On P2P systems for enterprise content delivery *

Antonio A. de Aragão Rocha[1], Daniel Sadoc Menasche[2], Don Towsley[2], Arun Venkataramani[2]

[1] PESC/COPPE – Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

[2]Department of Computer Science – University of Massachusetts, Amherst, MA, USA

arocha@land.ufrj.br, {sadoc, towsley, arun}@cs.umass.edu

Abstract. *In this paper we address the problem of optimizing p2p systems for enterprise bandwidth cost savings. Consider a scenario where an enterprise uses p2p swarming to deliver content to its clients. A content may be a file or a package (bundle) of files. For popular and large content, our work finds that enterprise servers can exclusively rely on the cooperation among the peers in the form of a swarm to disseminate the files. We introduce the idea of self sustaining swarms, i.e., swarms where the presence of the server is not necessary for the content to be efficiently transmitted. We characterize, under different scenarios, the concept of self sustainability. We then focus on swarms that are not always self sustaining, and propose a scheme for the server to decide how much bandwidth to allocate to a swarm as a function of the arrival and departure rate of the peers. To show the applicability of our controller, we deploy BitTorrent clients in PlanetLab and in a local cluster. Our emulations confirm that the controller leads to significant improvements in terms of cost savings without degrading the download time for the clients. Finally, we analyze the case of an enterprise providing content through multiple swarms, and present an optimization framework that can be used to determine how much bandwidth should be allocated for each of the swarms.*

1. Introduction

An analysis of the sales trends in today's markets indicates that there is a rising demand for obscure titles. For instance, [Anderson 2006] points out that, in 2004, products not available in traditional stores were responsible for 22%, 57% and 20% of the sales of Rhapsody, Amazon.com and Netflix, respectively. In November of 2008, [Page 2008] reported that 20% of the revenue of Rhapsody came from songs that are not on the top 52,000. This reduction of the blockbuster effect indicates that in the area of content delivery enterprises that can "make everything available", at a low cost, can have a significant advantage. In this context, commercial P2P systems are the natural solution for enterprises that want to deliver a huge amount of content at a low price.

*This work was supported in part by the National Science Foundation under award numbers CNS-0519922 and CNS-0721779. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation. Research of Antonio Rocha and Daniel Menasche also funded, respectively, in part by a scholarship from CAPES and CAPES/Fulbright (Brazil).

Commercial P2P systems (such as BitTorrent, Inc. [Cohen 2003] and Kontiki [Kontiki 2008]) have a number of advantages over their client-server counterparts: (i) *bandwidth cost savings*: P2P systems rely on the idle capacity of clients to boost system performance and guarantee fast delivery of the content to all the interested users; (ii) *scalability*: even if the content demand rapidly changes over time, p2p systems guarantee scalability since system capacity increases with the demand; (iii) *serve content behind firewalls*: despite the fact that many users are behind firewalls, if one copy of the file reaches a protected network all of the users in that network can access the file.

For a content delivery enterprise to thrive in a market where unpopular content plays a key role on its budget it is imperative to have adequate resource sharing mechanisms so that popular content is disseminated at (almost) zero cost by taking advantage of cooperation among the clients, while still spending as few resources as possible to serve unpopular content. One solution to attain this goal consists of coupling a strategic publishing mechanism to the existing BitTorrent system, as we describe next.

1.1. The BitTorrent System

BitTorrent [Cohen 2003] uses P2P file swarming to disseminate content. A swarm is formed by a set of users interested in downloading a given file (or set of files, packaged in a bundle). The file is chopped into blocks that peers download from the server and from each other. A tracker is used to coordinate the interaction among the peers. Periodically, a peer queries the tracker and obtains a random subset of other peers in the swarm. A leecher is a peer that hasn't finished its download yet. After concluding the download, the leecher becomes a seed. Peers that have incentives to make the content available are referred to as publishers. The very first peer that makes available a given content is the original publisher.

A swarm may be in one of the following three regimes: (1) unpopular, when the number of peers requesting the content is so small that peers must download it from the server; (2) critical, when peers occasionally use the server to download blocks, but most of the time rely on each other to get the desired content and (3) self-sustaining, when even if the server is turned off, the clients can still successfully complete their downloads.

1.2. Contributions

In this paper, we are interested in answering the following questions: (1) how to precisely characterize that a swarm does not need the support of the server, i.e., how to classify a swarm as being self sustaining? (2) in the critical and unpopular regimes, how much capacity should the server devote to a swarm in order to guarantee a high level of service to the clients while still not incurring in unnecessary costs? (3) for an enterprise supporting multiple self sustaining swarms, how should it split its bandwidth across these swarms?

We make the following contributions.

1. We propose a model for content availability in BitTorrent. Using this model, we characterize self-sustainability under different scenarios;
2. focusing on swarms in the critical and unpopular regimes, we propose a controller that determines how much bandwidth should be devoted to a swarm as a function of the arrival and departure rates of clients to that swarm. We use controlled experiments to validate the applicability of our controller;

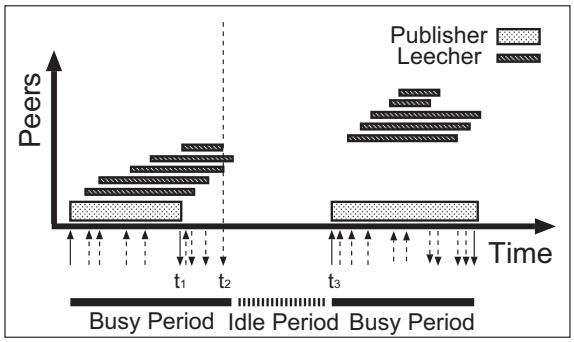


Figure 1. Busy and idle periods.

