A Measurement-Driven Comparison of Server Bandwidth Controllers for Client-assisted Content Delivery

Abhigyan Sharma
University of Massachusetts Amherst

Arun Venkataramani
University of Massachusetts Amherst

Antonio Rocha
Fluminense Federal University, Brazil

ABSTRACT

A key challenge in client-assisted content delivery is determining how to allocate limited server bandwidth across a large number of files being concurrently served so as to optimize global performance and cost objectives. In this paper, we present a comprehensive experimental evaluation of strategies to control server bandwidth allocation. As part of this effort, we introduce a new *model-based* control approach that relies on an accurate yet concise "cheat sheet" based on a priori offline measurement to predict swarm performance as a function of the server bandwidth and other swarm parameters. Our evaluation using a prototype system, SwarmServer, instantiating static, dynamic, and model-based controllers shows that static and dynamic controllers can both be suboptimal due to different reasons. In comparison, a model-based approach consistently outperforms both static and dynamic approaches provided it has access to detailed measurements in the regime of interest. Nevertheless, the broad applicability of a model-based approach may be limited in practice because of the overhead of developing and maintaining a comprehensive measurement-based model of swarm performance in each regime of interest.

1. INTRODUCTION

Faced with the challenge of ever-increasing demand for content, content distributors have turned to *client-assisted content delivery* in recent times. A client-assisted content delivery architecture enables content distributors to provide performance in a scalable and cost-effective manner by opportunistically leveraging client resources, especially their uplink bandwidth, to augment their managed infrastructure resources. Although client-assisted content delivery systems have their roots in peer-to-peer file sharing systems [3, 18], commercial CDNs such as Akamai, Velocix, and Octoshape [1, 28, 17] as well as live streaming services such as PPLive and Sopcast [15, 25] have warmed up to using them for mainstream enterprise content delivery in recent times.

A key problem in client-assisted content delivery is bandwidth management, i.e., determining how to allocate limited server bandwidth across a large number of files being concurrently served to clients so as to balance the performance and cost objectives of the content distributor. Unlike purely client-server systems or purely peer-to-peer systems, this problem is particular to client-assisted content delivery systems that attempt to combine the predictable performance and ease of management of the former with the scalability and cost-effectiveness of the latter. The sever bandwidth allocated to a *swarm*, or a set of clients concurrently downloading the same file, is critical in determining the effectiveness of client-to-client exchanges and by consequence client-perceived performance. Furthermore, the appropriate allocation may be counter-intuitive, e.g., a popular file requires *less* server bandwidth compared to an unpopular file, all else being equal, in order to ensure similar client-perceived performance.

Our primary contribution is a measurement-driven comparative analysis of several existing and new strategies for allocating server bandwidth in client-assisted content delivery systems. To this end, we classify these bandwidth allocation strategies, or *controllers*, into three categories. The first is *static*, a class of controllers that use simplistic strategies such as allocating bandwidth uniformly, on a best-effort basis, or proportional to the demand across files [3]. The second is *dynamic*, a class of controllers that constantly adjust the allocation in response to fine-grained client-perceived performance so as to optimize the performance or cost objectives of the content distributor [19, 20].

In this paper, we present a third, new class of controllers called *model-based* controllers that allocate server bandwidth based on a predictive model of client-perceived performance as a function of the server bandwidth and other swarm parameters such as the request arrival rate, file size, and client upload capacities. Unlike dynamic controllers that can be suboptimal due to long convergence delays while searching for an optimal allocation in situ, model-based controllers can jump to the optimal allocation in a single step by solving the underlying optimization problem "on paper" .

We have implemented a prototype system, SwarmServer, to facilitate our comparative analysis of controllers. In addition to several simple static and dy-

namic controllers, SwarmServer supports a model-based controller called CheatSheet for three bandwidth allocation objectives: minimizing the average download time, maximum download time, or the server bandwidth consumed so as to achieve a target performance objective. CheatSheet uses extensive a priori measurement to develop an accurate and concise model of performance as a function of the server bandwidth and a number of swarm parameters. To our knowledge, CheatSheet is the first attempt at developing a detailed *empirical* model of swarm performance.

Our extensive experiments with SwarmServer in conjunction with BitTorrent swarms running over 350 PlanetLab nodes reveal several insights. First, simple static controllers are hit-or-miss; while they perform well for some performance objectives and workloads, even outperforming dynamic controllers, they fall severely short on others. The suboptimal performance of static controllers is unsurprising and consistent with previous findings [19] for one our three objectives of interest. Second, model-based control is feasible and promising—CheatSheet consistently outperforms both static and dynamic controllers provided its model is based on detailed a priori measurements in an environment similar to the operational environment. CheatSheet performs up to 4× better than static schemes and up to 1.7× better than dynamic controllers.

Nevertheless, having gone through the experience of building a model-based controller, our conclusions about its practicality are somewhat mixed because of several reasons. First, it is hard. To appreciate this, consider that CheatSheet's model used in the experiments in this paper alone required over 12 days of measurement data on PlanetLab so as to account for a number of parameters such as the server bandwidth, request arrival rate, distribution of client upload capacities, file size, etc. Second, while a measurement-driven model is robust to small variations in the operational environment, significant changes require recalibrating the model. For example, we find that the model developed over PlanetLab is inaccurate when deployed on a public cloud such as Amazon EC2 or a local cluster in our department. Similarly, significant changes in the client population or behavior such as participation in multiple swarms introduce further uncertainties into the model. Thus, model-based control may be appropriate primarily for relatively predictable environments (e.g., distributing TV shows and movies to FIOS [29] customers).

The rest of the paper quantifies these nuanced pros and cons of the three classes of controllers. We begin with a background on client-assisted content delivery.

2. BACKGROUND

Client-assisted content delivery seeks to combine the best of traditional client-server and peer-to-peer swarming systems, namely, the predictable performance and ease of management of the former and the scalability and cost-effectiveness of the latter.

A client-assisted content delivery system consists of a server that acts as the primary source for all content. All clients concurrently downloading the same file are referred to as a swarm. Clients follow a common peer-to-peer protocol for downloading (uploading) the file from (to) other clients in the swarm. In this paper, we focus on the BitTorrent protocol because of its open nature and wide deployment, however our findings are qualitatively applicable to other comparable plugins offered by content distributors [1, 17]. The server is logically centralized and participates by contributing bandwidth to all swarms.

A key goal of a client-assisted content delivery system is to optimize a system-wide objective, e.g., minimize the average download time of all clients, by judiciously allocating limited server bandwidth across all swarms. To this end, a *controller* at the server collects information from all swarms and uses this information to compute and effect an allocation of server bandwidth so as to optimize the system-wide objective.

2.1 Classification of controllers

We classify existing controllers as static or dynamic, and introduce a new class called model-based controllers, as described in turn below.

Static: A static controller allocates server bandwidth using a simple heuristic while being agnostic to the system-wide performance objective and unresponsive to actual client-perceived performance. Static controllers therefore obviate any online measurement of client performance. We analyze the following static controllers in this study: (1) best-effort, or using BitTorrent as-is by repurposing a common seeder across all swarms as the server; (2) equal-split, or splitting server bandwidth equally across all active swarms; (3) proportional split, or allocating bandwidth proportional to the arrival rate within a swarm.

Dynamic: A dynamic controller continuously monitors fine-grained information about client-perceived performance for all clients in each swarm (see Figure 1), and accordingly adjusts the bandwidth allocation in each monitoring epoch. An example of a dynamic controller is AntFarm [19] that monitors the number of blocks uploaded and downloaded by each client in each epoch and uses a strategy based on perturbation and gradient-ascent in order to optimize the aggregate download rate across all clients across all swarms.

