CheatSheet: A Model-Driven Approach for Capacity Allocation in Managed Swarming

Abhigyan Sharma
University of Massachusetts Amherst

Antonio A. A. Rocha
Federal University of Rio de Janeiro, RJ, Brazil

Arun Venkataramani
University of Massachusetts Amherst

ABSTRACT

In this paper, we address the problem of allocating server capacity across competing swarms in a managed swarming environment. To this end, we make two contributions. First, we develop a detailed measurement-based model to quantify the relationship between swarm performance and the amount of server bandwidth supplied. Second, we present the design and implementation of CheatSheet, a system to allocate server capacity to competing swarms so as to maximize the average download rate or to achieve a set of target download rates while minimizing the total amount of server bandwidth consumed. The key strength of CheatSheet is its use of the measurement-based model that enables it to converge to the optimal allocation in a single round. We evaluate an implemented prototype of CheatSheet and show that it significantly improves swarm performance and server bandwidth cost compared to BitTorrent as well as state-of-the-art approaches to control capacity allocation in an online manner. For the max-min metric for download rate among swarms, CheatSheet performs better than the online strategy by 10% and a naive controller strategy by 20%. It also enables bandwidth savings compared to online controller strategy by as much as 50% in some cases.

1. INTRODUCTION

Peer-to-peer swarming systems such as BitTorrent [10] have witnessed tremendous success and growth over the last decade. However, although P2P swarming has been widely adopted for disseminating user-generated or pirated content, P2P strategies have largely been shunned by large-scale commercial content distribution services (e.g., Netflix, iTunes etc.). One reason is that commercial content distributers fear being associated with the piracy "bad rep" of P2P systems. However, a more fundamental reason is that P2P systems do not offer predictable performance and are difficult to manage, which makes them unattractive for commercial content distribution.

An emerging trend that portends to change this state of affairs is the emergence of managed swarming [25, 26] for large-scale commercial content distribution. Managed swarming seeks to achieve the best of client-server dissemination and peer-to-peer swarming, namely the predictable performance and ease of management of the former and the scalability and bandwidth cost savings of the latter. To this end, managed swarming seeks to opportunistically leverage P2P bandwidth when available falling back to server bandwidth otherwise. As server bandwidth is limited and expensive, a key challenge in large-scale managed swarming is to allocate server capacity across competing swarms so as to optimize system-wide performance or bandwidth cost objectives.

In this paper, we present the design and implementation of CheatSheet, a novel "offline control" approach to optimize capacity allocation in managed swarming. The key strength of CheatSheet is a detailed measurement-based model that quantifies the performance or *health* of a swarm as a function of the server capacity allocated to it. Through extensive measurements based on running large-scale BitTorrent swarms, we show that the health of a swarm shows a consistent and predictable relationship to the server bandwidth, peer arrival rate, the distribution of peer upload capacities, and the size of the file. Furthermore, we show that this model can be stored in a concise manner while preserving much of its accuracy. To our knowledge, our work is the first to develop a comprehensive *predictive* model of swarm health.

The measurement-based predictive model of swarm health enables CheatSheet to easily optimize a variety of objectives such as maximizing the minimum download rate across swarms, maximizing the average download rate across swarms, or minimizing the server bandwidth consumed in order to achieve specified target download rates for all swarms. More importantly, the model enables CheatSheet to converge to the optimal allocation in a single round. In contrast, state-of-the-art systems such as Antfarm [26] for capacity allocation based on "online control" take several rounds of bandwidth probing and performance feedback in order to converge to the desired allocation.

We have implemented a fully protocol-compliant prototype of CheatSheet based on BitTorrent and instantiated with all of the above three objectives. We evaluate CheatSheet through large-scale experiments over PlanetLab against BitTorrent with static allocation policies as well as online control strategies. Our results show that CheatSheet's offline control approach yields up to 20% improvement in swarm performance compared to BitTorrent and can reduce server bandwidth cost by up to 50% compared to an online control strategy.

2. BACKGROUND AND GOALS

We assume that the reader is familiar with peer-to-peer swarming and BitTorrent [10] in particular. A brief introduction may be obtained in [18] or the background sections of recent papers on swarming (e.g., [27, 23]).

Managed swarming seeks to get achieve the best of client-server file distribution and peer-to-peer swarming, namely, the predictable performance and ease of management of the former and the bandwidth cost savings of the latter. In addition to a tracker for each file or bundle, we assume the presence of an always online seed (also referred to as the *publisher* or *server*). We refer to users interested in downloading the file simply as *peers* and assume that they are not altruistic, i.e., they leave as soon as they complete their download.

The primary goal of managed swarming is to opportunistically leverage peer-to-peer bandwidth when possible and fall back to the publisher's bandwidth otherwise. Server capacity in practice is limited and incurs a higher cost with higher usage, so a large content publisher may wish to opportunistically leverage peer-to-peer bandwidth for one of two broad classes of objectives. The first is to allocate the total amount of available server capacity across all active swarms so as to maximize the overall performance. The second is to achieve a specified target performance level for all swarms while minimizing the server bandwidth usage. We consider both classes of objectives in this paper.

The server capacity allocation problem would be straightforward if the capacity contributed by peer-to-peer swarming were relatively insensitive to the server bandwidth available. Ideally, the aggregate bandwidth would simply be aggregate upload capacities of peers concurrently downloading the file plus the bandwidth supplied by the server. Indeed, this is a common assumption made by textbook presentations of peer-to-peer systems [18] or simplistic theoretical models of BitTorrent performance that incorporate a constant inefficiency factor associated with the utilization of peer-to-peer capacity [28]. Although such simplistic models are well suited for pedagogical purposes or for deriving theoretically tractable performance models, they do not adequately capture the dependence of swarming efficiency on the server bandwidth.

To appreciate this claim, consider Figure 1 results of a set of experiment conducted by running a swarm using real BitTorrent clients on PlanetLab. Peers arrive into the swarm at the rate of one every 12.5 seconds each with an upload capacity of 200 KBps. Each point shows the average download rate experienced by peers as a function of the corresponding publisher bandwidth. We defer further details of the experimental setup to Section 3.2.1.

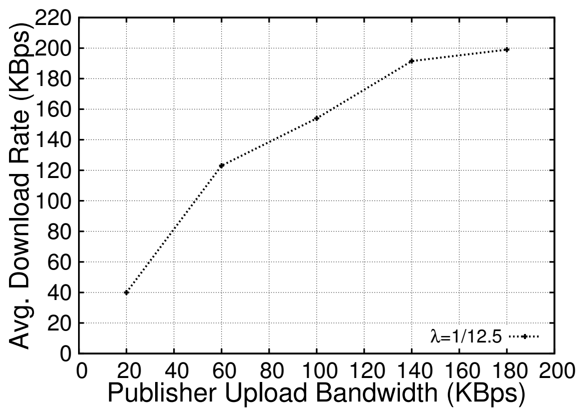


Figure 1: Average peer download rate as a function of the publisher bandwidth.

Figure 1 clearly shows that swarming efficiency is quite sensitive to the publisher bandwidth. For example, increasing the publisher bandwidth from 20KBps to 60KBps improves the average download experienced by peers from 40KBps to 120KBps. The average number of peers in the swarm at any given point is TBD in the first case and TBD in the second case. Thus, increasing the server bandwidth by just 20KBps has increased swarming efficiency by almost 4×. If the server bandwidth is further increased, we notice that peers experience increased download rates, but diminishing marginal returns. For instance, increasing the publisher bandwidth from 140KBps to 180Kbps improves the average download by less than 20KBps. This is because the block diversity in the swarm increases as a function of the server bandwidth, favoring the saturation of the peers downlinks and uplinks, which in turn favors a slower growth of the average download rate of the peers.

Thus, in order to address the server capacity allocation problem, it is important to first quantify the impact of server capacity on the efficiency of peer-to-peer swarming.

3. A MEASUREMENT-BASED MODEL OF SWARMING EFFICIENCY

In this section, we formally introduce the problem of modeling swarming efficiency, explain why the problem is analytically challenging, and present an empirical measurement-based model of swarming efficiency.