3. we analyze the case of an enterprise that provides multiple content and present an optimization framework that can be used to decide how much bandwidth should be allocated for each of the swarms.

2. A Model for Self Sustainability

2.1. Model Overview

Figure 1 illustrates the dynamics of content availability in BitTorrent. Each swarm passes through busy and idle periods. The beginning of the first busy period is characterized by the arrival of a publisher. After the publisher departs (time t_1) the content eventually becomes unavailable. When the content becomes unavailable (time t_2) the idle period begins. The return of a publisher (time t_3) ends this idle period and characterizes the beginning of a new busy period.

Our goal is to understand how the provider can strategically decide when to go on-line or off-line as a function of 1) the popularity of the swarm, measured in terms of the arrival rate of the peers λ; 2) the mean time for which peers stay in the system, s/μ, where s is the size of the file and μ the mean download rate of peers (Table 1).

2.2. Background

We define the *availability* of the content as the fraction of time that the system is in a busy period. As a first approximation, we assume that the busy period only vanishes when the number of peers goes to zero. In § 2.4 we relax this last assumption. In order to obtain the results, we model the system as an $M/G/\infty$ queue.

Our results for availability will rely on those reported by Browne and Steele [Browne and J.M.Steele 1993]. Let customers arrive according to a Poisson process with parameter λ and have their service time distributed according to an exponential random variable with mean θ. The mean busy period in this standard case is given by:

$$E[B] = \frac{1}{\lambda}(e^{\lambda\theta} - 1) \tag{1}$$

If we allow customers initiating a busy period to draw their service time from an exponential distribution with mean θ while all other customers draw their service times from an exponential distribution with mean α, the expected busy period is given by:

$$E[B] = \theta + \alpha\theta \sum_{i=1}^{\infty} \frac{(\lambda\alpha)^i}{i!(\alpha + i\theta)} \tag{2}$$

variable	description
λ	arrival rate of peers (peers/s)
s	size of file (bytes)
r	arrival rate of publishers (peers/s)
u	mean uptime of server (s)
s/μ	mean time to download a file (s)

Table 1. Notation.

In its full generality, the Browne and Steele model [Browne and J.M.Steele 1993] allows for customers initiating a busy period to draw their service time from an arbitrary distribution with Laplace transform $h(s)$ and mean θ while all other customers draw their service times from a distribution with LT $g(s)$ and mean α. $E[B]$ in this case is:

$$E[B] = \theta + \sum_{i=1}^{\infty} \frac{(\lambda\alpha)^i \alpha[1 - h(i/\alpha)]}{i!i} \tag{3}$$

If N denotes the number of peers served during a busy period, then $E[N] = \lambda E[B]$.

2.3. Self-Sustainability Characterization

Our goal is to characterize the *self-sustainability* of a swarm under different scenarios. First, we consider the case where there is a single publisher that departs and never returns ($r \rightarrow 0$). After that we consider the case where publishers arrive with aggregate rate r. Finally, we study the situation in which content may become unavailable even if the population size does not vanish to zero.

2.3.1. A Single Publisher That Never Returns

We assume that the first publisher for content k stays online for a mean time $u = \kappa s/\mu$ after which it goes off-line and never returns. Peers complete the download only if they arrive during the first busy period.

Denoting by B the busy period duration, from (2) we have:

$$E[B] = u\Big[1 + (s/\mu)\sum_{i=1}^{\infty} \frac{(\lambda(s/\mu))^i}{i!((s/\mu) + iu)}\Big] \tag{4}$$

If the publisher wants the busy period to last for an average of $E[B] = M$, the following theorem provides the time u^* that it must stay on:

Theorem 2.1 *$u^* = \kappa^* s/\mu$ where κ^* is given by the positive root x of*

Hypergeom$([1,(1+x)/x],[2,(2x+1)/x],\lambda s/\mu)xs^2\lambda + \mu xs + \mu x^2 s - \mu^2 M - M\mu^2 x = 0$

which, under the assumption that $(1+x)/x \approx 1$ and $(2x+1)/x \approx 2$, is given by

$$\frac{-As^2\lambda - s\mu + \mu^2 M + \sqrt{A^2 s^4\lambda^2 + 2As^3\lambda\mu - 2As^2\lambda\mu^2 M + s^2\mu^2 + 2s\mu^3 M + \mu^4 M^2}}{2s\mu}$$

where $A = Hypergeom([1,1],[2,2],\lambda s/\mu) = \int_0^1 \int_0^1 e^{xy\lambda s/\mu} dxdy$. *In general,* $Hypergeom(n,d,z) = \sum_{k=0}^{\infty} \frac{C_{n,k}}{C_{d,k}} \frac{z^k}{k!}$ *where* $C_{v,k} = \prod_{j=1}^{|v|} \Gamma(v(j)+k)/\Gamma(v(j))$ *and* Γ *is the standard gamma function.*

Proof: Follows from equating (4) to M and solving for u. □

2.3.2. Intermittent Publishers with Impatient Peers

We now consider the scenario where there is a population of publishers (e.g., machines in a cluster) that arrive at rate r and stay in the system for a mean time u.

