A Content Propagation Metric for Efficient Content Distribution

Ryan S. Peterson
Cornell University &
United Networks, L.L.C.
Dept. of Computer Science
Ithaca, NY
ryanp@cs.cornell.edu

Bernard Wong
Cornell University &
United Networks, L.L.C.
Dept. of Computer Science
Ithaca, NY
bwong@cs.cornell.edu

Emin Gün Sirer
Cornell University &
United Networks, L.L.C.
Dept. of Computer Science
Ithaca, NY
egs@cs.cornell.edu

ABSTRACT

Efficient content distribution in large networks comprising datacenters, end hosts, and distributed in-network caches is a difficult problem. Existing systems rely on mechanisms and metrics that fail to effectively utilize all available sources of bandwidth in the network. This paper presents a novel metric, called the Content Propagation Metric (CPM), for quantitatively evaluating the marginal benefit of available bandwidth to competing consumers, enabling efficient utilization of the bandwidth resource. The metric is simple to implement, imposes only a modest overhead, and can be retrofitted easily into existing content distribution systems. We have designed and implemented a high-performance content distribution system, called V-Formation, based on the CPM. The CPM guides V-Formation toward a global allocation of bandwidth that maximizes the aggregate download bandwidth of consumers. Results from a PlanetLab deployment and extensive simulations show that V-Formation achieves high aggregate bandwidth and that the CPM enables hosts to converge quickly on a stable allocation of resources in a wide range of deployment scenarios.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Performance

Keywords

Content distribution, Hybrid, Peer-to-peer

1. INTRODUCTION

Multimedia content distribution is a critical problem that accounts for a majority of all Internet traffic [3]. Delivering content at large scale with low cost requires taking advantage of all resources available. Yet existing approaches to content distribution have architectural and protocol limitations that fail to utilize available resources effectively.

Content distribution systems have three sources of bandwidth: content distributors' origin servers, in-network cache servers, and clients. Content distribution systems based on a client-server architecture, such as YouTube, place the entire resource burden on the first two sources of bandwidth, and thus necessitate a large initial investment and incur high running costs [16]. In contrast, peer-to-peer protocols, such as BitTorrent [1] and others [7, 31, 35], rely primarily on bandwidth contributed by clients. While these protocols permit the utilization of bandwidth from origin servers and in-network caches [4, 5], they lack mechanisms for managing such bandwidth to achieve commercial objectives and service level guarantees a content distributor might seek. Finally, a new class of emerging content distribution systems based on a hybrid, peer-assisted architecture [32] manage the bandwidth from a single centralized server using a global optimization. Yet the optimization mechanism in such systems does not support in-network caches or distributed datacenters that house only a partial subset of the managed content. As a result, hybrid systems cannot provide performance guarantees for a deployment that comprises origin servers in datacenters, cache servers, and clients.

Achieving a performance objective of any kind requires a *metric* that can measure the performance of the system and thus help adjust its behavior to progress toward the performance goal. But the design of a suitable metric is non-trivial. For instance, BitTorrent hosts use a ranking derived from continuous block auctions [19] to determine which peers to unchoke in order to maximize their reciprocal bandwidth. This ranking is of limited use in other systems because it is intertwined with the BitTorrent block transfer mechanisms, is vulnerable to attack [21, 27], and can lead to undesirable global behaviors such as swarm starvation [32]. An ideal metric would be effective at achieving globally-desirable performance objectives, easy to implement, able to handle network churn, and backwards compatible with existing systems.

This paper presents a unifying metric, called the *Content Propagation Metric (CPM)*, that enables a content distribution system to efficiently manage the resources of origin servers, in-network cache servers, and clients. The key insight behind this metric is to capture how quickly a host's uploaded content propagates transitively throughout a set of peers downloading that content (a swarm). To this end, the CPM is calculated by computing the average size of recent block propagation trees rooted at a particular host for a given swarm. The CPM handles changing swarm dynamics, such as changes in swarm size, changes in link capacities, churn, block availability, and content uploads from other hosts, which can all

SIGCOMM'11, August 15–19, 2011, Toronto, Ontario, Canada.

affect the rate of content propagation. The CPM offers a consistent way for hosts to measure their marginal utility to a particular swarm, and to make informed decisions with their bandwidth among swarms competing for content.[1]

This paper makes three contributions. First, it introduces and defines the content propagation metric, discusses how it can be realized in practice, and examines its effectiveness in dynamic settings. Second, it outlines the design and implementation of a content distribution system, called V-Formation, that uses the CPM to guide hosts toward an efficient allocation of bandwidth that maximizes global aggregate bandwidth. The CPM enables V-Formation to converge on an efficient system-wide allocation of bandwidth in a broad range of deployment scenarios that include origin servers, cache servers, and clients in multiple swarms. Finally, it evaluates the performance impact of using the CPM to existing content distribution systems through a deployment and simulations. PlanetLab experiments show that V-Formation can improve aggregate bandwidth by approximately 60% and 30% over BitTorrent and Antfarm, respectively.