3.1 Problem statement

Our goal is to quantify the performance of a swarm as a function of the publisher bandwidth and other relevant swarm parameters. Consider a swarm distributing a file of size S to peers arriving at a rate λ. Let x denote the bandwidth supplied by the publisher and μ the average peer upload capacity. Unless otherwise stated, we assume that the download capacity of peers is unlimited. Let y denote the average download rate of peers in the swarm. The model seeks to quantify y as a function of x, μ, λ and S as follows.

$$\frac{y}{\mu} = f(x, \tilde{\mu}, \lambda, S) \tag{1}$$

We refer to the left hand side above, i.e., the average download rate normalized by the average peer upload capacity, as the *health* of a swarm. The normalization allows us to compare swarms with very different characteristics against a common performance metric. We use $\tilde{\mu}$ to denote the distribution of peer upload capacities. Note that a swarm with uniform upload capacities may behave differently compared to one with a skewed capacity distribution (as is observed in practice) despite both having the same mean.

It is straightforward to intuit some aspects of the nature of $f(.)$. For example, increasing the publisher bandwidth (keeping all else fixed) always improves the health of the swarm. Intuitively, at very low values of x, the publisher bandwidth becomes a bottleneck and prevents potential opportunities for block exchanges between peers, thereby hurting the swarm performance y. Increasing x increases y both by improving the utilization of peer upload capacity as well as simply by providing more bandwidth as in a client-server system. Thus, $f(.)$ must be monotonically increasing in x.

Similarly, it can be argued that a higher arrival rate implies a healthier (unhealthier) swarm in the regime $x \ll \mu$ ($x \gg \mu$). When $x \ll \mu$, a very low arrival rate effectively reduces the swarm to a client/server system, so y is limited by and equal to x. Increasing the arrival rate increases opportunities for peer-to-peer exchanges and improves y. On the other hand, when $x \gg \mu$, a significant fraction of the swarm's performance comes from the publisher as opposed to peer-to-peer interactions, so increasing the arrival rate reduces the download rate (despite the improved utilization of peer upload capacity) as there are more peers competing for the publisher's bandwidth. Thus, $f(.)$ increases (decreases) with λ when $x \ll \mu$ ($x \gg \mu$).

However, some aspects of $f(.)$ are not intuitive and even run counter to intuition. For example, increasing the peer upload capacity μ can in some scenarios hurt the download rate y (and by consequence the health y/μ as well). A higher peer upload capacity nominally enables existing peers to complete the download and depart quicker, but the resulting swarm consists of fewer peers and is less efficient in utilizing peer-to-peer exchange opportunities. The increased peer upload capacity is more than outweighed by the loss of efficiency in utilizing that capacity. Similarly, it is nontrivial to intuit the effect of increasing arrival rate or increasing file size except in the extreme cases ($x \gg \mu$ or $x \ll \mu$) outlined in the previous paragraph.

Analytical models.

If it were easy to analytically derive the relationship posed in equation (4), that would be sufficient for our modeling goals. However, deriving this relationship is nontrivial. In the appendix, we present a strawman analytical model that simplistically assumes that the health of a swarm is determined by the average number of peers in the swarm (as alluded to by prior work [26]). However, such models fail to capture even the concave nature of the health curve shown in Figure 1.

Our position is that developing a deeper theoretical understanding of a swarm's health requires as a first step a careful, empirical characterization of how the effectiveness of peer-to-peer exchanges depends on various swarm parameters. So, in the rest of this paper, we focus on developing a measurement-based model that is accurate enough to enable us to control a broad range of swarms encountered in practice.

3.2 Measurement-based model

In this section, we present a measurement-based model to predict the health of the swarm. A naive approach to this end would be to "measure" the relationship posed in Equation (4) for all foreseeable values of the four underlying dimensions $(x, \tilde{\mu}, \lambda, S)$. In contrast, the key strength of our model is that it is concise, i.e., a small number of measurements suffice to predict the health for a broad range of unmeasured scenarios. Furthermore, the measurements shed light on which aspects of the four dimensions are most critical to predicting the health, thereby improving our understanding of swarm behavior and the ability to control it.

We develop the measurement-based model by presenting a sequence of claims supported by experimental evidence as follows. We begin by describing the experimental setup.

3.2.1 Experimental setup

We used our experimental setups for measurements on the PlanetLab and a local cluster. The design of the experimental setup was identical for both experiments. Each node in an experimental setup was installed with a BitTorrent client. Two nodes were setup as the seeder and the tracker respectively. The seeder distributed a file of size S The upload capacity of the seeder node was fixed at a limit x KBps. Each experiment simulated a peer arrival at rate λ by starting a new BitTorrent client to join the swarm once every $1/\lambda$ seconds. The upload capacity of the peers was fixed according to a distribution. We let the swarm evolve until the arrival of at least 200 peers after which we terminate the experiment. We measure the download rates of the peers which finish download and the average value of the download rate is measured for the experiment.

In this manner, we collect data for a range of values of λ and by varying the seeder bandwidth $x = 1/10 \times \mu$ to $x = \mu$. We keep the values of S and μ fixed throughout. We collected data based on 5 runs of experiment for each tuple(x, μ, λ, S) on PlanetLab and a local cluster. On the local cluster, we chose the values of S = 20 MB, $\mu = 200 KBps$ a range of λ values from $\lambda = 1/3.125$ to $\lambda = 1/100$. On PlanetLab, we chose the values of S = 10 MB, $\mu = 100 KBps$ a range of λ values from $\lambda = 1/2$ to $\lambda = 1/100$.

3.2.2 *Dependence on x, μ, and λ*

CLAIM 1. $f(.)$ depends on x, μ and λ.

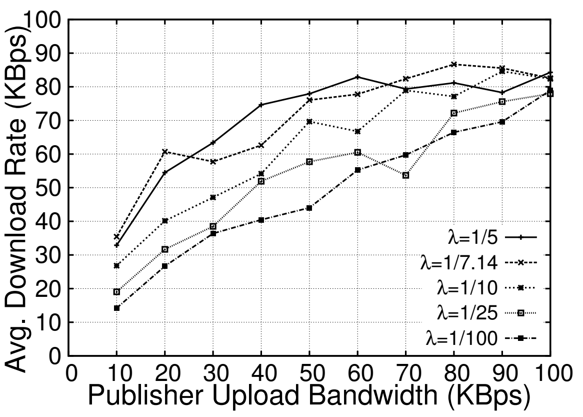


Figure 2: Dependence on x, μ, λ.

Figure 2 shows experimental evidence to support the above claim. Each line plots the average peer download rate as a function of the publisher capacity and different lines correspond to different arrival rates. Although some points buck the trend, the lines generally show an increasing and concave behavior. Each point in the figure represents the average of five runs, i.e., the average download rate across all peers averaged over five runs. The deviant points are an artifact of fickle PlanetLab nodes that impact some experiments and skew the average. For example, consider the erratically low value at x = 70KBps and y = 55Kbps on the line λ = 1/25s. The average download rates corresponding to the five runs are $(40, 70, 75, 73, 77)$ with each value being the average of the download rates of roughly 200 peers. The first run is clearly an outlier and is caused by the seeder, which was hosted at a PlanetLab node, being constrained in bandwidth due to other slices at the same node.

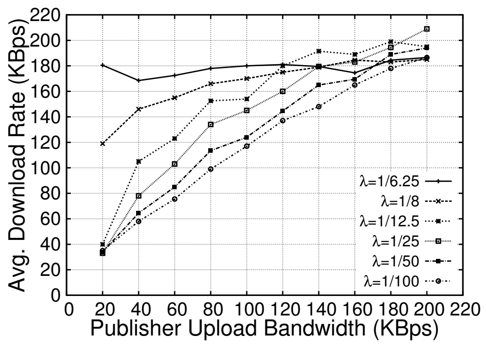


Figure 3: Dependence on x, μ, λ. Cluster with uniform peer upload capacities.

Figure 3 shows a similar experiment conducted over the local cluster and with peer upload capacities uniformly set to 200KBps. As before, each point is averaged over five runs. As expected, the increasing and concave trend is more clearly delineable in this case as the peer upload capacities are uniform and the cluster machines show more predictable behavior. We return to the impact of variance in peer upload capacities after analyzing the impact of file size below.