We assume that peers that arrive during an idle period leave immediately. The probability, P, that a request leaves the system without being served is given by:

$$P = \frac{1/r}{1/r + E[B]} \tag{5}$$

Recall that the average residence time of publishers and leechers is u and s/μ, respectively. We approximate the residence time of an arbitrary peer, a mixture of two exponentials, by an exponential random variable with mean $1/\mu' = \frac{\lambda}{\lambda+r} s/\mu + \frac{r}{\lambda+r} u$.

We are interested in determining the minimum arrival rate, r^*, so that P lies below a threshold P_t:

Theorem 2.2 $r^* \leq (1-P_t)/(P_t E[B])$ *where*

$$E[B] = u\Big[1 + 1/\mu' \sum_{i=1}^{\infty} \frac{((\lambda+r)1/\mu')^i}{i!(1/\mu' + iu)}\Big]$$

Proof: The result follows from equating (5) to P_t and solving for r. The equation for $E[B]$ follows from (2). We have an inequality (rather than an equality) for r^* since the increase of $E[B]$ due to the increase of r is not taken into account when inverting (5). □

2.3.3. Intermittent Publishers with Patient Peers

We now assume that peers arriving during an idle period wait for a publisher to become available. As in the previous section, we consider a population of publishers (e.g., machines in a cluster) that arrive at rate r and stay in the system for a mean time u.

It can be shown that the average download time of a file, $E[T]$, is given by:

$$E[T] = s/\mu + \frac{1/r}{1/r + E[B]} \frac{1}{2}\Big(\frac{1}{r} + \frac{1}{r+\lambda}\Big) \tag{6}$$

We are interested in determining the minimum arrival rate, r^*, so that $E[T]$ is below a threshold T_t. Under the assumption that $\lambda \gg r$ and that $E[B]$ is given as in Theorem 2.2 , we have:

Theorem 2.3 *r^* is bounded by*

$$r^* \leq \frac{T_t\mu - s + \sqrt{s^2 - 2T_t\mu s + T_t^2\mu^2 - 2sE[B]\mu + 2E[B]\mu^2 T_t}}{2(s - T_t\mu)E[B]}$$

Proof: Follows from equating (6) to T_t and solving for r. We have an inequality (rather than an equality) for r^* since the increase of $E[B]$ due to the increase of r is not taken into account when inverting (6). □

2.3.4. Premature Content Unavailability

Up to this point we assumed that content becomes unavailable only when the busy period ends, i.e., the number of peers goes to zero. In this section we consider the more general case where content may become unavailable even if the number of peers in the system is greater than zero.

While the publisher is available it can monitor the number of peers in the swarm and depart when the population size reaches a minimum critical size n. At this point, to save costs, the publisher goes to sleep mode. As far as the number of peers in the system is greater than $k < n$ (a parameter to be estimated) all the blocks are available with high probability.

Let $B[n, k]$ be defined as the expected length of a (modified) busy period which initiates with n peers and ends as soon as the population size reaches k. Just before entering in sleep mode the publisher can set an alarm to trigger after $B[n, k]$, at which point it returns. In what follows, we derive an expression for $B[n, k]$ assuming the arrival rate of peers to be λ and the expected average service time s/μ.

Theorem 2.4 *For all n,*

$$B[n, 0] = \Big[\sum_{i=1}^{n} \frac{s}{i\mu} + \sum_{i=1}^{\infty} \frac{(\lambda(s/\mu))^i (s/\mu)[1 - h(i/(s/\mu))]}{i!i}\Big]$$

where $h(\mathfrak{s}) = \prod_{i=1}^{n}(i\mu)/(\mathfrak{s} + i\mu)$. $B[n, k]$ can obtained using the following recursion:

$$B[n, k] = B[n, i] + B[i, k], \qquad \forall i : n < i < k$$

Proof: The result follows from equation (3). □

It seems hopeless to try to invert the formulas for $B[n, k]$ in order to obtain n as a function of s/μ and λ. However, we can always use a table of values for $B[n, k]$ and then find n^* such that if the server is turned off when the population reaches size n^* the content will be available for an expected time equal to $B[n^*, k]$.

3. A Controller For The Critical Regime

For swarms that are not self sustaining, we propose a controller that dynamically adjusts the rate at which the server uploads content to the swarm. The controller is simple: every w seconds, the server computes the difference between the number of arrivals and departures in the previous window of time. Whenever this difference is greater or equal to zero,

the server reduces the capacity allocated to that swarm. Whenever the difference is less than zero, the server increases the capacity allocated to the swarm. Note that in the extreme case where the number of departures is zero (e.g., if some chunks of the content are not available) the server always increase the capacity allocated to the swarm. We consider as exception the case when both arrivals and departures are equal to 0.

