analytical framework to model such algorithms was presented and a novel seed scheduling strategy to achieve better content replication was proposed. Through numerical evaluation of the model as well as simulations the improved performance of the proposed PFS algorithm over existing strategies in the literature (LRF and the existing mainline random scheduling schemes) was demonstrated. As a natural extension to this paper we will assess the impact of PFS on a real BitTorrent network through measurements using modified clients deployed on Planet Lab.

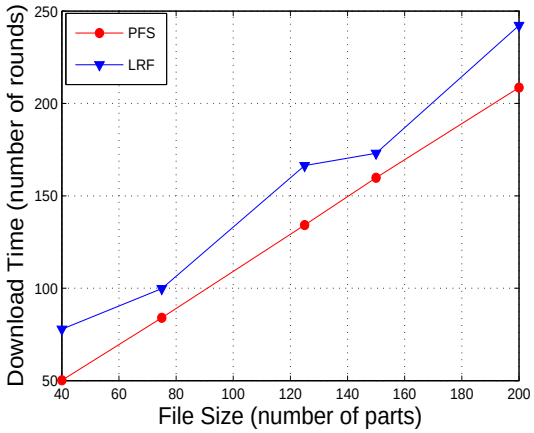


Figure 3: Impact of file size on performance of PFS and LRF.

References

[1] A. Legout, G. Urvoy-Keller and P. Michiardi, *Rarest First and Choke Algorithms Are Enough*, ACM SIGCOMM/USENIX IMC 2006, Rio de Janeiro, Brazil.

[2] G. Urvoy-Keller and P. Michiardi, *Impact of Inner Parameters and Overlay Structure on the Performance of BitTorrent*, IEEE Global Internet Symposium 2006, Barcelona, Spain.

[3] Y. Tian, D. Wu and K. W. Ng, *Modeling, Analysis and Improvement for BitTorrent-Like File Sharing Networks*, IEEE INFOCOM 2006, Barcelona, Spain.

[4] D. Qiu and R. Srikant, *Modeling and performance analysis of BitTorrentlike peer-to-peer networks*, ACM SIGCOMM 2004, Portland, OR, USA.

[5] B. Fan, D-M. Chiu and J. C. Sl Lui, *Stochastic Differential Equation Approach to Model BitTorrent-like P2P Systems*, IEEE ICC 2006, Istanbul, Turkey.

[6] C. Gkantsidis and P. Rodriguez, *Network Coding for Large Scale Content Distribution*, IEEE INFOCOM 2005, Miami, USA.

[7] B. Cohen, *Incentives Build Robustness in BitTorrent*, Workshop on Economics of Peer-to-Peer Systems 2003, Berkeley, USA.

[8] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra and L. Garces-Erise, *Dissecting BitTorrent: Five Months in a Torrent's Lifetime*, PAM 2004, Antibes, France.

[9] A. Bharambe, C. Herley and V. N. Padmanabhan, *Analyzing and Improving a BitTorrent Network's Performance Mechanisms*, IEEE INFOCOM 2006, Barcelona, Spain.

[10] X. Yang and G. de Veciana, *Service capacity in peer-to-peer networks*, IEEE INFOCOM 2004, Hong Kong, China.

[11] F. Mathieu and J. Reynier, *Missing Piece Issue and Upload Strategies in Flashcrowds and P2P-assisted Filesharing*, Technical Report, ENS, France.

[12] H. J Kushner and G. Yin, *Stochastic Approximation Algorithms and Applications*, 2nd ed. Berlin, Germany: Springer-Verlag, 2003.

[13] H. J Kushner and P. A Whiting, *Convergence of Proportional-Fair Sharing Algorithms Under General Conditions*, IEEE Transactions on Wireless Communications, Vol. 3, No. 4, July 2004

[14] D. Erman, D. Ilie and A. Popescu, *BitTorrent Session and Message Models*, ICCGI 2006, Bucharest, Romania.

[15] B. Radunović and J.Y. Le Boudec, *A Unified Framework for Max-Min and Min-Max Fairness with Applications*, Technical Report, EPFL, July 2002