3.2.3 *Dependence on S*

CLAIM 2. $f(.)$ depends on S.

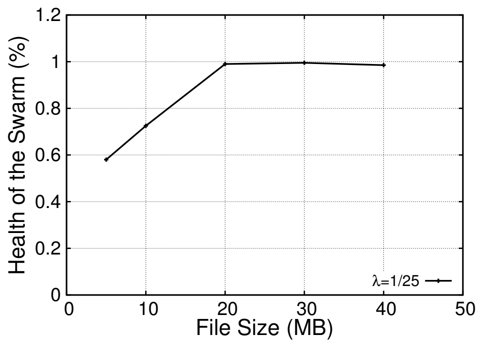


Figure 4: Dependence on S.

Figure 4 shows that the health of a swarm improves with the file size. In fact, a careful inspection of the data reveals that increasing S has an effect similar to increasing λ. The reason for this phenomenon is that the health of a swarm is in significant part determined by the size of the swarm.

Based on the above observations, we put forward the following candidate model.

$$\frac{y}{\mu} = f(\frac{x}{\mu}, \frac{\lambda S}{\mu}) \tag{2}$$

The thesis underlying the above model is that $f(.)$ does not depend on all possible combinations of (x, μ, λ, S), but only on two normalized quantities: the *publisher bandwidth ratio* x/μ, and the *healthy swarm coverage* $\lambda S/\mu$. The health of the swarm monotonically increases with x/μ and at $x = \mu$, the swarm is about as efficient as it gets. Note that even at $x = \mu$, y is less than μ (refer Figures 2 and 3). This is partly because of the protocol overhead (e.g., exchanging bitmaps, block requests, TCP inefficiencies etc.) and partly because peers sometimes do not possess useful blocks to exchange leading to underutilization of available capacity.

The health of the swarm increases with $\lambda S/\mu$ in the regime $x < \mu$. Note that $\lambda S/\mu$ would be the average number of peers in the swarm if it were perfectly efficient. The claim follows from Little's law assuming that S/μ is the average amount of time peers spend in the swarm. Hence, we refer to this quantity as the *healthy* swarm coverage. The actual average number of peers in the swarm on the other hand would be $\lambda S/y$ and is not useful for predicting y as it is determined by y itself.

In the regime $x > \mu$ (not shown in the above figures), the health of the swarm decreases as $\lambda S/\mu$ increases. This is because at $x > \mu$, the swarming efficiency is already maximal. Increasing x further only serves to reduce the average number of peers in the system, which in turn makes the swarm less healthy.

Unfortunately, although the model described by equation (2) is simple, it turns out to be inaccurate for two reasons. First, the model uses only the mean μ ignoring the effect of the underlying distribution. Second, the model ignores the impact of the size of the file, i.e., it assumes that the health is determined only by the healthy swarm coverage $\lambda S/\mu$ for large and small files alike. We discuss the impact of these two parameters next.

3.2.4 Dependence on upload capacity variance

CLAIM 3. $f(.)$ depends on the standard deviation σ of peer upload capacities.

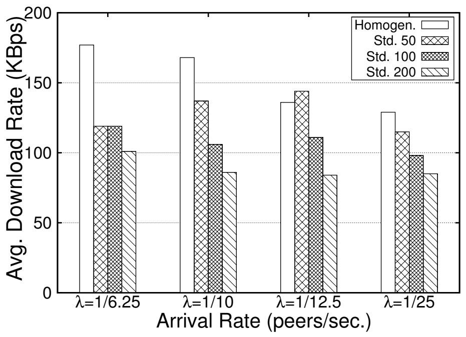


Figure 5: Dependence on variance.

Figure 5 shows that the health decreases with the variance in the distribution keeping the mean fixed. The experiment is done on the cluster so as to limit the variance introduced by the environment itself. For each arrival rate, the four bars plot the average download rate for increasing levels of variance in peer upload capacities keeping the mean fixed at μ = 200KBps. The first bar corresponds to a uniform distribution while the other three are drawn from a normal distribution with the specified mean and variance. The figure shows that high variance significantly reduces the health of the swarm, e.g., at an arrival rate of 1/10s, the health almost halves when the deviation in peer upload capacities is equal to the mean compared to the uniform distribution scenario.

How do we account for the impact of σ on the health so as to refine the model in equation 2? A cursory inspection of Figure 5 reveals that the impact of heterogeneity can not be simply captured as an "inefficiency factor" $v(\sigma)$ that can be latched onto equation (2) as $\frac{y}{\mu} = v(\sigma) f(\frac{x}{\mu}, \frac{\lambda S}{\mu})$, as the impact of σ at different arrival rates is different. A closer inspection of Figure 2 and Figure 3 (conducted with a heterogeneous and homogeneous distribution respectively) suggests that σ has the effect of reducing the healthy swarm coverage. For example, consider the lines corresponding to $\lambda = 5$ in Figure 2 and $\lambda = 6.25$ in Figure 3. In the first case, the healthy swarm coverage is equal to 20 compared to the second case where it is equal to 16. However, the line in Figure 2 starts to become unhealthy at larger values of x compared to the corresponding line in Figure 3 that remains largely self-sustaining.

Based on the above observation, we posit that σ affects the health a swarm as follows:

$$\frac{y}{\mu} = f(\frac{x}{\mu}, v(\frac{\sigma}{\mu})\frac{\lambda S}{\mu}) \qquad (3)$$

3.2.5 Further dependence on S/μ

Both of the models in equations (2) and (3) assume that the impact of the file size S is completely subsumed by the term $\lambda S/\mu$. We experimentally test the validity of this assumption next. If the assumption is true, then it must be the case that scaling x, μ, and S by the same factor leaves the health unchanged (as both x/μ and $\lambda S/\ u$ remain unchanged.

Figure 6 however shows that this assumption does not hold. The figure plots the health of the swarm as a function of the file size with each line corresponding to a fixed μ. The publisher capacity is fixed at $x = \mu$ for all lines, so we expect all swarms to be healthy. The arrival rate λ is set so that $\lambda S/\mu = 16$ remains a constant for all points on all lines in the figure. We chose the value 16 as it was found to be large enough to ensure that the swarm is healthy irrespective of the publisher capacity (based on the experiments in Figure 2). Now,

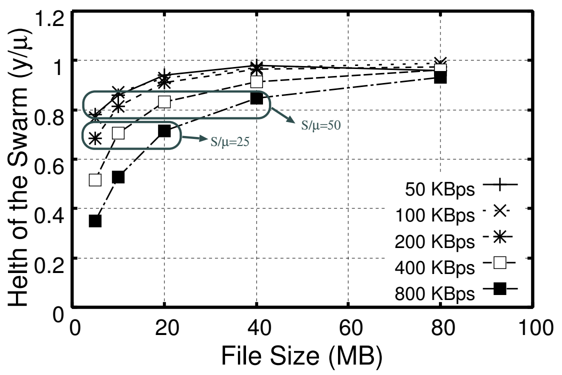


Figure 6: Dependence on S/μ.

if the model in equation (2) were true, the health of each line should be fixed at $f(1,16)$ and should be close to 0.9. However, we observe that not all swarms are that healthy and the health of a swarm generally decreases substantially as S decreases to small values.

We also observe that the health decreases as μ increases (i.e., we move vertically along the different lines), an observation that is consistent with and can indeed be explained by equation (2). A closer inspection of the data shows that if S and μ are both increased by the same factor K (so as to keep S/μ fixed), the health remains almost unchanged or increases very slowly if at all with K. For example, we see that the contour line for $S/\mu = 50$ overlaid on the graph is nearly horizontal.

Taken together, the above observations suggest that S/μ is by itself an independent determiner of swarm health, i.e., the health improves with S/μ even if $\lambda S/\mu$ remains unchanged. The reason S/μ improves the health is that it improves the effectiveness of peer-to-peer block exchanges. To appreciate this, let T denote the length of a choke/unchoke epoch in BitTorrent (typically 30 seconds). If S/μ is small, say equal to one epoch length, then peers have little time to improve their neighbor set. As $\frac{S}{\mu T}$ increases, the health rapidly improves. Once $\frac{S}{\mu T}$ is longer than about 10 epochs, the health is no longer limited by this metric. Based on these observations, we put forward the following claim.