Figure 2 illustrates the process. At time $t1 + 10$, the server identifies that in the window $(t_1, t_1 + 10)$ there were no departures and therefore increases its upload rate. The server then waits for $s = t_2 - t_1$ seconds. A number of clients depart in the interval $(t_2, t_2 + 10)$, so that at instant $t_2 + 10$ the server can reduce its bandwidth again. The procedure is summarized in Algorithm 1. In the following section we will evaluate the efficiency of this controller using experiments.

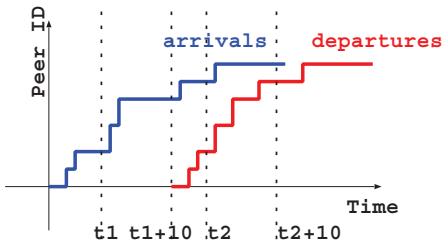


Figure 2. Typical arrival and departure of peers.

algorithm 1 Controller for Server Capacity

Step 1: $A \leftarrow$ number of arrivals in the last 10 seconds
Step 2: $D \leftarrow$ number of departures in the last 10 seconds
Step 3: If $A > D$, $R \leftarrow \min(R + \delta, C_{\max})$ else $R \leftarrow \max(R - \delta, 0)$
Step 4: Wait w seconds. Go to step 1.

4. Experimental Evaluation

In this section we empirically show the potential benefits of strategically setting the server rate by deploying several large scale experiments with private torrents swarms. The goals of the experiments are to study, using real BitTorrent: 1) the conditions under which swarms are self sustaining; 2) the impacts of the server strategy on its costs (in terms of bandwidth reduction) and delay (as perceived by the peers).

4.1. Experimental Setup

Our first sets of experiments (§ 4.2- 4.4) were conducted using 26 machines in a local cluster. Our last set of experiments (§ 4.5) were performed on Planetlab [Peterson et al. 2006]. Planetlab turned out to be very convenient for the large scale experiments. The results in § 4.5 involved more than 1200 BitTorrent clients connecting to a single swarm. For these experiments we used simultaneously roughly 300 stable Planetlab machines (out of a total of 936 nodes as of December, 2008).

For all experiments, we had a host at UMass as the controller and another host as a tracker. The controller host maintains a list of events to be executed. For each event we store its (1) type (leecher arrival, publisher arrival or publisher departure), (2) time, and (3) host name. When an event triggers, the controller dispatches the corresponding command via `ssh` to the appropriate node. When the experiment finishes, we collect and process all the traces. Each node's trace contains the evolution of the instantaneous

download and upload rates in time slots of 1 second as well as the fraction of the file downloaded until that time.

We used the official Bittorrent version 4.0.2 instrumented by Legout et al. [Legout et al. 2007]. The instrumented Bittorrent client logs every message sent or received by the peer, the content of the message, as well as every internal state change. The logging overhead was determined to have a negligible effect on client performance. We also made changes to the Bittorrent client source code including an option to dynamically change the upload rate.

4.1.1. Experimental parameters

Our experiments consist of torrents each one publishing a single file. These files are of size S MB. Peers arrive with rate λ peers/second. The uplink capacity of each peer is μ KBps and of the publisher is p KBps. The publisher alternates between being up and down in intervals of length U and D, respectively. Unless otherwise stated, the default values are $S = 4$ MB, $\mu = 100$ KBps, $p = 100$ KBps, $\lambda = 1/5$ peers/second. For the minimal units of shared content, we used the default BitTorrent values (chunks of 256 Kb and pieces of 16 Kb). The other parameters are described in the following section.

4.2. Self Sustainability of Swarms

Figure 3 shows the dynamics of peers arrivals and departures as a function of time. Each line represents the arrival and departure of a peer.

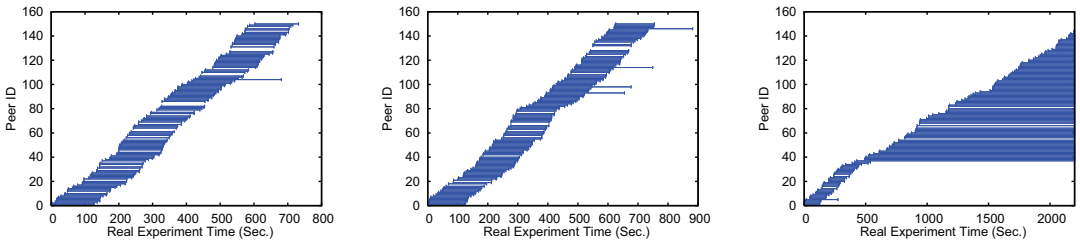
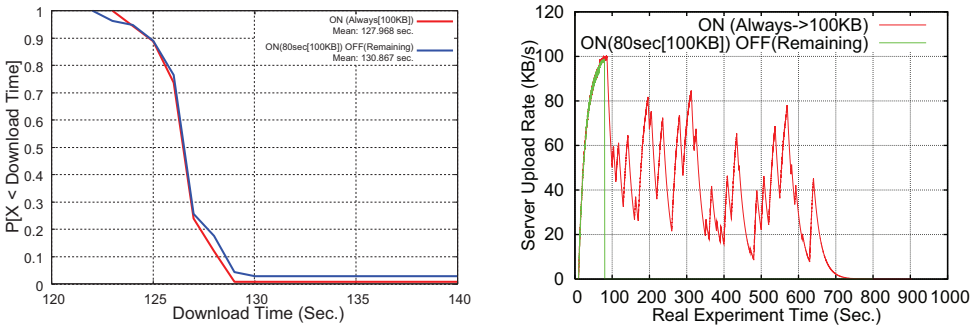


Figure 3. Swarms self sustainability.