Model-based: A model-based controller relies on a predictive model of swarm performance as a function of the supplied server bandwidth and other swarm parameters such as the file size, the peer arrival rate, and the upload capacity distribution of peers. Unlike dy-

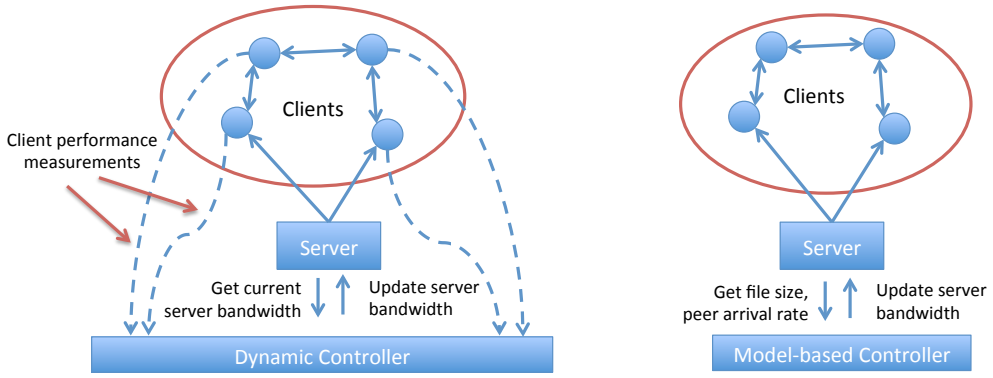


Figure 1: A comparison of dynamic and model-based control architectures.

namic controllers, a predictive model obviates explicit measurement of client-perceived performance, requiring only parameters that are already available or easily inferred at the server. More importantly, it obviates in situ perturbation and gradual adjustment of the allocation enabling the controller to jump to the optimal allocation in a single step by using the model to solve the underlying optimization problem "on paper". Thus, a model-based controller can quickly adapt to sudden changes in request arrival rates.

2.2 Limitations of dynamic control

Our motivation for investigating model-based control stems from the limitations of dynamic controllers in realistic environments. Unlike static controllers that are but naive baseline strategies, the limitations of dynamic control are less obvious and are described next.

Convergence time: Dynamic control works in a feedback-driven manner by perturbing the current allocation, monitoring the performance impact of the perturbation, and accordingly determining the next perturbation. This approach is prone to prohibitively long convergence delays, especially with hundreds or thousands of swarms and optimization objectives that do not lend themselves well to greedy heuristics. Even with a small number of swarms and a greedy gradient ascent strategy, the convergence time can be on the order of thousands of seconds. This is because the effect of a perturbed allocation can take several minutes to propagate through the swarm so as to be observable by the controller, and adjusting the allocation more frequently is likely to be unproductive or even detrimental.

As an example, AntFarm updates its bandwidth once every 300 seconds by 5KB/s, so an adjustment of 50KBps requires nearly an hour to take effect. If peer arrival rates change significantly during the course of a typical convergence period, the controller can perpetually be in a state of suboptimal allocation. Even with predictable arrival rates, a dynamic controller needs to be carefully designed for each performance objective or risk perpetually oscillating about the optimal allocation. In comparison, with an accurate model, a model-based controller can potentially estimate the optimal allocation in a single epoch of a shorter duration that is required only to obtain a reliable estimate of the arrival rate.

Measurement overhead and error: The performance of any controller in steady state depends on how accurately it can estimate the relation between server bandwidth and swarm performance. Dynamic controllers could be inaccurate because they measure swarm performance for the current bandwidth allocation only for a single measurement interval of a few hundred seconds. Even with a fixed server bandwidth, the number of peers joining the swarm and the upload capacities of peers in the swarm can differ from one measurement interval to the next. Thus, dynamic controllers have to choose between the high overhead of fine-grained measurement and the inaccuracy of infrequent or sampled measurements.

These limitations of dynamic controllers compel us to explore model-based controllers. Our hope is that the measurement overhead could be relegated to an a priori offline phase to develop an accurate model of swarm performance in exchange for increased responsiveness in the operational phase. The challenge, of course, is to develop an accurate model of swarm performance with a tractable measurement overhead and small representation size, a challenge we address next.

3. A MEASUREMENT-BASED MODEL

In this section, we develop a measurement-based model of swarm performance, which is the key building block for a model-based controller. Unlike prior theoretical models [22, 6, 14] that over-simplify swarm behavior, our work, to our knowledge, is the first effort at developing a measurement-based model of swarm performance.

3.1 Goal and model assumptions

We start with the following question: *what is the average download time of peers in a BitTorrent swarm when given a certain amount of server bandwidth*? The answer to this question of course depends on several

characteristics of the swarm such as the arrival and departure patterns of peers, their upload and download capacities, the size of the file being distributed, etc. The answer also depends on design parameters of BitTorrent clients such as the number of active peers to which a peer concurrently uploads and how it splits its upload capacity across them, the length of an optimistic unchoke round, the size of chunks, etc. Finally, network conditions and artifacts of the transport protocol (TCP or custom transport protocols such as μTP for non-interfering downloads [27]) will also impact swarm performance. Clearly, a model attempting to account for all of the factors affecting a swarm's performance quickly becomes intractable.

In our quest for a model that simple and useful in practice, we make several simplifying assumptions that abstract away less important details of the system. To this end, consider a swarm distributing a file of size S to peers arriving at a rate λ. The upload capacities of arriving peers are drawn from a distribution with mean μ. The download capacity of peers is unlimited. Peers depart immediately after finishing their download (so the departure rate of peers is equal to the arrival rate λ in steady state). Let x denote the (fixed) bandwidth supplied by the server. Our model postulates that the average download time of peers, τ, can be determined as a function of x, μ, λ and S. We state this dependence as

$$\frac{S}{\tau} = f(x, \mu, \lambda, S) \quad (1)$$

The dependence between $f(.)$ and τ is stated in this seemingly convoluted manner because it is convenient to refer to $f(.)$ as "swarm performance", i.e., the higher, the better.

By assuming that τ is determined by the above four parameters alone, the model implicitly makes a few assumptions. The model assumes that network loss rates and round-trip times are not so high that they reduce the effective average peer upload capacity (or equivalently that μ already incorporates these effects). It also implicitly assumes that all peers use a standard BitTorrent client and that implementation variations across operating systems are minor. It further assumes that μ already incorporates the effect of user-specific configurations that limit their upload contribution. Finally, the model assumes that despite all these heterogeneous factors affecting the *distribution* of peer upload capacities in practice, this distribution is stationary, so the *average* upload capacity μ (in conjunction with the other three parameters) is sufficient to determine the average download time.

3.2 Measurement-based model

We take an empirical, measurement-driven approach to capture the relationship in Equation (1). A naive approach to this end would be to "measure" the relationship posed in Equation (1) for all foreseeable values of the four underlying dimensions (x, μ, λ, S), which is impractical. Instead, our approach is to summarize the relationship using a small number of measured scenarios and use simple interpolation to estimate the unmeasured scenarios. We begin with a description of our measurement setup.

3.2.1 Measurement setup

We use PlanetLab for running private swarms in order to obtain measurement data as follows. Two nodes act as the server and the tracker respectively, while the rest of the nodes act as peers and run an instrumented BitTorrent client [11]. In each swarm run, peers arrive over time to download the file and depart immediately after completing the download. Peer inter-arrival times follow an exponential distribution with mean $1/\lambda$. Each swarm is run long enough so that the download times of peers stabilize, and the server records the average download time of peers that have completed downloads at the end of the experiment. Each swarm run is repeated five times with a fixed set of swarm parameters (x, μ, λ, S) and different runs vary these parameters.