CLAIM 4. $f(.)$ depends on $\frac{S}{\mu T}$ even if $\lambda S/\mu$ remains unchanged.

As with the deviation σ, we note that S/μ improves the health by effectively impacting the healthy swarm coverage term inside $f(.)$. This can be deduced, for example, by observing that for S = 10MB and μ = 100KBps, a healthy swarm coverage of at least 20 is needed for the swarm to be self-sustaining, however if both S and μ are scaled to 80MB and 800KBps respectively, a healthy swarm coverage of 16 suffices to make the swarm self-sustaining. Thus, the impact of $\frac{S}{\mu T}$ can be incorporated as modifying the healthy swarm coverage as in Claim 5 below.

3.2.6 *The complete model*

CLAIM 5. $f(.)$ only depends on the quantities $\frac{x}{\mu}$, $\frac{\lambda S}{\mu}$, $\frac{S}{\mu T}$, and $\frac{\sigma}{\mu}$ as follows.

$$\frac{y}{\mu} = f(\frac{x}{\mu}, d(\frac{S}{\mu T})v(\frac{\sigma}{\mu})\frac{\lambda S}{\mu}) \tag{4}$$

Concise representation.

All of the quantities in equation (4), $\frac{y}{\mu}$, $\frac{x}{\mu}$, $\frac{S}{\mu T}$, $\frac{\sigma}{\mu}$, $\frac{\lambda S}{\mu}$, are unit-less and lie in a bounded range. The first two are positive fractions. The latter three can take arbitrary positive values, however their range of interest is small in practice. For example, a healthy swarm coverage of about 20 suffices to make a swarm self-sustaining with no further room for improving the health. Similarly, values of $\frac{S}{\mu T}$ beyond about 10 do not improve the health as that is no longer the limiting factor.

The small bounded ranges enable us to store most of the information contained in the model in a concise manner as follows. For all ten values of x/μ belonging to the set $\{0.1, 0.2, \cdots, 0.9, 1\}$ and all $\lambda S/\mu$ in the set $\{1, 2, 4, 8, 10, 12, 16, 20\}$, we store the corresponding value of the health y. This approximate representation of $f(\frac{x}{\mu}, \frac{\lambda S}{\mu})$ consists of 80 values. Similarly, we store the functions $d(.)$ and $v(.)$ using five values each respectively from the range sets $\{1, 2, 4, 8, 16\}$ and $\{0.1, 0.25, 0.5, 1, 2\}$. Thus, the entire model can be approximately stored using under 100 values. The health can be estimated for combinations not stored in this representation by simply linearly interpolating between the stored values.

Validation.

We note that Claim 5 above is not one that can be proved or disproved. We can only assess how well the claim holds by exhaustively analyzing a large amount of experimental data (as is true of any modeling problem). Our position is that the many experiments presented above provide support for the claim. The more important question is if the above model is sufficiently accurate to enable us to control server capacity allocation effectively, a question that we address in detail in the next two sections. We do however present two experiments that serve to validate some of the predictive ability of the model.

First, we check how well we can predict the health of a swarm with heterogeneous peer upload capacities from data for $f(.)$ collected from swarms with homogeneous capacities, both with the same mean. If the dependence on σ can be characterized simply by the function $v(\sigma/\mu)$ that impacts the healthy swarm coverage, then it should be possible to predict a health curve in Figure 2 with arrival rate λ as simply the health curve corresponding to $v(\sigma/\mu)\lambda$ in Figure 3 for some constant value of $v(\sigma/\mu)$.

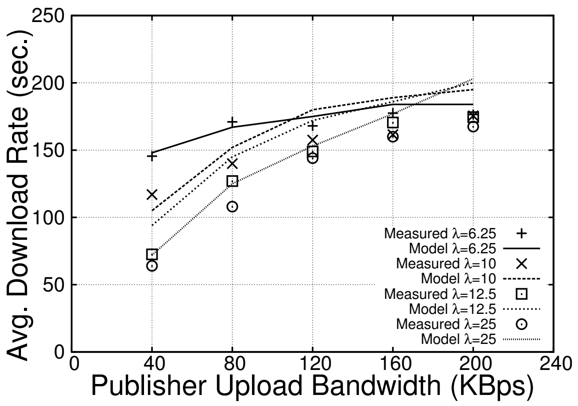


Figure 7: Predicting the health curves for a heterogeneous swarm using data from a homogeneous swarm.

Figure 7 shows the results for four such health curves by setting $v(\sigma/\mu)$ to 0.9. The lines correspond to the prediction using the homogenous data and the points correspond to actual measured experimental values. We see that the predicted lines capture the trend of the measured values well and in most of the cases are within 15% of the measured value.

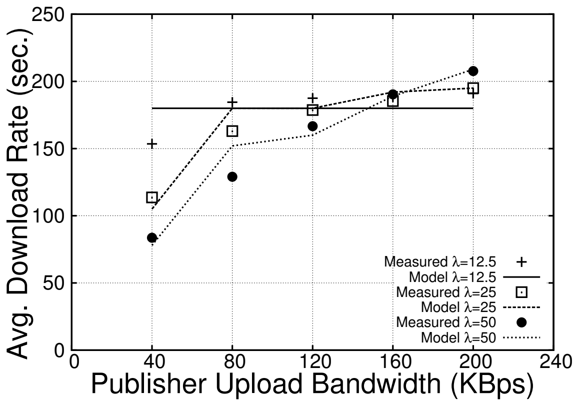


Figure 8: Predicting the health curves for a swarm with file size S_1 using data from a swarm with file size S_2.

In a similar vein, the experiment illustrated in Figure 8 checks how well the model can predict the health curve of a swarm with arrival rate λ_1 and file size S_1 using the data from the health curve of another swarm with arrival rate λ_2 and file size $S_2 > S_1$ such that $\lambda_1 S_1 = \lambda_2 S_2$ with all other parameters (x, μ, σ) being identical for both. The model would suggest that any health curve for the former (λ_1, S_1) can be predicted simply as the health curve of a swarm with file size S_2 and arrival rate $c\lambda_2$ where $c = d(\frac{S_1}{\mu T})/d(\frac{S_2}{\mu T}) < 1$ is a constant. Figure 8 shows the results for three such health curves, where $S_1 = 20MB$, $S_2 = 20MB$, and the constant $c = 0.9$.

Summary.

The most significant determiners of a swarm's health are the publisher bandwidth ratio x/μ and the healthy swarm coverage $\lambda S/\mu$. Two other factors secondarily impact the health, namely, the relative deviation σ/μ and the number of unchoke epochs $\frac{S}{\mu T}$ constituting a typical download. Their impact can be understood as changing the effective coverage $\lambda S/\mu$ via a slowly decreasing function $v(\sigma)$ and a slowly increasing function $d(S/\mu)$ respectively. The model can be approximately stored in a concise manner using a lookup table of about a hundred values.

4. SERVER CAPACITY CONTROL

In this section, we formally introduce the server capacity allocation problem, describe two approaches—online control and offline control—to address it, and then present the design and implementation of CheatSheet, an offline control system for server capacity allocation.

4.1 Problem

The server capacity control problem seeks to allocate server bandwidth to multiple swarms so as to optimize a system-wide objective. Let k denote the number of swarms managed by a server with total upload capacity X and $\{x_i, \tilde{\mu}_i, \lambda_i, S_i\}$, $1 \le i \le k$, denote the respective parameters of the k swarms. We consider several system-wide objectives for the server capacity control problem as follows.

MAX_MIN: The following optimization problem seeks to maximize the minimum download rate across swarms:

$$\max(\min_{1\le i\le k}(y_i)) \tag{5}$$

subject to

$$y_i = \mu_i f(x_i, \tilde{\mu}_i, \lambda_i, S_i), \quad 1 \le i \le k \tag{6}$$
$$\sum_{1\le i\le k} x_i \le X \tag{7}$$

MAX_AVG: The following objective maximizes the average download rate across all peers in all swarms:

$$\max \sum_{1\le i\le k} \lambda_i y_i \tag{8}$$

subject to the same constraints as (6) and (7) above.