In the experiment shown in Figure 3(a) the publisher stays online all of the time ($U = \infty$). One notes that every peer that arrives to the swarm is served with success and leaves. We are interested in understanding what the role of the publisher is in the system. In particular, is the presence of the publisher a necessary condition for the peers to conclude their downloads? Figure 3(b) indicates that this is not the case (i.e., the swarm is self sustaining). In the scenario of Figure 3(b) the publisher stays on-line for only 80s ($U = 80s$, twice the time necessary to upload a file of size $S = 4MB$ with an upload rate of $p = 100KBps$) and all peers (but the last one) are able to conclude the download. The rarest-first mechanism implemented in BitTorrent [Legout et al. 2007] plays a key role to guarantee that all the blocks are available even in the absence of the publisher.

The situation changes if we reduce the arrival rate of the peers, λ, from 1/5 to 1/15 peers/sec. Figure 3(c) shows this scenario. The publisher stayed on-line only for the first 80 seconds ($U = 80s$). After 500 seconds no peer was able to finish the download.

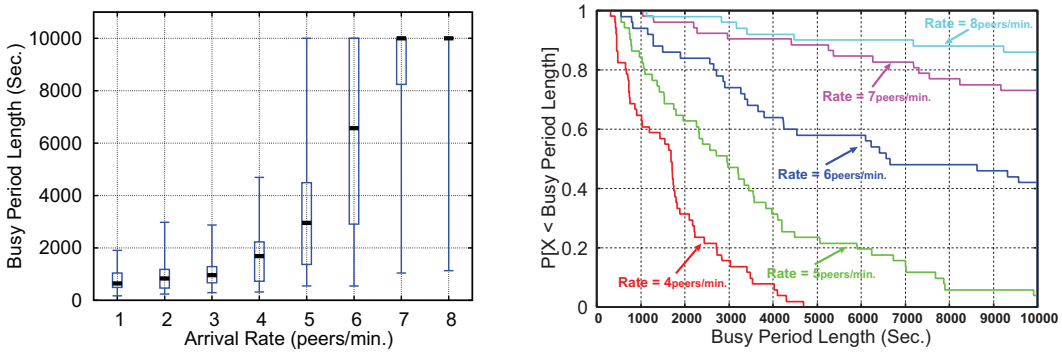


(a) (b)

Figure 4. Efficiency on self sustaining swarms.

Figure 4(a) shows the download time distribution of peers for the first two experiments described above (Figure 3(a) and (b)). We note that the effect of the server strategy on the expected download time is marginal. Nevertheless, the server strategy has remarkable impact on bandwidth savings: Figure 4(b) shows the upload rate of the server as a function of time. In the case where the publisher stays on-line all the time (red curve) the upload rate varied between 20 and 80 KBps. In the case where the publisher is turned off 80 seconds after its arrival (green curve) the server incurs zero cost after the initial regime.

4.3. The Threshold for Self Sustainability



(a) (b)

Figure 5. The threshold for self sustainability: (a) Busy Period Length Distribution; (b) CCDF Busy Period Length.

Now we wish to understand the conditions under which peers will get blocked (due to the unavailability of blocks) as a function of their arrival rate (all the other parameters kept constant). Our performance metric is the length of the busy period, i.e., how long the swarm can survive serving peers after the publisher leaves. We conducted 50 runs of the experiment for each arrival rate. We varied the arrival rate (λ) from 1 to 8 peers/minute. Each run stops when one of the following two conditions is satisfied: (i) the population size reaches size 100 (which is an indication that all the peers are blocked); (ii) the run reaches 10000 seconds. The distribution of the *Busy Period Length* (average duration of a run) as a function of the arrival rate is summarized in Figure 5. Figure 5(a) shows using boxplots the quartiles (25%, 50% and 75%) of the *Busy Period Length* distribution, as well as the minimum and maximum observed values. We see that the median increases with arrival rate. For arrival rates λ=5,6,7, 8 there were a number of swarms that survived

up to the limit of 10000 seconds. Figure 5(b) shows the distribution of the busy period for λ=4,5,6,7,8. For $\lambda \geq 6$, P[Busy Period Length $>$ 10000] $\geq$ 0.42. For $\lambda = 8$, P[Busy Period Length $>$ 10000] $\geq$ 0.85 which indicates that for high values of λ the chances that peers get blocked is small even if the server stays on line only for a small window of time.

4.4. Upload Controller

Next we ran experiments considering the scheme described in § 3 to throttle the server capacity. In this set of experiments we used a controller that dynamically adjusts the upload rate at which the publisher serves content to the swarm. For the first 80 seconds the publisher operated at 100KBps. After that, its bandwidth is adjusted dynamically in the range of 1-100KBps. We then reproduced the experiment with the same arrival pattern but considering a server that did not use the controller to adjust the upload rate. The results are shown in Figure 6.