Figure 2 shows the aggregate results of our measurement experiments. In this figure, each point corresponds to a swarm run averaged over five repetitions as described above. We use the upload capacity distribution of peers reported in [21], which was scaled and truncated to remove very high capacity peers so as to accommodate the daily limit on the maximum data transfer imposed on PlanetLab nodes. The resulting average upload capacity (μ) was 100KBps with upload bandwidths in the range of 40-200 KBps for individual peers. No restrictions were imposed on the maximum download rate of any client. The file size is fixed at S=10MB. Each line corresponds to a fixed arrival rate λ as shown, and plots the mean download rate for different values of the server bandwidth x that is varied from 0 to 100KBps (also the average peer upload capacity) in 10KBps increments. With these parameters, a swarm run takes between 2000 to 5000 seconds, so the total running time to generate this figure is over 10 days (5 runs per point $\times$ 50 points $\times$ an hour roughly per run = 250 hours).

3.2.2 Swarm performance vs. server bandwidth

Figure 2 presents several insights about how the swarm performance depends on server bandwidth and peer arrival rate. First, swarm performance as expected increases with server bandwidth keeping all else fixed, as can be seen from the increasing trend of all lines. Second, swarm performance is concave with respect to server bandwidth. This is because, when the server bandwidth is very low, it becomes the bottleneck pre-

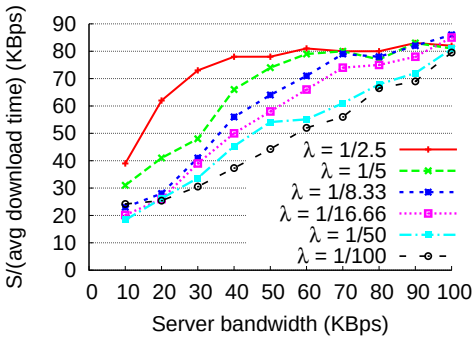


Figure 2: Dependence of swarm performance on server bandwidth, and peer arrival rate λ for S = 10MB and μ = 100KBps. Unit of λ = sec^{-1}.

venting peers from efficiently utilizing their upload capacity for exchanging blocks. In this regime, increasing server bandwidth slightly improves the efficiency of P2P exchanges and therefore performance significantly. At high values of server bandwidth, there is less room for improving the efficiency of P2P exchanges, so the server's bandwidth improves performance similar to traditional client-server systems, i.e., the bandwidth is divided across extant peers. When the server bandwidth equals the average peer upload capacity we find that a swarm's utilization of P2P bandwidth is about as efficient as it can be, and any additional server bandwidth is simply used as in a client-server system. As a result, the swarm performance in the regime $x > \mu$ (not shown in Figure 2) can be easily derived analytically obviating time-consuming measurements.

Third, in the regime $x \leq \mu$ shown in the figure, swarm performance improves with the arrival rate (keeping all else fixed). The lines corresponding to increasing λ increasingly appear pulled towards the top-left. At very low arrival rates, e.g., λ=1/100/s, the swarm essentially behaves like a client-server system as there is at most one peer most of the time, so the corresponding curve resembles the line $y = x$. At higher arrival rates, the swarm remains efficient (i.e., it maintains a healthy download rate of over 80KBps) for values of x much smaller than μ. This is because large swarms are mostly self-sustaining and need only a tiny amount of server bandwidth to supply missing blocks in the unlikely event that none of the extant peers possess those blocks.

3.2.3 *Model representation*

To concisely represent the model, we carefully select a small number of values of each parameter for measurements. The model captures the dependence of swarm performance on server bandwidth and peer arrival rate for a given upload capacity distribution and file size (as in Figure 2) using a small number ($\approx$ 100) of values. We take measurements for 10 values of x ranging from $\mu/10$ to μ, and for ten values of λ in a range determined by a metric we refer to as the "healthy swarm size". The healthy swarm size is the number of peers when the efficiency of P2P exchanges in maximum. The intuition for healthy swarm size comes from Little's law [10], healthy swarm size is $\lambda \times S/\mu$, as S/μ is average download time of peers in this case. When the healthy swarm size is one or less, the swarm essentially behaves like a client-server system. We empirically observe that when the healthy swarm size is 50 or more, the swarm is essentially self-sustaining, i.e., even with a server bandwidth of just a $\mu/10$, the swarm is efficient. So we take measurements for values of λ selected such that the healthy swarm size $\lambda S/\mu$ increases from 1 to 50 in 10 equal increments. The total number of combinations of x and λ is therefore 100.

We maintain a table, referred to as the "cheat sheet", that records the swarm performance for all of the above combinations. This cheat sheet is used to approximately estimate by simple linear interpolation the swarm performance for values of x and λ that are not explicitly measured.

3.2.4 *Varying file size*

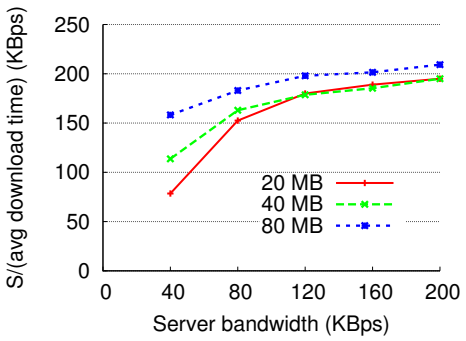


Figure 3: Dependence of swarm performance on server bandwidth for varying file size S.

The cheat sheet as described above can only be used to estimate swarm performance for a given file size. To address file size diversity, we use an interpolation approach similar to the one used for arrival rates and server bandwidth. A separate cheat sheet is stored for a small number of file sizes spanning the regime of interest, e.g., 10 file sizes in geometric progression from 1MB to 10GB, and the swarm performance for file sizes in between is estimated via interpolation.

At the onset of this work, we expected that a larger file size could be treated as equivalent to a larger arrival rate, i.e., $f(x, \mu, \lambda, kS)$ could be approximated as $f(x, \mu, k\lambda, S)$, thereby obviating the need to maintain separate cheat sheets for different file sizes. The intu-

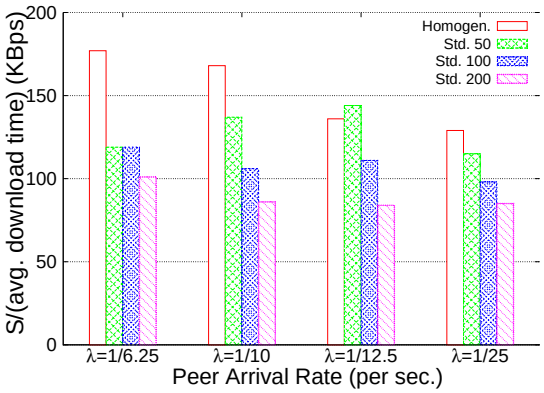


Figure 4: Swarm performance reduces as the variance of upload capacities increases.

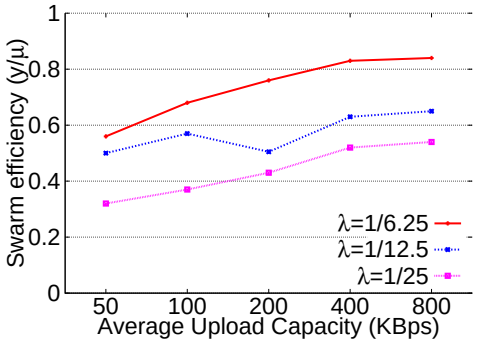


Figure 5: Swarm efficiency improves as upload capacity (μ) increases (x and S also increase proportionally).

ition for this expectation is that λS (bits/sec) represents the aggregate demand arriving into the system, so the response curve should not change significantly if the demand remains unchanged. Unfortunately, this turns out not to be the case as shown by the experiment in Figure 3. The figure plots the swarm performance as a function of the server bandwidth, and the different lines increase (decrease) S (λ) by the same factor, i.e., λS is the same for all points in the graph. The lines clearly show a slight uptrend suggesting that larger file sizes boost swarming efficiency more than larger arrival rates or, equivalently, a swarm distributing a larger file performs better than a swarm distributing a smaller file even though both have the same aggregate demand, client upload capacities, and publisher bandwidth.