MIN_COST: The following objective minimizes the server bandwidth cost while achieving a set of specified target download rates $(t_1, \cdots, t_k)$:

$$\min(x_1 + \cdots + x_k) \tag{9}$$

subject to

$$t_i = \mu_i f(x_i, \tilde{\mu}_i, \lambda_i, S_i), \quad 1 \leq i \leq k \tag{10}$$

4.2 Solution (on paper)

Solving any of the above optimization problems requires knowledge of $f(.)$. Furthermore, if $f(.)$ is known to satisfy certain monotonicity and/or concavity properties, the optimal solution can be efficiently computed using a simple greedy search strategy as described below. Below, we use the notation $f^{-1}(y, \mu, \lambda, S)$ to denote x such that $f(x, \tilde{\mu}, \lambda, S) = y$, and $f^{'}(x_i, \tilde{\mu}_i, \lambda_i, S_i)$ to denote the derivative with respect to x.

If $f(.)$ monotonically increases with x, MAX_MIN can be solved as follows. (1) Start with $y = \Delta$ for a small Δ; (2) Set $x_i = f^{-1}(y, \tilde{\mu}_i, \lambda_i, S_i)$, $1 \leq i \leq k$; (3) If $(\sum_i x_i < X)$, increment y to $y + \Delta$ and goto (2). It is straightforward to speed up the linear search using binary search.

If $f(.)$ is smooth and concave in x, MAX_AVG can be solved using a simple gradient ascent strategy that results in a unique solution. We start with $x_1 = x_2 = \cdots = \Delta$ for a small Δ and allocate the next Δ units of capacity (equally) to the swarm(s) with the largest value(s) of $\lambda_i f^{'}(x_i, \tilde{\mu}_i, \lambda_i, S_i)$, proceeding until all X units of capacity have been allocated. If $f(.)$ is piecewise linear and concave, the same strategy still works, but the resulting solution may not be unique.

If $f(.)$ is invertible, then MIN_COST can be solved by setting $x_i = f^{-1}(t_i, \mu, \lambda, S)$ if the resulting $\sum_i x_i < X$, otherwise the target allocation is not feasible.

4.3 CheatSheet design and implementation

The formal description above skirts several important practical design and implementation issues. Most importantly, we have assumed above that $f(.)$ is known a priori, but must be estimated in practice. There are two distinct design options available to address this problem: the *online* approach and the *offline* approach.

4.3.1 Online control

The online approach begins with no a priori knowledge of $f(.)$ (except for assumptions about its monotonicity and concavity) and estimates it in situ. For example, in order to optimize the MAX_AVG objective, the server capacity controller can implement the gradient ascent strategy described above by allocating Δ to each swarm in the beginning, monitoring the resulting download rates and the corresponding gradients, and allocating the next Δ to the swarm(s) with the largest gradient. Once all server capacity has been allocated, the controller must periodically perturb the allocations to all swarms by a small amount and, based upon the outcome, re-adjust the allocations. Antfarm [26] is an example of a system based upon the online approach (but for an objective different from MAX_AVG as described in [26]).

The periodic perturbations are required in the online approach for two reasons. The first is to gather additional data points so as to refine the estimate of $f(.)$. The second is to account for changes to the arrival rates of swarms, completion of swarms, and the initiation of new swarms. If the resulting gradients upon the perturbation are unequal by more than a threshold, the controller must move some capacity from swarm(s) with the smallest gradient to those with the largest gradient. Note that the online strategy does not need to know even μ_i and S_i for each swarm i, but simply the current value of (x_i, y_i) and the gradient $\lambda_i y_i^{'}$ at that point.

However, the online control approach suffers from several key shortcomings. First, estimating $f(.)$ based on online measurements is noisy and error-prone. Therefore, a large number of measurements are needed in order to fit a smooth or piecewise linear function to the measured scatterplot. Second, the estimated model needs to be updated if the peer arrival rate changes or a new swarm begins. As we show in the measurements in Section 3, $f(.)$ for different arrival rates can be dramatically different. Estimating the new $f(.)$ and ascending or descending its gradient incurs additional latency. Note that the convergence time of the gradient ascent strategy is limited by choice of the perturbation Δ in each step. Too large a Δ may be prone to perennial oscillations while two small a Δ can excessively inflate the convergence time and indeed fail to converge ever if the arrival rate changes often.

Third, and most importantly, an online control system is more complex to engineer for a broad range of objectives. The gradient ascent strategy is not applicable to the other two objectives MAX_MIN and MIN_COST, each of which requires a different kind of controller. For example, MIN_COST is better suited to an additive-increase multiplicative-decrease (AIMD) or MIMD based strategy to achieve the target download rates while minimizing cost. Each of these controllers further need to deal with measurement error and swarm dynamics, and carefully balance the dual concerns of convergence time and stability.

4.3.2 Offline control

CheatSheet employs an offline control approach that circumvents the problems with online control described above. The offline control approach knows $f(.)$ a priori as obtained from the measurement-based model described in the previous section. Although the model is not analytical, it can be stored concisely as a "cheat sheet" containing just over a hundred values and suffices to predict the download rate for a broad range of swarms $(x, \tilde{\mu}, \lambda, S)$ encountered in practice.

CheatSheet's offline computation is extremely efficient. All of the greedy strategies described above converge to the optimal allocation in a fraction of a second on commodity machines as computing $f(.)$ just involves a table lookup. The online controller can take many allocation or perturbation epochs each lasting on the order of minutes as it has to empirically measure the download rates for the most recent allocation. In contrast, CheatSheet converges to the optimal allocation in just a single epoch and re-adjust the allocation also in a single epoch if the arrival rates change.

A key requirement for the CheatSheet approach to be practical is that the server knows the parameters $\tilde{\mu}, \lambda$, and S. Of these, S and λ are already known to or easily tracked by existing trackers, but estimating the peer upload capacity distribution $\tilde{\mu}$ is nontrivial. There are several ways to alleviate this problem. First, our measurements suggest that it largely suffices to know the mean and deviation of the upload capacity as opposed to the exact distribution. Second, the server may be able to obtain a sufficiently accurate estimate by sampling the upload capacities of a fraction of the peers. A peer's upload capacity can be inferred either by monitoring its bitmaps (as has been demonstrated by previous works []) or by occasionally requesting a block from a peer and passively monitoring the TCP packet traces to estimate the bottleneck capacity. Furthermore, the server can conduct such measurements at coarse-grained intervals (e.g., once every couple months) by probing peers interested in the file (or even a broader content category) assuming that the distribution changes slowly.

Third, if the server can infer either just one of the two quantities–the mean μ or the deviation σ in upload capacities–it can refine its estimate of the other by monitoring the current average download rate y and reversing $f(.)$ (equation 4). The average download rate can be inferred either by monitoring bitmaps or simply by monitoring peer arrivals and departures. This approach increases the convergence time from one epoch to two epochs, but is significantly simpler than directly measuring or estimating the upload capacity distribution using any of the techniques described above.

4.3.3 *CheatSheet implementation*

We have implemented a prototype of CheatSheet with support for all three objectives described in Section 4.1. The current implementation assumes that the mean and variance of the peer upload capacity distribution is known. The model requires the value of peer arrival rate (λ) which is obtained from the tracker node. Other inputs to the model, i.e, the file size (S) and the publisher bandwidth (x) is available at the seeder node.

CheatSheet system is implemented in Python and consists of approx. 5000 lines of code. The system does not require any modification to the BitTorrent protocol for either the peers or the tracker. Our implementation uses an instrumented BitTorrent client developed by Legout et al [21]. The only modification we made to the client was the ability to change the maximum upload bandwidth for the client without restarting the client at the publisher.

4.4 Implementation of Online controllers

We describe the implementation of online controller strategies for the objectives in Section 4.1.

The design of online controller depends on two key parameters: (1) Bandwidth update interval (t) (2) Value of the bandwidth change for each interval (Δ). Empirically, we observe that a value of Δ equal to one-tenth of the upload capacity balances the convergence time as well as the stability of the controller. We choose a value of $t = 200$ seconds and decide the publisher bandwidth allocation based on the average download rate of swarm in the past 100 seconds. We find that these values are reasonable to remove the effect of spikes in the average download rate data (Figure 10). The exception to these values is in the case of Antfarm in which case we choose larger values of Δ and a smaller value of t such that it can quickly build a response curve for each swarm.