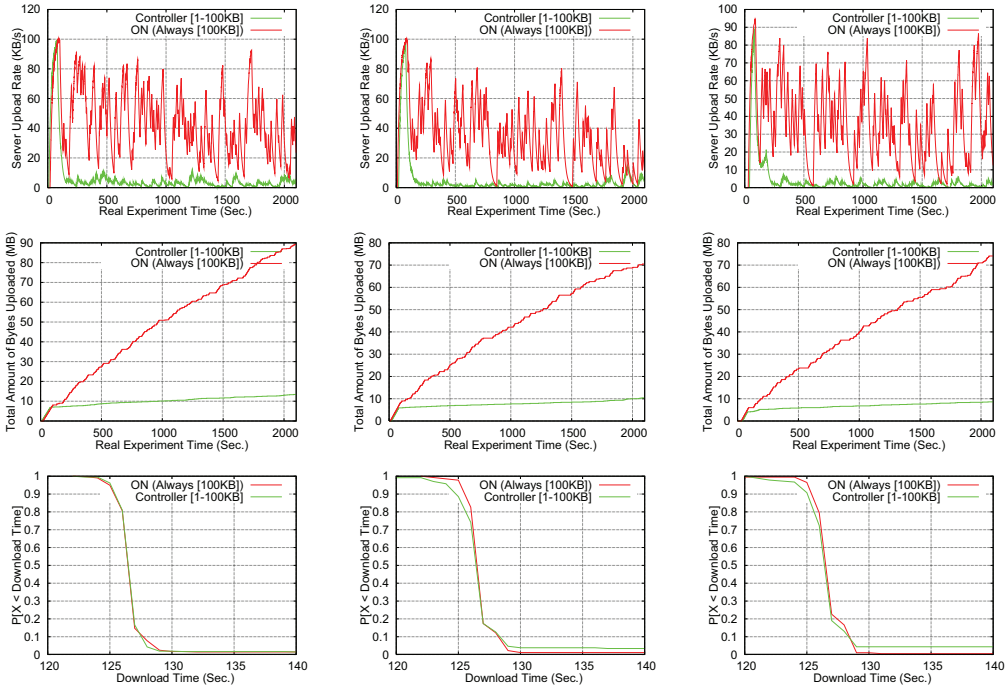


Figure 6. Experiments using upload controller: (a) λ=1/5 peers/s; (a) λ=1/8 peers/s; and, (c) λ=1/10 peers/s

The results for λ=1/5, λ=1/8, and λ=1/10 are shown, respectively, in Figures 6(a), (b), and (c). The upper graphs show the upload rate of the servers as a function of time. We note that after the initial regime (80 seconds), the servers using the controller experienced significant reduction in terms of bandwidth usage. The middle graphs confirm this trend, showing that the number of total uploaded bytes grows much slower when using the dynamic rate allocation. The bandwidth savings are above 80% for the three experiments. The lower graphs compare the download time distribution of peers for both schemes. To sum up, the usage of the server controller lead to a significant reduction in terms of bandwidth, with a negligible increase in the download time experienced by the users.

4.5. Bursty Arrivals

In this section we evaluate the applicability of our rate controller in the presence of bursty arrivals. For this purpose, we model the arrival rate of peers as a Markov modulated Poisson process (MMPP). The MMPP has 3 states s_1, s_2 and s_3. The transition rates $\gamma_{1,2}, \gamma_{2,3}, \gamma_{3,2}, \gamma_{2,1}$ are 0.015, 0.0255, 0.05 and 0.03, respectively. The arrival rates in the three states (s_1, s_2 and s_3) are 0, 0.2 and 1, respectively. This allows us to capture burtiness in the arrival process. The other parameters were set to $S = 10$ MB, $\mu = 50$ KBps, and the publisher operated with $p = 100$ Kbps for the first 200 seconds, before starting the controller.

Figures 7(a) and 7(b) show the dynamics of arrivals and departures of peers when the server is always on and when the controller is used, respectively. Qualitatively, the dynamics are very similar which means that peers experience approximately the same download time in both cases (quantitatively, the mean download times are 250 and 260 seconds, respectively). Nevertheless, the cost reduction for the provider is substantial when the controller is used. Figure 7(c) shows that the number of uploaded bytes when the publisher is always on grows linearly as a function of time with slope close to 0.1, while the growth is much slower if the controller is used.

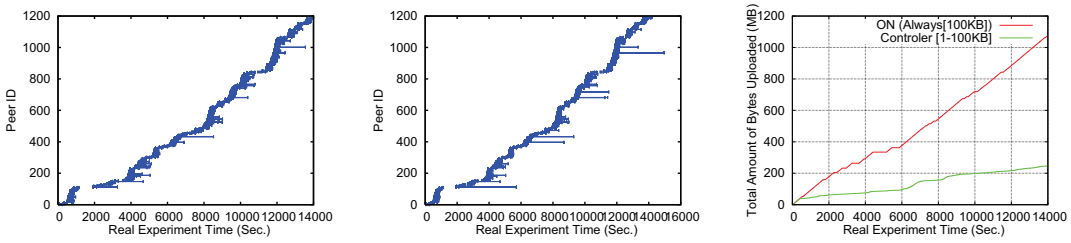


Figure 7. Bursty arrivals.

5. The Multiswarm Case

In this section we study the scenario where a provider for N swarms ($k = 1, \ldots, N$) splits its capacity μ bps across the swarms.