3.2.5 *Variation in upload capacity distribution*

There are two kinds of variations that occur in peer upload capacities. First, the upload capacity distribution of any sample of peers currently participating in a swarm may differ from the overall distribution. Our model implicitly accounts for this statistical variation because peer upload capacities during measurements are chosen by randomly sampling the distribution. Second, the overall upload capacity distribution of peers visiting the site can change. However, we expect that upload capacity distribution is unlikely to change at short time scales, as it depends on technology trends and the population of users who visit the site, which is likely to remain stable over the course of several months.

The changes in the upload capacity distribution at time scales of several months can be addressed by updating the cheat sheet with new measurements. Next, we present two experiments in which we vary the upload capacity distribution. These experiments provide evidence that new measurements are needed if the upload capacity distribution changes significantly, and also show how swarm performance is affected on changing the upload capacity distribution. Both experiments were performed on a local cluster to reduce the variability due to available upload capacity on PlanetLab nodes.

The first experiment shows that for the same mean upload capacity, increasing the variance reduces the swarm performance. For this experiment, the mean upload capacity fixed at 200 KBps, file size is set to 20 MB and the server bandwidth is set to 60 KBps. In Figure 4, four bars are shown for each arrival rate. The first bar is for a homogenous upload capacity and upload capacities for the other three bars are sampled from a normal distribution with specified mean and variance. As the variance of upload capacities increases, swarm performance reduces significantly, e.g. for $\lambda = 1/10$/s, upload capacity reduces by half.

The second experiment shows that as mean upload capacity increases (keeping variance the same), swarm efficiency (the ratio of swarm performance to mean upload capacity) also increases. We experiment with mean upload capacities in the range 50 KBps to 400 KBps. Variance is always zero as we assign equal upload capacity to all peers. The server bandwidth is set to 30% of the upload capacity. To match the aggregate demand (λS) with the upload capacity, file size (S) is increased in proportion to the upload capacity. Figure 5 shows that for all arrival rates, an increase in upload capacity (and a proportional increase in server bandwidth and file size) also improves swarm efficiency.

3.2.6 *Effect of measurement testbed*

The measurement-based model requires network conditions to remain relatively similar to the environment in which the model's measurements were obtained. We repeated the experiment shown in Figure 2 on two other testbeds - Amazon EC2 [5], and a local cluster. For the EC2 experiment, we select equal number of machines from five geographic locations to differentiate the EC2 testbed from the local cluster which has microsecond round trip latencies. In Figure 6, we compared the

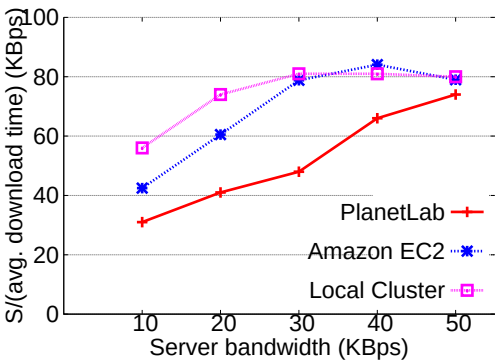


Figure 6: Swarm performance varies as experimental testbed changes. Swarm performance is better on Amazon EC2 and on a local cluster compared to Planetlab. $\lambda = 1/5$/sec.

swarm performance on the three testbeds for a peer arrival rate of $\lambda = 1/5$/s. Swarm performance on EC2 is up to 30 KBps higher than on PlanetLab. Experiments on the local cluster show even better swarm performance than on EC2. Thus, the measurement-based model is most useful when it is feasible to conduct measurements in regimes similar to those encountered in deployment.

Swarm performance differs on the three testbeds as their *effective* upload capacities are different. The round-trip times in the local cluster are much smaller than in PlanetLab which reflects in the form of higher effective upload capacities and better performance. EC2 only has a small extent of geographic diversity (five different locations), so neighbor relationships between peers in the same data center tend to dominate (a clustering effect that has also been alluded to by prior work [11]). This clustering effect again results in the form of EC2 nodes having higher effective upload capacities.

3.2.7 *Summary and limitations*

Although the measurement-based model can capture the dependence on key swarm parameters such as server bandwidth, peer arrival rate, file size, and upload capacity distribution, it still has several limitations. The most critical limitation is the extensive measurement needed to build a cheat sheet. The second limitation is that an accurate estimate of upload capacity distribution may not be available, e.g., due to peers downloading files from multiple swarms simultaneously. The third limitation of the model is that it assumes each peer arrives to download a file and departs only when its download is complete. Prior studies indicate that users may abort the download before completion and return later to resume a download [26, 7]. Therefore, the model also needs to account for peer arrivals and departures in the middle of a download.

4. SwarmServer SYSTEM

In this section, we present an implemented prototype of our system, SwarmServer, to compare different controller strategies. We begin with a brief description of our implementation and the content distribution objectives that we use for our comparison. Then, we discuss the design of model-based, dynamic, and static controllers implemented in SwarmServer.

4.1 Implementation

SwarmServer system is implemented in Python and consists of approximately 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 the instrumented BitTorrent client developed by Legout et al. [12], which we modified to enable us to change the maximum upload bandwidth of the client without restarting it.

4.2 Content distribution objectives

We compare controller strategies on three content distribution objectives.

- **MIN_AVG**: Minimize the average download time across all peers in all swarms for a given total server bandwidth.
- **MIN_MAX**: Minimize the maximum value of the average download time across swarms for a given total server bandwidth.
- **MIN_COST**: Minimize the total server bandwidth while achieving a set of specified target download times for each swarm.

4.3 Model-based controller

We call the model-based controller CheatSheet. The controller works without continuous measurement of client performance. Instead, it determines server bandwidth allocation by solving an optimization problem based on the measurement-based model developed in §3.

Next, we describe the optimization formulations used by CheatSheet to calculate bandwidth allocation for the objectives introduced in Section 4.2. Below, we assume that there are a total of k swarms and the average upload capacity, arrival rate, and file size of the i'th swarm $1 \leq i \leq k$ are given by λ_i, μ_i, S_i respectively. The goal is to determine server bandwidth allocations $\{x_i\}_{1 \leq i \leq k}$ so as to optimize the desired objective.

Optimization formulation for MIN_AVG:

$$\min \sum_{1 \leq i \leq k} \lambda_i \tau_i \qquad (2)$$

subject to

$$\tau_i = S_i/f(x_i, \mu_i, \lambda_i, S_i), \quad 1 \leq i \leq k \quad (3)$$

$$\sum_{1 \leq i \leq k} x_i \leq X \quad (4)$$

The first constraint (3) above simply rephrases Equation (1) relating the average download time τ to the server bandwidth x and other parameters of the swarm. The second constraint above limits the total bandwidth the server has at its disposal to allocate to the k swarms.

CheatSheet uses its measured knowledge of $f(.)$ to solve this optimization problem. If $f(.)$ is known to be smooth and concave in x, MIN_AVG can be solved using a greedy gradient-ascent strategy that computes a unique, optimal solution as follows: (1) Start with $x_1 = x_2 = \cdots = x_k = \Delta$ for a small Δ; (2) Allocate the next Δ units of capacity (divided equally) to the swarm(s) with the largest value(s) of the gradient $\lambda_i f^{'}(x_i, \mu_i, \lambda_i, S_i)$; (3) If not all X units of capacity have been allocated, goto (2). Else terminate.

If $f(.)$ is piecewise linear and concave, the above strategy still works, but the resulting solution may not be unique. In order to be able to arrive at a unique optimal solution, CheatSheet cleans the measured $f(.)$ by fitting smooth and concave polynomial curves for each line in Figure 2. We assume that this data cleaning has been already performed while describing the solutions to the next two objectives as well.