MaxMinHeur: This heuristic algorithm seeks to optimize the MAX_MIN objective function. It starts with an equal split of server bandwidth among all swarms, which is retains for the period $t0$. Thereafter, every $t1$ seconds, it increases the server bandwidth by Δ for each swarm which has less than the median average download rate among swarms. Conversely, the controller reduces the server bandwidth by Δ for each swarm with higher than the median average download rate. The values of the parameters we use are as follows: $t0 =$ 500s, $t1 =$ 200s, $\Delta =$ 10KBps .

Antfarm: We implement an online controller for the MAX_AVG objective based on the algorithm for the Antfarm system [26]. The Antfarm system is not publicly available.

The controller first allocates a bandwidth Δ to all swarms for a time interval $t0$. Then, after each $t0$ time interval, it increases the publisher bandwidth by Δ for the swarm which shows the maximum increase in bandwidth since the previous update of publisher bandwidth for the swarm. It continues this allocation strategy until it uses all its seeder bandwidth.

AntFarm controller measures the average download rates of the swarm to compute the "response curve" for the swarm. Using periodic measurements, it obtains a set of tuples of (publisher bandwidth, average download rate of the swarm). It fits a piecewise linear model to this data using the least squares method. In addition, it assumes that the curve is concave and the maximum value possible of the download rate for for $0 \le x \le \mu$ is μ. The fitted model obeys these constraints. The bandwidth allocation for each swarm is computed using a gradient ascent algorithm using these curves. It starts with a minimum bandwidth of δ to each swarm and then allocates one additional unit to the swarm which has the maximum value of λ times the current slope of the curve.

Further, the publisher bandwidth is periodically perturbed by δ to get more data points for the response curve. The algorithms first perturbs the publisher bandwidth in a time interval t1 and recomputes the response curve and a bandwidth allocation for each swarm in the next time intervals t1. It alternates between the two thereby constantly updating the response curve based on new measured data points.

The value of the parameters for this controller are: t0 = 100s, Δ = 20 KBps, t1 = 200s, δ = 10 KBps.

AIAD: This online controller aims to optimize the MIN_COST objective while keeping the download rate higher than the target download rate. AIAD controller starts with an initial bandwidth $x0 = \mu$ to keep above the target download rate. Clearly, this is based on the assumption that the target download rate is less than the peer upload capacity. The controller assigns a bandwidth $x0$ for time interval $t0$. From now on, it runs the AIAD algorithm. If the current download rate of the swarm is more than the target download rate, it decreases the publisher bandwidth by Δ and vice versa. It updates the publisher bandwidth once every $t1$ seconds. The values of the parameters we use are as follows: $t0$ = 500s, $t1$ = 200s, Δ = 10KBps .

5. EVALUATION

5.1 Experimental setup

We used PlanetLab as our testbed and selected 250 nodes for our experiments. Each PlanetLab node was installed with an instrumented BitTorrent client. Two servers running at our university acted as the seeder and the tracker respectively. We placed a limit on the total upload capacity at the seeder. The controller algorithm running at the seeder determined the bandwidth allocated to each swarm. The seeder collected the current download rate reported by all active peers every 20 seconds. The average download rates of all peers is used by dynamic controller strategies to estimate the current swarm performance.

We selected a workload consisting of 8 swarms. The coverage (equal to $\frac{\lambda S}{\mu}$) for these swarms 1, 2, 6, 8, 12, 14, 20 and 40 which approximately follow a Zipf distribution with exponent 1.5. Each swarm distributed a file of size 10 MB. The mean, minimum and maximum upload capacities of peers were 100 KBps, 40 KBps and 200 KBps respectively. We use a distribution of upload capacity of BitTorrent peers reported in [27], which was truncated to remove the high capacity peers in view of the bandwidth limit imposed on PlanetLab nodes. The total seeder bandwidth for each experiment was 400KBps. Each experiment was run for a duration of 4000 seconds.

5.2 Controller Strategies

We compare the CheatSheet system against the controller strategies described in Section 4.4. Further, we include in our comparison two simple static control strategies which serve as the baseline for comparison. These simple controller strategies are (1) EqualSplit and (2) PropSplit. EqualSplit splits the total publisher bandwidth equally among all swarms while PropSplit splits its bandwidth proportional to the healthy swarm coverage ($\frac{\lambda S}{\mu}$).

5.3 Results

We present a comparison of controller strategies for the objectives MAX_MIN, MAX_AVG and MIN_COST described in Section 4.1.

5.3.1 *Max-min download rate*

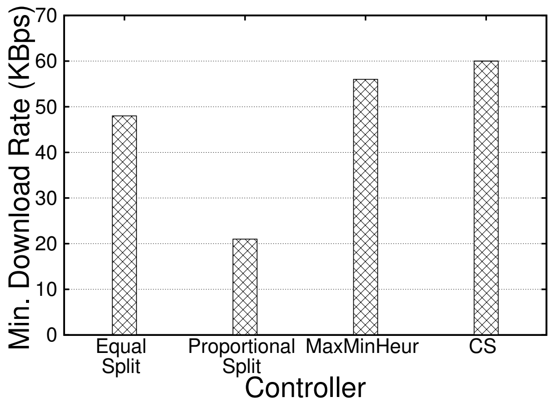


Figure 9: Minimum download rates for controller strategies

In Figure 9, we show the minimum download rates among all swarms for different controller strategies. CheatSheet has the highest minimum download rate among other controller strategies which demonstrates the efficiency of our approach. MaxMinHeur also works well for this problem and achieves a download rate within 10% of CheatSheet. Equal split and proportional split do not perform as well since they fail to allocate higher than the average bandwidth to the less popular swarms, unlike CheatSheet and MaxMinHeur. For our workload, S and μ are same for all files. Therefore, PropSplit splits bandwidth proportional to the λ for the swarm. As expected, it is especially worse since it allocates the least bandwidth to the swarm with the least popularity and therefore it has the lowest download rate.

The reason why CheatSheet performs better than MaxMinHeur is because MaxMinHeur can assign a lower than the optimal MAX_MIN publisher bandwidth to a

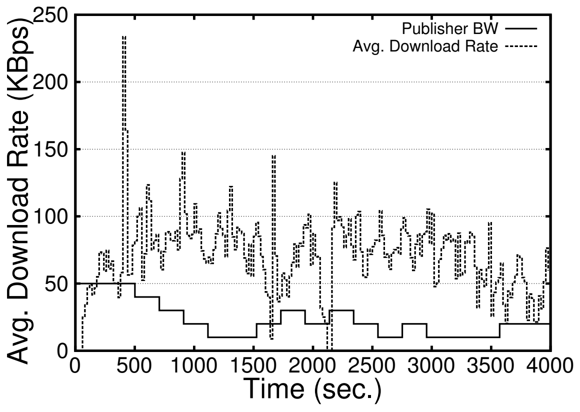


Figure 10: Publisher bandwidth and average download rate during the experiment

swarm due to constant reallocation. It waits for the swarms' download rate to decrease before it again increases the allocated bandwidth to the swarm, thereby hurting the average download rate for the swarm. Let us analyze a specific swarm where the MaxMinHeur has the lowest download rate. In Figure 10, we show the publisher bandwidth and and the average download rate for the experiment where MaxMinHeur had the least download rate. In this case, MaxMinHeur reduces the publisher bandwidth to 10 KBps for this swarm at 3000 seconds. The average download rate for the swarm decreases significantly a few hundred seconds after the publisher bandwidth is reduced. Again, the controller reacts to the change to increase the publisher bandwidth. Figure 10 also shows that the average download rate data show a lot of deviation from the mean value. Due to the reason, the publisher bandwidth cannot be adjusted too quickly based on an increase and decrease in the average download rate measured.

5.3.2 Avg download rate

We compared CheatSheet against EqualSplit PropSplit and AntFarm scheme.

The difference in performance among CheatSheet and our baseline controller strategies is less significant in this case. The average download rate of all peers depends heavily on the performance of larger swarms. Larger swarms require lesser publisher bandwidth to achieve a healthy swarm download rate. Therefore all schemes achieve nearly the same average download rate.