The rest of this paper is structured as follows. Section 2 gives background on allocating bandwidth in the presence of multiple swarms. Section 3 states the general content distribution problem that this paper addresses, incorporating in-network caches. Section 4 describes the CPM in detail and the core approach for allocating bandwidth based on measurements from individual hosts. Section 5 describes our implementation of V-Formation, which we use to evaluate the CPM in Section 6. Section 7 places our approach to content distribution in the context of related work, and Section 8 concludes.

2. BACKGROUND

Existing swarming protocols, such as BitTorrent (Figure 1), use mechanisms that allocate bandwidth efficiently within a single swarm, but their policies do not make efficient use of bandwidth from multiple origin servers and in-network cache servers.

To address content distribution in deployments where multiple swarms compete for bandwidth from a server, Antfarm [32] introduced a peer-assisted protocol that offers coordination among peers using a logically centralized coordinator. Antfarm uses active measurements to compute the optimal allocation of bandwidth from a single origin server, called the *seeder* (Figure 2). Every swarm exhibits a response to bandwidth that it receives from peers: everything else remaining constant, increasing the bandwidth that a peer contributes to a swarm increases the aggregate bandwidth within the swarm. Antfarm represents this relationship with a *response curve*, which captures a swarm's aggregate bandwidth as a function of the seeder's bandwidth allocated to that swarm. The seeder can use response curves collected for every swarm to determine which swarms benefit most from its bandwidth: the steeper the slope of a response curve, the more aggregate bandwidth the corresponding swarm achieves from additional seeder bandwidth.

Response curves are costly to obtain in practice, which renders them impractical for highly dynamic swarms. Measuring a single data point in a response curve requires a seeder to operate at a particular bandwidth for sufficiently long that the swarm's aggregate bandwidth stabilizes. While it is clearly unnecessary to measure a swarm's entire response curve in order to derive meaningful information, a response curve must contain sufficient data near the point of operation to obtain the curve's slope and calculate the expected benefit of an increase in seeder bandwidth. Furthermore, the optimal point of operation can change rapidly as swarm memberships, network conditions, and block availability change.

[1]The word "metric" is used in the networking, not mathematical, context. http://en.wikipedia.org/wiki/Metrics_(networking).

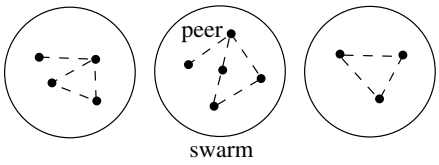

Figure 1: BitTorrent architecture. BitTorrent swarms are logically isolated; peers make bandwidth allocation decisions independently for each swarm.

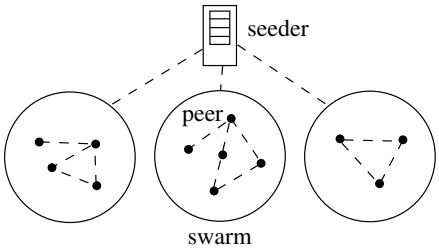

Figure 2: Antfarm architecture. Antfarm introduces a logically centralized coordinator to optimize bandwidth from a single origin server across multiple swarms, but neglects other inter-swarm bandwidth in its allocations.

Response curves provide an efficient allocation of bandwidth from a single seeder, but deployments where peers belong to multiple swarms add a level of complexity that response curves do not address. In a measurement study of over 6000 torrents and 960,000 users, we found that more than 20% of users simultaneously participated in more than one monitored torrent. Such peers are faced with choosing which swarms should receive their bandwidth, and their decisions can have dramatic effects on the performance of the system.

We discuss two approaches for adapting response curves to allocate bandwidth from multiple hosts, both of which result in suboptimal performance. In the first approach, the coordinator measures a set of response curves for each peer that belongs to multiple swarms, where each curve represents a swarm's response to bandwidth from a particular peer. Obtaining accurate measurements is difficult because peers' response curves are dependent on each other. The problem is exacerbated by the large time interval that a peer must wait for the swarm to stabilize at an aggregate bandwidth before taking a measurement; another peer's shift in point of operation during the time interval will perturb the measured value.

In an alternative approach, the coordinator instead maintains a single response curve per swarm, which captures the swarms' responses to bandwidth, independent of which particular hosts supply the bandwidth. The coordinator performs Antfarm's optimization on the response curves to calculate the optimal amount of bandwidth that each swarm should receive. Then, the coordinator assigns hosts to upload to particular swarms in order to realize the optimal bandwidth allocation based on the computed target swarm bandwidths and swarm memberships and upload capacities of individual peers. There are two problems with this approach. First, the assignment problem of assigning peers to swarms is difficult to solve at large scales, and greedy algorithms for assigning peer bandwidth to swarms can result in poor use of peers' resources. Second, using a single response curve for each swarm neglects vari-

ations among peers, such as which blocks they possess and network conditions to members of each of their swarms.