Optimization formulation for MIN_MAX:

$$\min(\max_{1 \leq i \leq k}(\tau_i)) \quad (5)$$

subject to the same constraints as (3) and (4) above.

If $f(.)$ monotonically increases with x, MIN_MAX can be solved optimally using a simple greedy heuristic. For a rate y, let $x = f^{-1}(y, \mu, \lambda, S)$ denote the server bandwidth x required to achieve an average download time of S/y. The heuristic is as follows: (1) Initialize $y = \Delta$ for a small Δ; (2) Set $x_i = f^{-1}(y, \mu_i, \lambda_i, S_i)$, $1 \leq i \leq k$; (3) If $(\sum_i x_i < X)$, increment y to $y + \Delta$ and goto (2). Else, terminate.

The above algorithm starts with a small target mean download rate (or large target average download time) and checks to see if an allocation that achieves that target is feasible. If so, it sets the allocation accordingly and increases the target. If not, the most recent allocation minimizes the maximum value of the average download time across the k swarms.

Optimization formulation for MIN_COST:

$$\min(x_1 + \cdots + x_k) \quad (6)$$

subject to

$$\tau_i = S/f(x_i, \mu_i, \lambda_i, S_i), \quad 1 \leq i \leq k \quad (7)$$

If $f(.)$ is invertible, then MIN_COST can be solved by setting $x_i = f^{-1}(S/\tau_i, \mu, \lambda, S)$.

4.4 Dynamic controller

SwarmServer implements three dynamic controllers AIAD, Leveler, and AntFarm, which optimize MIN_COST, MIN_MAX, and MIN_AVG objectives respectively.

4.4.1 AIAD

AIAD optimizes the MIN_COST objective. AIAD is extremely simple and works as follows. Suppose the target average download time of the swarm is τ and the file size is S. The controller initializes the server bandwidth x to S/τ. Once every epoch, it measures the average download rate, y, of peers in the swarm. If $S/\tau > y$, it increases the server bandwidth x by Δ. Otherwise it decreases x by Δ, except in the case that the decrement would cause x to dip below a minimum bandwidth threshold. In our implementation, the epoch length is set to 200 seconds, Δ is set to 10 KBps, and the minimum bandwidth threshold is set to 5 Kbps.

4.4.2 Leveler

Leveler optimizes the MIN_MAX objective. The controller starts with an equal split of server bandwidth among all swarms. Once every epoch, the controller measures the average download rate of all swarms. The server bandwidth is increased by a small, fixed Δ for swarms whose download rate is lower than the median of average download rates. Similarly, the controller reduces the server bandwidth by Δ for each swarm with average download rate higher than the median value. Similar to AIAD, the controller never reduces the server bandwidth allocated to a swarm below a minimum threshold. Epoch length, Δ, and the minimum bandwidth are the same as in AIAD.

4.4.3 AntFarm

AntFarm optimizes the MIN_AVG objective. This controller is similar to that used in [19], which has the same name.

At the start, AntFarm assigns the total server bandwidth, X, to swarms in small increments. Initially, each swarm is assigned a small bandwidth Δ_1. In each time epoch, it increments the server bandwidth by Δ_2 for the swarm that shows the maximum value of the following term: (increase in average download rate since the previous update of server bandwidth) $\times$ (peer arrival rate). We use $\Delta_1 = 5$ KBps, $\Delta_2 = 10$ KBps, and epoch length = 200 s in our implementation.

The bandwidth allocation in steady state is computed using "response curves" for each swarm. The response curve, $y = f(x)$, where y is the average download rate, x is the server bandwidth. Let λ_i denote the arrival rate of the i-th swarm. Given the response curves for a set of swarms, a gradient ascent algorithm calculates the

bandwidth allocation to swarms as follows: (1) Start with $x_1 = x_2 = \cdots = x_k = \delta$ for a small δ; (2) Allocate the next δ units of capacity (divided equally) to the swarm(s) with the largest value(s) of the $\lambda_i f(x_i + \delta) - f(x_i)$; (3) If all X units of capacity have been allocated, terminate. Else goto (2).

AntFarm builds response curves for the mean download rate vs. server bandwidth. To this end, the server obtains periodic measurements of the average download rate by perturbing the server bandwidth by Δ_3 once every epoch and fits a piecewise linear function that minimizes the least square error to fit the measured points. Each perturbation of the server bandwidth is used to refresh the response curve and recompute bandwidth allocations as in the above paragraph. The value of bandwidth perturbation, Δ_3, is set to 5 KBps and epoch length of perturbation is set to 200 seconds.

4.5 Static controller

We implement the following static controllers in SwarmServer: *BitTorrent, EqualSplit, PropSplit.* BitTorrent sets an upload limit at the server for a set of swarms but does not set a per-swarm limit. The server bandwidth to each swarm by the server is determined by the number of peers connected to the server. EqualSplit splits the available server bandwidth equally among all swarms. PropSplit splits total server bandwidth proportional to the peer arrival rate for each swarm.

5. EVALUATION

Our comparison of controller strategies, presented in this section, answers two main questions: (1) Do dynamic and model-based controllers improve performance over static controllers? If yes, then by how much? (2) Which type of controller, dynamic or model-based, performs better for the objectives in §4.2? Our experiments show that model-based controller outperforms dynamic controllers on all three objectives we compared. Static controllers cannot equal a model-based controller either; they perform well on some workloads and objectives but fare poorly on others.

5.1 Experimental setup

We performed our evaluation using about 350 PlanetLab nodes for our experiments. In addition, a server and a tracker for all swarms were hosted on two machines at our university. We used an instrumented BitTorrent client used in [12]. Peer inter-arrival times as well as peer upload and download capacities are the same we used in our measurements. Since multiple BitTorrent clients may be running on the same PlanetLab node, the maximum upload limit was imposed individually on each client instead of each PlanetLab node.

5.2 Average download time

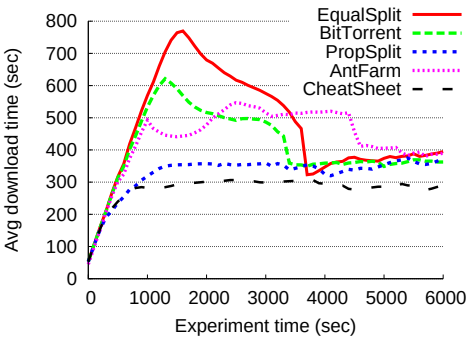


Figure 7: Average time spent by peers in all the swarms for Zipf workload. Simple approaches (e.g., Equal Spit) and online controllers (e.g., AntFarm) incur a higher download time in the initial phase as well as in steady state.

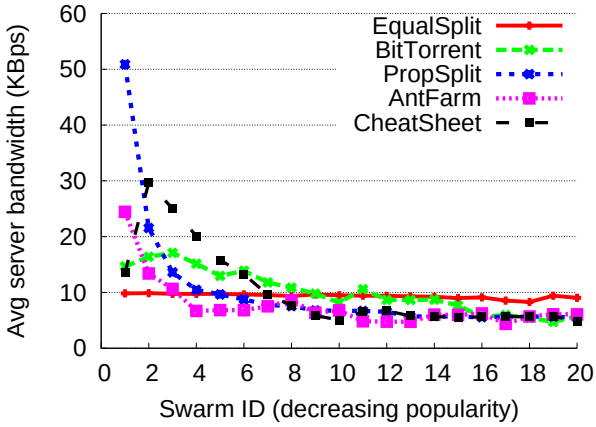


Figure 8: Server bandwidth to swarms by controllers to minimize the average download time. Swarms are ordered left to right in decreasing order of popularity.

First, we compare controllers on the MIN_AVG objective. We select a workload consisting of 20 swarms whose mean arrival rates are chosen according to a Zipf distribution with parameter 1.5. The mean arrival rates of the most popular swarm and the least popular swarm is 0.5/s and 0.0055/s respectively. Each swarm distributes a file of size 10 MB. The total server bandwidth is set to 200 KBps.