5.3.3 Target download rate

We compare the MIN_COST objective for 5 swarms with $\lambda = 1/7.14, 1/8.33, 1/10, 1/12.5, 1/100$ respectively. We show the comparison of CheatSheet and AIAD for a target download rate of 40 KBps in Figure 11 and for a target of 70 KBps Figure 12. The presented result is for a single run of experiment on PlanetLab. In Figure 11(a) we observe that CheatSheet perfoms below the target download rate for all swarms except $\lambda = 1/8.33$. We observe in Figure 12(a) that, unlike AIAD, CheatSheet consistently achieves a download rate within 5% of the target value for a target of 70KBps. For $\lambda = 1/100$, it achieves a bandwidth savings of nearly 50% over the AIAD controller.

We attribute this inaccuracy in the model prediction to the higher variance in the download rate value for smaller values of publisher bandwidth. A possible solution to this problem could be taking into account the variance observed in the measured data while predicting the download rate.

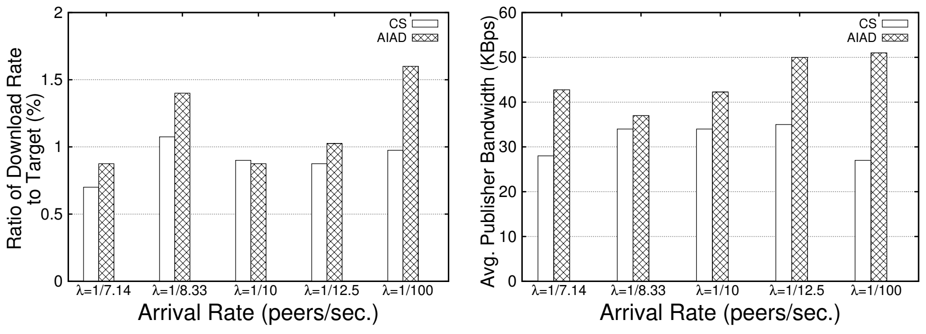
Figure 12(a) also illustrates that AIAD is not a good solution for the the problem of achieving a target bandwidth for a swarm. For $\lambda = 7.14$, the AIAD suffers from the same problem as illustrated in Figure 10. The AIAD controller first decreases the publisher bandwidth and then reacts to the reduced download rate. The design of an online controller for this objective appears a hard problem to us.

6. RELATED WORK

In 2005, Bram Cohen pointed out that "making seeding optimizations for enterprise use" was one of the "puzzles" to be solved to improve the BitTorrent protocol [11]. Interestingly, despite the growing interest on the use of peer-to-peer swarming in the enterprise domain (due to cost reductions in bandwidth [6, 31, 35] and energy [19]) and the popularity of server-assisted peer-to-peer systems (such as Shark [4], Coral [13] and Coblitz [25]) the literature on the enterprise use of peer-to-peer systems is still scarce and is comprised mainly of white papers [1, 17, 2] (some notable exceptions being [26, 9, 16, 37]).

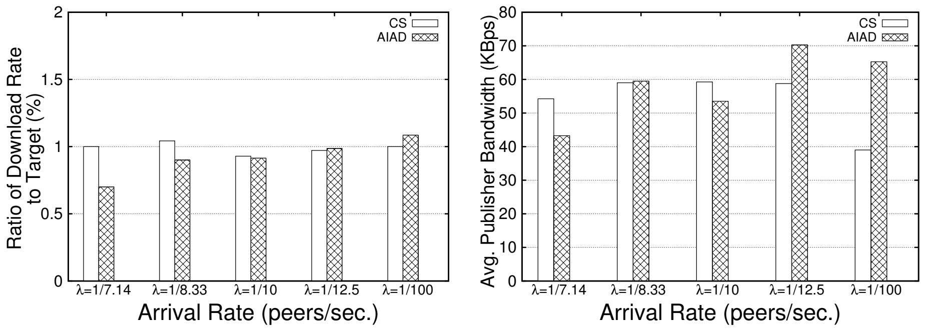
The literature on allocation strategies for server-assisted peer-to-peer content delivery encompasses measurement oriented works [26, 9] as well as analytical modeling efforts [12, 30, 37, 33]. To the best of our knowledge, this paper is the first to bridge the gap between these two approaches. The methodology adopted in measurement-oriented works [26, 9] consists of probing the swarms to infer the gains of modifying the bandwidth allocation. The model-driven efforts [12, 30, 37, 33], in contrast, usually consist of defining an optimization problem to be solved by the publishers and then showing how different system parameters affect the optimal bandwidth allocation strategy. The approach we take in this paper is a mix of these two. We couple off-line measurements and insights on how different system parameters affect bandwidth allocation with on-line probing in order to split the server bandwidth across multiple swarms.

Ioannidis et al. [16] study how quickly the bandwidth available at the server has to grow as the number of users increases. Hajek and Zhou [14] address a question



(a) (b)

Figure 11: Comparison of CheatSheet and AIAD Controller strategies: Target download rate = 40 KBps



(a) (b)

Figure 12: Comparison of CheatSheet and AIAD Controller strategies: Target download rate = 70 KBps

that, in nature, resembles the one in [16], and show conditions under which the server-assisted peer-to-peer system is stable, i.e., the population size remains bounded as a function of time. In this work, we assume that the system under consideration is always under a stable regime. In particular, we empirically observed that stability issues did not affect our results for the parameters and time spans considered in this paper.

Many recent works have studied how collaborations across swarms together with server capacity allocation can improve the overall system performance [22, 7, 36]. The key idea consists of incentivizing peers to exchange content that they did not request, which yields decreased mean download times due to availability gains. These mechanisms inevitably require modifications to the BitTorrent clients as well as strategies to enable transactions across swarms. In this paper, we showed that simple server bandwidth allocation strategies in many case suffice in order to achieve the desired performance goals, requiring no change to existing BitTorrent clients. This makes our proposed strategies suitable to deployment in systems that already natively support BitTorrent such as the Amazon S3 Cloud [3].

The seeding strategy encompasses both inter-swarm bandwidth splitting mechanisms as well as intra-swarm allocation of capacity to peers. In this paper we focused on the former, relying on the intra-swarm strategies prebuilt in the mainline BitTorrent. The key element of the intra-swarm seeding strategy implemented in the mainline BitTorrent is referred to as *super seeding* [15], which attempts to minimize the amount of data uploaded by a seed. When peers arrive to an under-populated swarm, the seed claims to have no pieces of the content. As peers connect to it, it informs that it received a new piece and allows a peer to download it. The seed does not upload any other piece to a peer until other peers advertise that they received that piece. Super seeding, as well as other works that proposed alternative intra-swarm seeding strategies [5, 20, 9], are complementary to ours. While [15, 5, 20, 9] focuses on incentive mechanisms to enforce that peers contribute to the system, in this paper our concern is with performance under the assumption that clients do not misbehave.

In this paper we proposed mechanisms for server capacity allocation for the transmission of stored content in the presence of trackers. The order at which packets are delivered, as well as strict delivery deadlines, are not relevant. Future work consists of extending the ideas presented in this paper to real-time peer-to-peer systems, whose performance was analyzed only in the context of isolated swarms [24, 34] (one exception being [32]). Extending our work to trackerless systems, which rely on random contacts between peers for content dissemination [8, 29], is also subject of future work.

7. CONCLUSIONS

We presented the design and implementation of CheatSheet, a system for allocating capacity to competing swarms in managed swarming environments. The key insight underlying CheatSheet is a concise, measurement-based predictive model of swarm performance as a function of the server bandwidth supplied to it. The model lets CheatSheet optimize the allocation in an offline manner enabling it to converge in a single round. Thus, CheatSheet is also simpler to implement than approaches based on perturbing the allocated capacities and measuring performance feedback in an online manner. We have implemented a prototype of CheatSheet based on BitTorrent that is protocol-compliant and requires no changes to existing BitTorrent clients. Our evaluation of CheatSheet over PlanetLab shows that CheatSheet achieves significant gains in swarm performance and server bandwidth cost compared to BitTorent and converges much faster than capacity allocation based on online control.

8. REFERENCES

[1] About.com. Enterprise technology: Peer-to-peer gets down to businees, 2005. `http://pcworld.about.com/magazine`.