Let μ_i be the download rate achieved by the users of swarm i, in bps, without accounting for the server help. We order the swarms in such a way that $\frac{1}{\mu_1} \geq \frac{1}{\mu_2} \geq \ldots \geq \frac{1}{\mu_N}$. Let s_i be the size of file i in bits and n_i the expected number of users in swarm i.

The time for a member of swarm i to complete its download is given by:

$$T_i = \frac{1}{\frac{\mu_i}{s_i} + \frac{\mu f_i}{n_i s_i}} = \frac{s_i}{\mu_i + \frac{\mu f_i}{n_i}} \tag{7}$$

where f_i is the fraction of the server capacity allocated to swarm i. Note that μ_i/s_i gives the rate of number of download completions per unit of time, assuming that only peers of the swarm help each other, while $f_i\mu/n_i$ is the rate dedicated by the server to a member of swarm i. $\frac{f_i\mu}{s_i n_i}$ is the rate of download completions per unit of time if users rely only on the server.

Our metric of interest is the expected download time, $E[T]$, $E[T] = \sum_{i=1}^{N} \lambda_i T_i = \sum_{i=1}^{N} n_i$, $n_i = \lambda_i T_i = \frac{\lambda_i s_i}{\mu_i + \frac{\mu f_i}{n_i}} \Rightarrow n_i = \frac{[\lambda_i s_i - \mu f_i]^+}{\mu_i}$.

Note that in the case where all μ_i are equal, the expected download time does not depend on how the server splits its capacity (as soon as $f_i < \lambda_i s_i/\mu$, for $i = 1, \ldots, N$). In what follows, we assume that all μ_i's are distinct.

5.1. Criteria 1: min E[T]

We wish to minimize the expected download time of the peers. Ignoring the non-negativity constraints of n_i, its derivative with respect to f_i is given by $\frac{dn_i}{df_i} = -\mu/\mu_i$ which implies that as we increase f_i the number of users in swarm i decreases, and if our goal is to minimize $\sum n_i$ we must prioritize the swarms where $1/\mu_i$ is higher. This leads to the following algorithm:

algorithm 2 Server Capacity Allocation: min E[T]

Step 1: $i \leftarrow 1$, $sf \leftarrow 0$
Step 2: $f_i = \min(1 - sf, \lambda_i s_i/\mu)$
Step 3: $sf \leftarrow sf + f_i$
Step 4: If $sf < 1$ and $i < N$, $i \leftarrow i + 1$, go to step 2.

5.2. Criteria 2: min max T_i

Our goal now is to minimize the maximum of the download times, i.e., $\min \max T_i$. The solution follows the water filling scheme [Boyd and Vandenberghe 2004].

Let the swarms be ordered in such a way that: $s_1/\mu_1 \geq s_2/\mu_2 \geq \ldots s_N/\mu_N$. Starting from the initial scenario where the server does not provide any assistance to any swarm, users of swarm 1 experience the highest delay, which from equation 7, is given by, $T_i = \frac{1}{\frac{\mu_i}{s_1}}$.

Allocating enough capacity to this swarm so as to make the delay experienced by its users equal to the delay experienced by users in swarm 2, and repeating the process as many times as possible, we obtain the following algorithm:

algorithm 3 Server Capacity Allocation: min max T_i

Step 1: $i \leftarrow 1$, $\epsilon \leftarrow 0.0001$
Step 2: for j=1..i, $g_{j,i} = \frac{1}{\mu}\left(s_j - \frac{\mu_j s_{i+1}}{\mu_{i+1}}\right)$
Step 3: $sf \leftarrow \sum_{j=1}^{i} g_{j,i}$
Step 4: If $sf < 1$ and $i < N$, $i \leftarrow i + 1$, go to step 2.
Step 5: while $sf > 1$
Step 5.1: for j = 1..i, $g_{j,i} = \frac{1}{\mu}\left(s_j - \frac{\mu_j (s_{i+1} - \epsilon)}{\mu_{i+1}}\right)$
Step 5.2: $\epsilon \leftarrow 2\epsilon$
Step 5.3: $sf \leftarrow \sum_{j=1}^{i} g_{j,i}$
Step 6: for j = 1..i, $f_j = g_{j,i}$

Note that $f_{g,i}$ corresponds to allocating to swarm g just enough capacity so that the delay experienced by its users is equal to the one experienced by users of swarm $i + 1$ in the case where no server capacity is allocated to swarm $i + 1$: $\frac{s_g - \mu f_{g,i}}{\mu_g} = \frac{s_{i+1}}{\mu_{i+1}} \Rightarrow \mu f_{g,i} = s_g - \frac{\mu_g s_{i+1}}{\mu_{i+1}} \Rightarrow f_{g,i} = \frac{1}{\mu}\left(s_g - \frac{\mu_g s_{i+1}}{\mu_{i+1}}\right)$. The process is repeated until reaching the last swarm or $\sum_{g=1}^{i} f_{g,i} > 1$, whichever comes first.

6. Related Work

The literature on the enterprise use of peer-to-peer systems is very scarce and mainly composed of white papers [About.com 2005]. [Cohen 2005] pointed out that "making seeding optimizations for enterprise use" is one of the "puzzles" to be solved to improve the BitTorrent protocol. The creators of Kontiki [Kontiki 2008], a commercial peer-to-peer delivery system, reported some of the challenges involved in the deployment of the system. Akamai [Akamai 2007] has recently started adopting peer-to-peer for content dissemination. However, none of the white papers presents quantitative evaluations of the impact of server strategies.