Figure 7 shows how the average download time changes over time for the different compared schemes. The average is computed using the download time of peers that completed their download within the previous 2000 sec interval as well as the resident time, i.e., the time since arrival, for peers whose downloads are under progress.

There are two main observation from the experiment in Figure 7. First, in the initial phase, EqualSplit, BitTorrent and AntFarm incur much higher average download times than PropSplit and CheatSheet, and their

average download times take considerably longer to stabilize. Second, even after all controllers have reached steady state, CheatSheet achieves a download time that continues to be lower (by at least 25%) compared to all other schemes (that perform roughly similarly in steady state in this experiment).

The explanation for these observations is as follows. EqualSplit, BitTorrent and AntFarm have a very high download time at the start of the experiment because they assign a small server bandwidth to large and small swarms alike. If the initial server bandwidth is small, a huge number of peers build up in highly popular swarms, which is reflected in the corresponding download time curves that rise rapidly. For example, the download times in EqualSplit increase rapidly until about 1500 sec as no peers have departed until then. At this point, the download time drops sharply as a result of a horde of peer departures that occur when the last block in a swarm has been uploaded by the server. In contrast, both CheatSheet and PropSplit assign higher bandwidth to popular swarms from the start, so peer departures start much quicker in popular swarms considerably reducing their average download times. We note here that CheatSheet is implemented so as to begin with an allocation identical to PropSplit until it has a stable estimate of peer arrival rates, at which point it switches to the model-based optimal allocation.

EqualSplit, BitTorrent and AntFarm take considerably longer to reach steady state because the number of peers in highly popular swarms goes through multiple rounds of ramp-ups followed by bulk departures before stabilizing. In this experiment, AntFarm takes the longest time to converge to a steady state because after assigning 5 KBps to each swarm at the beginning, it allocates remaining bandwidth in small chunks of 5 KBps once every 200 sec. AntFarm requires many such 200 sec epochs in order to build a stable response curve for all swarms, resulting in higher download times during this convergence phase. We have observed (not shown for brevity) that reducing the epoch length does not help and sometimes hurts performance as it increases the measurement error in learned response curves.

Why does CheatSheet outperform other schemes even in steady state? Figure 8 shows the steady-state allocations of server bandwidth achieved by different schemes that explain this observation. Swarms are ordered from left to right in decreasing order of popularity. CheatSheet uses the model to predict that the most popular swarm is mostly self-sustaining and therefore needs only a small bandwidth to achieve healthy download times. Compared to other controllers, CheatSheet assigns higher bandwidth values to the next four popular swarms that belong to a regime where a small amount of server bandwidth disproportionately improves performance, which considerably reduces the average download time. PropSplit and AntFarm by design assign the most bandwidth to the most popular swarm, but the extra server bandwidth hardly benefits that swarm. BitTorrent is biased more towards the popular swarms (as it receives more peer connections from these swarms compared to singleton swarms), but its allocation is nevertheless sub-optimal. EqualSplit clearly makes a sub-optimal decision by allocating equal bandwidth to all swarms in the light of the above reasons.

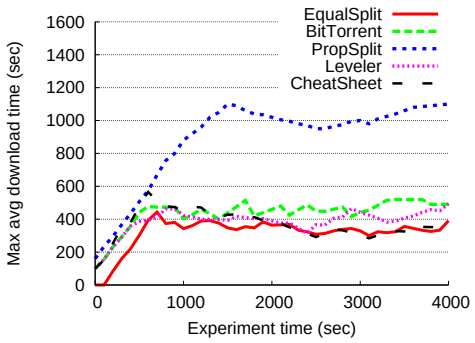


Figure 9: Maximum average download time across swarms for the Zipf workload.

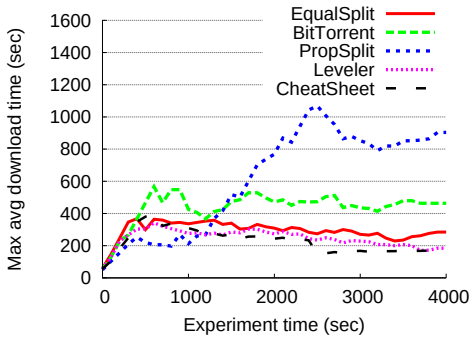


Figure 10: Maximum average download time across swarms for three self-sustaining swarms and one singleton swarm.

5.3 Min-max average download time

Next, we compare controllers on the MIN_MAX objective, i.e., minimizing the average download time of the swarm that has the worst average download time. We conduct experiments with two different workloads: (1) a Zipf workload same as that in the previous subsection, and (2) a workload dominated by popular swarms. We set the total server bandwidth to 500 KBps in both experiments.

5.3.1 Zipf workload results

Figure 9 shows the average download time of the swarm with the maximum average download time (referred to as MAD time in this discussion). We observe that, even though the workload is the same, the rela-

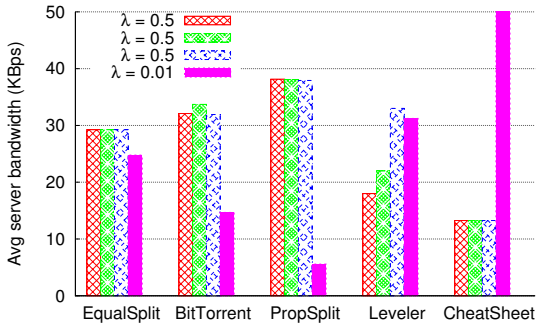


Figure 11: Server bandwidths to 3 self-sustaining swarms and one singleton swarm. CheatSheet gives higher bandwidth to the singleton swarm than other controllers.

tive performance of controllers is different compared to the experiment in the previous subsection. The MAD time achieved by PropSplit is twice as worse as other controllers that have relatively smaller differences between them. Both EqualSplit and CheatSheet achieve the lowest MAD time. BitTorrent incurs a higher MAD time in comparison to EqualSplit. The performance of Leveler varies with time because it changes the server bandwidth to each swarm periodically and struggles to converge to a steady bandwidth allocation as it shuffles bandwidth across 20 swarms. The reason (not visible in the figure) is that different swarms take different times to manifest the effect of the most recent change. Leveler sometimes "panics" and allocates more bandwidth to the currently worst swarms too quickly and at other times is too slow to move bandwidth away from swarms that could do without it. The fluctuating performance of Leveler reveals that it is nontrivial to design a robust dynamic controller.

Unpopular swarms, i.e., swarms with a small peer arrival rate, impact the MAD time significantly in this experiment. Unpopular swarms require higher bandwidth than popular swarms to achieve the same download time (Figure 2). Due to the Zipf popularity distribution, a majority of swarms for this workload are unpopular. PropSplit incurs the highest MAD times because it assigns the least bandwidth to the most unpopular swarm, which significantly increases the download time of that swarm. EqualSplit, unlike PropSplit, assigns equal bandwidth to all the swarms and hence has a much smaller MAD time. CheatSheet performs the same as EqualSplit because the unpopular swarms in the workload have nearly the same performance in both cases. Due to a large number of unpopular swarms, CheatSheet only assigns 5 KBps more bandwidth to each unpopular swarm than EqualSplit, which does not sufficiently impact the MAD times.

5.3.2 Heavy-head workload

Does EqualSplit achieve the least MAD times in all scenarios? Our experiment with heavy-head workload shows that it is not the case. This workload is dominated by popular swarms and consists of three highly popular swarms each with a peer arrival rate of 0.5/s and a fourth unpopular swarm with a peer arrival rate of 0.01/s. The total number of swarms is small in this experiment as we are limited by the total number of reasonably reliable nodes we could find on PlanetLab. The total server bandwidth is set to 120 KBps.