[2] Akamai. Akamai goes peer to peer, 2007. `www.forbes.com`.

[3] Amazon. Using BitTorrent with Amazon S3. http://aws.amazon.com/.

[4] Siddhartha Annapureddy, Michael J. Freedman, and David Mazieres. Shark: scaling file servers via cooperative caching. In *NSDI*, 2005.

[5] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Some observations on Bittorrent performance. In *SIGMETRICS*, 2005.

[6] Blizzard. Blizzard wow uses p2p for downloading, 2010. `http://us.battle.net/wow/en/`.

[7] N. Carlsson, D. Eager, and A. Mahanti. Using torrent inflation to efficiently serve the long tail in peer-assisted content delivery systems. In *Networking*, 2010.

[8] Augustin Chaintreau, Jean-Yves Le Boudec, and Nikodin Ristanovic. The age of gossip: spatial mean field regime. In *SIGMETRICS*, 2009.

[9] Alix Chow, Leana Golubchik, and Vishal Misra. Improving bittorrent: A simple approach. In *IPTPS'08*, 2008.

[10] Bram Cohen. Incentives build robustness in Bittorrent. In *P2PEcon*, 2003.

[11] Bram Cohen. Interview with Bram Cohen, the inventor of BitTorrent, 2005. `http://torrentfreak.com/`.

[12] S. Das, S. Tewari, and L. Kleinrock. The case for servers in a peer-to-peer world. In *ICC*, 2006.

[13] M.J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In *NSDI*, 2004.

[14] B. Hajek and J. Zhu. The missing piece syndrome in peer-to-peer communication. In *ISIT*, 2010.

[15] John Hoffman. Super seeding, 2003. `http://en.wikipedia.org/wiki/Super-seeding`.

[16] Stratis Ioannidis and Peter Marbach. On the design of hybrid peer-to-peer systems. In *SIGMETRICS'08*, 2008.

[17] Kontiki. The power of commercial peer-to-peer delivery, 2008. `www.kontiki.com/_download/The-Power-of-Commercial-P2P.pdf`.

[18] J. Kurose and K. Ross. *Computer Networking: A Top Down Approach Featuring the Internet*. Pearson Addison-Wesley, 2010.

[19] Nikolaos Laoutaris, Georgios Smaragdakis, Pablo Rodriguez, and Ravi Sundaram. Delay tolerant bulk data transfers on the internet. In *SIGMETRICS/Performance*, pages 229–238, 2009.

[20] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in Bittorrent systems. In *SIGMETRICS*, 2007.

[21] Arnaud Legout, Nikitas Liogkas, and Eddie Kohler. Rarest first and choke algorithms are enough. In *IMC*, 2006.

[22] N. Lev-tov, N. Carlsson, Zongpeng Li, C. Williamson, and Song Zhang. Dynamic file-selection policies for bundling in bittorrent-like systems. In *IWQoS*, 2010.

[23] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. Bittorrent is an auction: Analyzing and improving Bittorrent incentives. In *SIGCOMM*, 2008.

[24] L. Massoulie and A. Twigg. Rate optimal schemes for peer-to-peer live streaming. *Performance Evaluation*, 65:804–822, 2008.

[25] KyoungSoo Park and Vivek S. Pai. Scale and performance in the coblitz large-file distribution service. In *NSDI*, 2006.

[26] R. S. Peterson and E. G. Sirer. Antfarm: efficient content distribution with managed swarms. In *NSDI*, 2009.

[27] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? In *NSDI*, 2007.

[28] D. Qiu and R. Srikant. Modeling and performance analysis of Bittorrent-like peer to peer networks. In *SIGCOMM*, 2004.

[29] Joshua Reich and Augustin Chaintreau. The age of impatience: optimal replication schemes for opportunistic networks. In *CONEXT*, 2009.

[30] Ivica Rimac, Anwar Elwalid, and Sem Borst. On server dimensioning for hybrid peer-to-peer content distribution networks. In *P2P'08*, 2008.

[31] Janko Roettgers. Wikipedia is using bittorrent p2p for html5 video, 2010. `http://gigaom.com/`.

[32] Sheldon Ross. *Stochastic Processes*. Wiley, 1996.

[33] S. Shakkottai and R. Johari. Demand-aware content distribution on the internet. *IEEE/ACM Transactions on Networking*, 18, April 2010.

[34] S. Tawari and L. Kleinrock. Analytical model for bittorrent-based live video streaming. In *NIME*, 2007.

[35] Ubuntu. Download Ubuntu using BitTorrent. `http://torrent.ubuntu.com:6969/`.

[36] Y. Yang, A. Chow, and L. Golubchik. Multi-torrent: a performance study and applications. *Int. J. Advanced Media and Communication*, 4(1), 2010.

[37] B. Zhang, S. Borst, and M. Reiman. Optimal server scheduling in hybrid p2p networks. *Performance Evaluation*, 67:1259–1272, 2010.

9. APPENDIX

Practical swarms are inefficient, i.e., peers do not consistently upload at their capacities. We denote the inefficiency of a swarm by $\gamma(N(t))$ that depends on the number of peers $N(t)$ in the system. The justification for assuming γ as dependent on N is empirical: we observed experimentally that large swarms are more efficient than small swarms. Then, we have

$$y(t) \quad = \quad \gamma(N(t))\mu + \frac{x}{N(t)} \tag{11}$$

where we note that defining $\gamma(1) = 0$ implicitly captures the client-server mode. The number of peers $N(t)$ is governed by the following markov chain

$$\begin{array}{rl} 1 \rightarrow 0, & x \\ i \rightarrow i+1, & \lambda \\ (i > 1)\ i \rightarrow i-1, & \gamma(i)i\mu + x \end{array}$$

Let N denote the expected value of $N(t)$ in steady-state. Obtaining a closed-form expression for N is difficult. Little's law still holds as follows

$$N = \lambda \cdot E[\frac{S}{y(t)}]$$

where the second term in the product above denotes the mean download time of a peer.

Suppose we make the following simplistic assumptions: (1) $E[\frac{S}{y(t)}] = S/y$ where y is the mean download rate of a peer, (2) N is an integer, and (3) $y = \gamma(N)\mu + x/N$. Then,

$$N = \lambda \cdot \frac{S}{\gamma(N)\mu + x/N}$$

or

$$N\gamma(N) = \frac{\lambda S - x}{\mu}$$

Note that we have implicitly assumed above that $x < \lambda S$ as otherwise N must be zero. When $x > \lambda S$, the performance y is simply equal to the publisher bandwidth x. The system behaves like a client-server system in this case.

In the cases below, we assume $x < \lambda S$. Suppose we make the further simplifying assumption that $\gamma(N) = 1 - \frac{1}{N^\alpha}$ for some $\alpha > 0$.

Case 1.

Consider $\alpha = 1$. Then, $N = (\lambda S - x + \mu)/\mu$, which yields

$$y = \frac{\lambda S \mu}{\lambda S - x + \mu} \tag{12}$$

Note that the function is convex and increasing in this regime.

Case 2.

Consider $\alpha = 2$. Then,

$$N - \frac{1}{N} = \frac{\lambda S - x}{\mu}$$

which yields a quadratic equation for N. The corresponding performance is given by

$$y = \frac{2\lambda S \mu}{(\lambda S - x)(1 + \sqrt{(}1 + \frac{4\mu^2}{(\lambda S - x)^2}))} \tag{13}$$

Other values of α will result in similar polynomial equations that can be solved numerically. In general, this equation will have multiple solutions, but only some of them may be meaningful. The exact value of α needs to be determined empirically.

Unfortunately, models such as above do not even capture the concave behavior of y with respect to x in the regime $0 < x < \mu$. For example, it is easy to see that the slope of the curve $\frac{dy}{dx}$ in equation (12) monotonically increases with x. The reason such models fail to capture realistic swarm behavior is that they assume that the swarm reaches a steady-state where the number of peers N remains fixed. However, in practice, we observe that at smaller values of x, the number of peers fluctuates significantly never reaching a steady-state. Thus, the swarming efficiency in turn varies over time and modeling these time-varying dynamics as a function of the publisher bandwidth x is nontrivial and necessitates a thorough empirical characterization of swarm health as a first step.