Seeding behavior in BitTorrent has been explored in the context of incentive mechanisms. For instance, super seeding [Wikipedia] was proposed as a feature to attempt to minimize the amount of data uploaded by the original seed until the first completion of a downloading peer. Initially, the seed claims to have no piece but as peers connect to it, it informs that it received a new piece and allows the peer to download it. The seed does not upload any other piece to that peer until other peers advertise that they received the piece. [Bharambe et al. 2005] and [Legout et al. 2007] also studied strategies for initial seeding behavior. [Chow et al. 2008] considered the problem of how to better utilize the capacity of the seeds even after the initial seeding stage. These works are complementary to ours, since we study the impact of the popularity of the files on the seeding strategy. While the related work focuses mainly on incentive mechanism, our key concern is with performance under the assumption that clients do not misbehave.

Still under the scope of incentive mechanisms, but closely related to our work, is the paper by [Ramachandran et al. 2007]. The authors study the blocked leecher problem (BLP), where peers may have to wait for a long period of time for some blocks of the file that are no longer available. To address the problem, they propose BitStore, a token based incentive architecture to obtain the missing blocks cached at other peers that had already downloaded the whole file. Again, our focus is not on the incentive mechanisms but rather on server strategies for disseminating content to collaborative clients.

Enterprise peer-to-peer systems are under the category of hybrid peer-to-peer systems [Yang and Garcia-Molina 2001, Lu and Callan 2003]. [Ioannidis and Marbach 2008] formally analyze the design of hybrid peer-to-peer systems for content delivery. Even though they report that for some numerical experiments they observed zero traffic at the server (which corresponds to a self sustaining swarm), these results focused only on the search aspect of the problem, i.e., on how users can find the desired content.

An orthogonal but important aspect of enterprise peer-to-peer networks is the search mechanism. Peers must be able to find the desired content, and for that purpose architectures such as DONA [Koponen et al. 2007] and DOT [Tolia et al. 2006] are applicable. Note, however, that these architectures do not specify how content should be transmitted, therefore the techniques presented in this paper could be used jointly with one of the proposed architectures.

7. Conclusion

In this paper we analyzed enterprise peer-to-peer systems. We proposed and characterized the concept of a self sustaining swarm, i.e., a swarm that does not need to rely on the

publisher to get its content disseminated. After that, we showed that for swarms that are not self sustaining a simple controller can be used to decide how much bandwidth should be allocated as a function of the arrival and departure rates of peers. Through emulations, we showed that (i) a swarm can sustain itself even in the absence of publishers; (ii) the proposed rate controller is promising to reduce distribution costs. Finally, we presented an optimization framework for the multi-swarm case where an enterprise needs to share its capacity across multiple swarms.

References

About.com (2005). Enterprise technology: Peer-to-peer gets down to businees. `http://pcworld.about.com/magazine`.

Akamai (2007). Akamai goes peer to peer. `www.forbes.com`.

Anderson, C. (2006). *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion.

Bharambe, A. R., Herley, C., and Padmanabhan, V. N. (2005). Some observations on bittorrent performance. In *ACM SIGMETRICS'05*.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Browne, S. and J.M.Steele (1993). Transient behavior of coverage processes with applications to the infinite server queue. *J. Appl. Prob.*, 30 (3):589–601.

Chow, A., Golubchik, L., and Misra, V. (2008). Improving bittorrent: A simple approach. In *IPTPS'08*.

Cohen, B. (2003). Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer Systems*.

Cohen, B. (2005). Interview with Bram Cohen, the inventor of BitTorrent. `http://torrentfreak.com/`.

Ioannidis, S. and Marbach, P. (2008). On the design of hybrid peer-to-peer systems. In *SIGMETRICS'08*.

Kontiki (2008). The power of commercial peer-to-peer delivery. `www.kontiki.com/_download/The-Power-of-Commercial-P2P.pdf`.

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. (2007). A data-oriented (and beyond) network architecture. In *SIGCOMM'07*.

Legout, A., Liogkas, N., Kohler, E., and Zhang, L. (2007). Clustering and sharing incentives in bittorrent systems. In *ACM SIGMETRICS'07*.

Lu, J. and Callan, J. (2003). Content-based retrieval in hybrid peer-to-peer networks. In *CIKM'03*.

Page, W. (2008). More long tail debate: mobile music no, search yes. `http://longtail.typepad.com/`.

Peterson, L., Bavier, A., Fiuczynski, M. E., and Ly, S. M. (2006). Experiences building planetlab. In *USENIX OSDI*.

Ramachandran, A., das Sarma, A., and Feamster, N. (2007). Bitstore: An incentive compatible solution for blocked downloads in bittorrent. In *Proc. Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing*.

Tolia, N., Kaminsky, M., Andersen, D., and Patil, S. (2006). An architecture for internet data transfer. In NSDI'06.

Wikipedia. Super seeding entry. `http://en.wikipedia.org/wiki/Super-seeding`.

Yang, B. and Garcia-Molina, H. (2001). Comparing hybrid peer-to-peer systems. In *VLDB'01*.