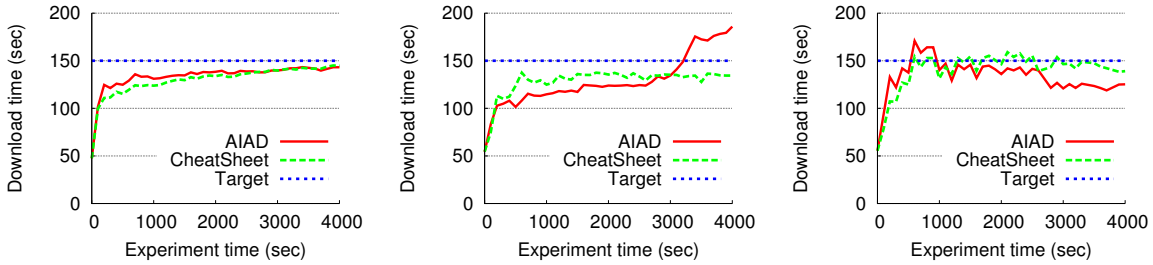
Figure 10 shows the MAD time over time for different swarms. Figure 11 (analogous to Figure 8) shows the corresponding bandwidth allocation decisions made by different controllers. Unlike the Zipf workload experiment, EqualSplit has a significantly higher MAD time than CheatSheet. This is because CheatSheet allocates twice the bandwidth to the least popular swarm compared to EqualSplit (swarm 4 in Figure 11). So this swarm incurs nearly twice the download time compared with EqualSplit than with CheatSheet, which is reflected in the corresponding MAD times. Leveler converges to the same value of the MAD time as CheatSheet over time. Unlike the previous Zipf workload, Leveler happens to converge more smoothly with this heavy-head workload workload consisting of just four swarms. This experiment illustrates that performance of static controllers such as EqualSplit can vary depending on the workload.

5.4 Target download time

Next, we compare CheatSheet and AIAD against the MIN_COST objective. We do not compare against the simplistic static schemes as they are designed to always use all available capacity (and can therefore be made to appear arbitrarily worse by choosing a sufficiently low target download time in an experiment). Our workload for this experiment consisted of six swarms with peer arrival rates of 0.5/s, 0.14/s, 0.12/s, 0.1/s, 0.08/s, and 0.01/s. All swarms distributed a file of size 10 MB. The target download time for all the swarms is set to 150 sec. For brevity, we only present detailed results for arrival rates 0.5/s, 0.12/s, and 0.01/s here. Results for other arrival rates are qualitatively consistent and are omitted due to lack of space.

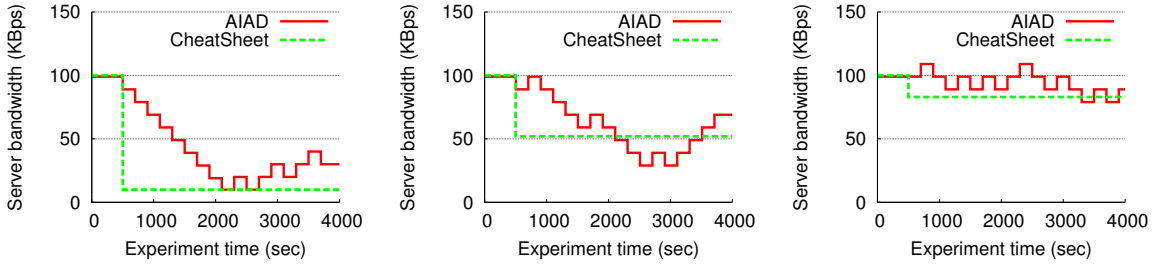
Figure 12 shows the average download time achieved by each strategy over the duration of the experiment. Figure 13 just below shows the corresponding server capacity set by the controllers over the same duration. The actual bandwidth consumed at the server is very close to the configured capacity shown in Figure 13 except during intervals when there happen to be no peers in the swarm.

Figure 12 shows that CheatSheet meets the target download time well in all cases, but AIAD sometimes significantly exceeds the target download time as in the



(a) $\lambda = 0.5/s$ (b) $\lambda = 0.12/s$ (c) $\lambda = 0.01/s$

Figure 12: Average download time of controllers with target download time = 150 s. CheatSheet meets the target well but AIAD fails to meet target for $\lambda = 0.12/s$



(a) $\lambda = 0.5/s$ (b) $\lambda = 0.12/s$ (c) $\lambda = 0.01/s$

Figure 13: Server bandwidth set by controllers. AIAD fails to meet the target for $\lambda = 0.12/s$ as it drastically reduces server bandwidth near 3000 s.

later part Figure 12(b). This is because AIAD is not always able to accurately estimate the relation between server bandwidth and the download time. To illustrate this point, Figure 14 shows the measured download rate of the swarm and the server bandwidth limit set by AIAD during this experiment. At t = 2200 s, the measured download rate of swarm is above the corresponding target download time (10 MB / 150 s = 67KBps). Hence it decreases the server bandwidth to 40 KBps at t = 2200 s and then to 30 KBps at t = 2400 s. This causes the measured download rate to drop sharply which is reflected in the increased download time of peers in Figure 12(b). The download time curve shows an increase somewhat later as it is calculated as an average over a window of 2000s.

We also experimented by changing the interval after which AIAD updates bandwidth to 300 sec, but it continues to fluctuate above the target download time. Of course, if the bandwidth update interval is increased to a sufficiently high value and the bandwidth increments/decrements made small, the AIAD controller will converge to the target download rate. However, it will take longer to converge and will be less responsive if peer arrival rates change.

CheatSheet consumes much less bandwidth compared to AIAD for $\lambda = 0.5/s$ especially in the first 2000 seconds of the experiment. While AIAD takes several cycles of measurement and perturbations to reach the bandwidth allocation, CheatSheet directly jumps to the minimal required bandwidth using its model.

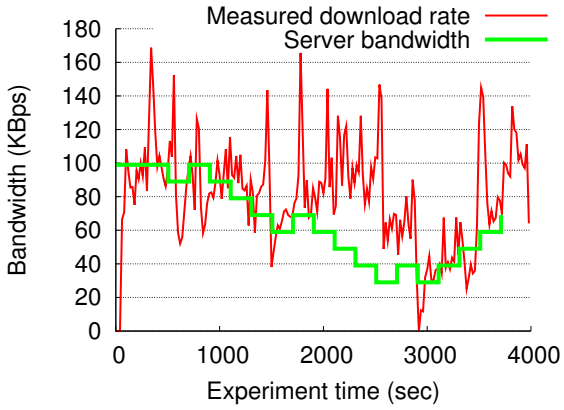


Figure 14: Server bandwidth set by AIAD in response to measured download rates for $\lambda = 0.12/s$.

5.5 Summary and discussion

In summary, our evaluation shows that bandwidth allocation done by static controllers is hit-or-miss. A static controller that works well for one objective and

workload combination may perform poorly for others. This is intuitively unsurprising and is also consistent with the findings in prior work analyzing a specific optimization objective [19]. For a fixed performance objective however, the simplicity of static controllers may outweigh their sub-optimality (e.g., EqualSplit for the MIN_MAX metric or PropSplit for the MIN_AVG metric).

Our evaluation also shows that designing a dynamic controller for scenarios involving peer arrivals and departures is nontrivial. Although dynamic controllers are generally superior to any given simplistic static scheme when evaluated over a range of objectives and workloads, we find that they are far from optimal. Indeed, in some scenarios, simple schemes like EqualSplit or PropSplit outperform dynamic control. The reason is that measuring the relationship between swarm performance and allocated bandwidth in an online manner is nontrivial. As a result of measurement errors, a dynamic control scheme is vulnerable to prolonged convergence delays or persistent fluctuations.

The experiments in this paper suggest that a model-based approach is feasible and promising. We find that when a model-based controller is given a cheat sheet based on prior measurements in the regime of interest, it consistently outperforms both static and dynamic controllers for different objectives and workloads.

Nevertheless, having gone through the experience of making a model-based controller work, our conclusions about its practicality are somewhat mixed. A model-based approach will work well only if (1) all the model parameters can be estimated accurately, and (2) the controller has access to an accurate model of swarm performance with respect to those parameters. The first requirement is challenging primarily because the effective peer upload capacity distribution may not be known or be stationary because of several reasons.

First, network conditions can significantly change the effective upload capacity distribution as shown in §3.2.6. Second, the user population for any particular content may have a (persistently) different upload capacity distribution than the general population at a managed swarming site, say because only high capacity peers are interested in very large high-definition movie files. In such cases, the model-based approach entails additional offline measurement to estimate the content-specific peer upload capacity distribution and to build corresponding response curves. Third, peers may download files from multiple swarms simultaneously or otherwise limit their upload capacity. If the aggregate effect of all of these factors makes the peer upload capacity distribution non-stationary and unpredictable, the model-based approach is unlikely to be effective.

6. RELATED WORK

Our primary contribution is a comparative analysis of different categories of bandwidth controllers for client-assisted content delivery systems and the design and implementation of a model-based control approach, that to our knowledge has not been attempted before. Our work builds upon a large body of prior work that can be grouped into dynamic controllers, models of swarm behavior, and swarm seeding strategies.

Dynamic controllers: Peterson et al. [19] take a dynamic controller approach to server bandwidth allocation in managed swarms. V-Formation [20] monitors propagation of each block through the swarm. Based on the propagation distance of each block, a central coordinator calculates bandwidth allocation from all peers to all swarms in a multi-swarm setting. Unlike V-Formation, the model-based controller requires only the aggregate peer statistics such as average arrival rate, let alone block-level information from each peer. Our comparison of controller strategies does not include V-Formation because its implementation is proprietary and not available publicly.

Models of swarm behavior: Qiu [22], Fan [6], and Liao [14] analytically model BitTorrent to derive expressions for average download time of peers and other swarm metrics. But, their models make assumptions that over-simplify swarm behavior. For example, upload capacity is homogenous [22], number of peers is fixed [14], seeds contribute full upload capacity to swarm [22, 6], and swarm inefficiency is fixed for all swarms [22]. To address these concerns, we took a measurement-based approach to model swarm performance.

Guo et al. [8] model the evolution of torrents from the time they are published till they become unavailable due to lack of seeds. Menasche et al. [16] model content availability in swarming system and show that content bundling exponentially reduces swarm unavailability. We address the server bandwidth allocation problem assuming that the server is always online and content availability is guaranteed.

Stutzbach et al. [26] study the churn due to arrival and departures of peers in three P2P networks, Gnutella, BitTorrent, and Kad, and characterize churn-related metrics such as the distribution of session lengths.

Incentive strategies: Several BitTorrent clients that improve BitTorrent's incentive strategies have been proposed. Piatek et al. [21] improve BitTorrent's incentive strategy by carefully selecting peers and contribution rates. Levin et al. [13] design a strategy proof BitTorrent client. FairTorrent [23] client uses a deficit-based distributed algorithm to improve fairness.

Dandelion [24] incentivizes peers to contribute bandwidth to swarm through the use of virtual currencies which can be redeemed by content providers for monetary discounts. AntFarm [19] implements a token-based protocol to limit selfish behavior. Our position is that

a large majority of users use BitTorrent clients as-is or use unmodifiable closed-source clients, e.g., Akamai's NetSession [1], so incentive issues are less important.

Legout et al. [12, 11] use measurements from an instrumented BitTorrent client to analyze specific aspects of BitTorrent, e.g., unchoking strategy. [12] shows that BitTorrent's rarest first strategy ensures close to ideal piece diversity and its choke algorithm is efficient in practice and is robust to free riders. [11] establishes the clustering of similar bandwidth BitTorrent peers, the effectiveness of BitTorrent's sharing incentives, and high upload capacity utilization of peers. In contrast, our work treats BitTorrent as a black box in building a measurement-based model of swarm performance.

Seeding strategies: There are two kinds of seeding strategies relevant in swarming systems: inter-swarm and intra-swarm. In this paper we focused on the former, relying on the intra-swarm strategies pre-built in the mainline BitTorrent. The key element of the intra-swarm seeding strategy implemented in the mainline BitTorrent is referred to as *super seeding* [9], which attempts to minimize the amount of data uploaded by a seed. Super-seeding, as well as other works that proposed alternative intra-swarm seeding strategies [2, 11, 4], are complementary to ours.

7. CONCLUSIONS

In this paper, we performed a comparative evaluation of strategies to control server bandwidth in client-assisted content delivery systems. As part of this effort, we introduced a new approach referred to as model-based control and presented the design and implementation of a model-based controller, CheatSheet, that uses a concise model based on a priori offline measurement of swarm performance as a function of the server bandwidth and other swarm parameters. Our experiments show that simple static strategies are unreliable as they perform well on some workloads and objectives but fare poorly on others. Dynamic control can also lead to a sub-optimal performance as it is prone to prolonged convergence delays and persistent fluctuations. In comparison, a model-based approach consistently outperforms both static and dynamic approaches provided it has access to detailed measurements in the regime of interest. Nevertheless, the broad applicability of a model-based approach may be limited in practice because of the overhead of developing and maintaining a comprehensive measurement-based model of swarm performance in each regime of interest.

8. REFERENCES

[1] Akamai. Netsession interface. `http://www.akamai.com/html/misc/akamai_client/netsession_interface_faq.html`.

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

[3] BitTorrent. Bittorrent, inc., 2008. `www.bittorrent.com`.

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

[5] Amazon EC2. `http://aws.amazon.com/ec2/`.

[6] B Fan, Dah-Ming Chiu, and J Lui. Stochastic differential equation approach to model peer-to-peer. In *ICC*, 2006.

[7] A Finamore, M Mellia, M Munafò, R Torres, and S Rao. Youtube everywhere: impact of device and infrastructure synergies on user experience. In *IMC*, 2011.

[8] L Guo, S Chen, Z Xiao, E Tan, X Ding, and X Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *IMC*, 2005.

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

[10] Little's Law. `http://en.wikipedia.org/wiki/Little's_law`.

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

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

[13] D Levin, K LaCurts, N Spring, and B Bhattacharjee. Bittorrent is an auction: Analyzing and improving Bittorrent incentives. In *SIGCOMM*, 2008.

[14] W Liao, F Papadopoulos, and K Psounis. Performance analysis of bittorrent-like systems with heterogeneous users. *Performance Evaluation*, pages 876 – 891, 2007.

[15] PP Live. `http://www.pplive.com/`.

[16] D Menasche, A Rocha, B Li, D Towsley, and A Venkataramani. Content availability and bundling in swarming systems. In *CoNEXT*, 2009.

[17] Octoshape. `http://www.octoshape.com`.

[18] Pando. `http://www.pando.com/`.

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

[20] R S Peterson, B Wong, and E G Sirer. A content propagation metric for efficient content distribution.

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

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

[23] A Sherman, J Nieh, and C Stein. Fairtorrent: bringing fairness to peer-to-peer systems. In *CoNEXT*, 2009.

[24] M Sirivianos, X Yang, and S Jarecki. Robust and efficient incentives for cooperative content distribution. *IEEE/ACM Trans. Netw.*, 2009.

[25] SopCast. `http://www.sopcast.com/`.

[26] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC*, 2006.

[27] Micro transport protocol. `http://en.wikipedia.org/wiki/Micro_Transport_Protocol`.

[28] Velocix. `http://www.velocix.com/`.

[29] Verizon. Fios. `http://www.verizon.com/fios`.