Overall, response curves offer an intuitive model for swarms that enables a logically centralized seeder to allocate bandwidth optimally among competing swarms. However, real-world issues render them less useful for highly dynamic swarms, and infeasible when swarms compete for bandwidth from multiple hosts distributed throughout the network or peers that belong to multiple swarms.

3. PROBLEM STATEMENT

To formalize bandwidth allocation among multiple swarms, we introduce the *general multi-swarm content distribution problem*. This defines a global performance goal over a class of realistic content distribution scenarios comprising origin servers, in-network caches, and end hosts organized into swarms (Figure 3).

Formally, given a set of peers P, a set of swarms S, and a set of memberships $M \subseteq P \times S$, the general multi-swarm content distribution problem is to determine the upload bandwidth $U_{p,s}$ that peer p should allocate to swarm s for all $(p, s) \in M$ in order to maximize global aggregate bandwidth $\sum_{p \in P} D_p$, where D_p is the download bandwidth of peer p.

This general formalization removes restrictions on the location of content and membership of peers in swarms. The CPM addresses the general multi-swarm content distribution problem by guiding hosts to an efficient allocation of bandwidth in these deployment scenarios.

4. APPROACH

The Content Propagation Metric provides an accurate measure of hosts' contributions, offering a practical approach for addressing the general multi-swarm content distribution problem. This section describes the CPM in detail. It then explores how peers use measured CPM values to compute an efficient allocation of bandwidth. The section concludes with discussions of how the CPM remains effective in the presence of highly dynamic swarms. We leave implementation details, including how to obtain and process CPM measurements, to Section 5.

4.1 Block Propagation Bandwidth

Peers that simply aim to saturate their upstream bandwidth without regard to the selection of the download recipients are not necessarily acting in the best interest of the global ecosystem. A recipient that fails to forward blocks to other peers provides little benefit to the swarm. The propagation of a block is hindered if a peer that receives it is unwilling to contribute its upstream bandwidth, or if the receiving peer's neighbors already possess the block. It is more beneficial to upload blocks to peers that are willing to contribute their bandwidth but lack desirable blocks that enable them to saturate their own upload capacities. As a result, blocks of equal rarity can have vastly different values to a swarm depending on which particular peers have those blocks and what other blocks those peers possess.

Block propagation bandwidth is a metric that captures these complex multi-peer interactions by encompassing the global demand for blocks, block availability, network conditions and topologies, and peer behavior. Block propagation bandwidth is defined for a particular block transfer between two peers, called a *tracked transfer*. Informally, the metric is the system-wide bandwidth during a specified time interval resulting from block transfers that occurred as a direct consequence of the tracked transfer. This metric provides

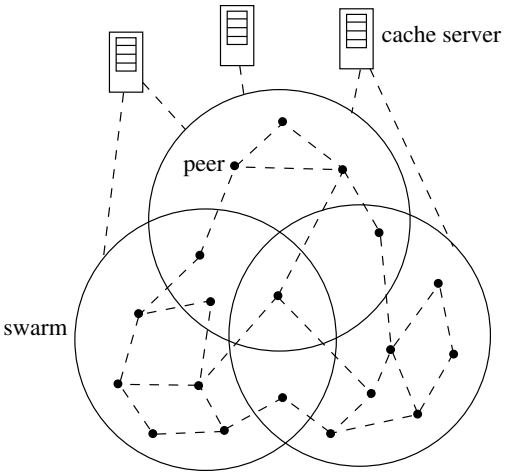

Figure 3: General multi-swarm content distribution. Peers, which includes all hosts that upload or download content, belong to arbitrary sets of potentially overlapping swarms.

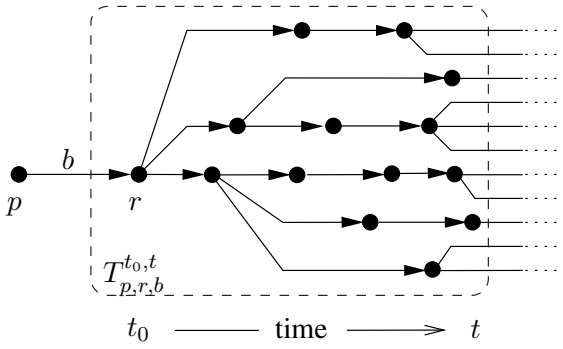

Figure 4: Propagation of a block. The dashed box indicates the propagation tree that results from peer p's tracked transfer of block b to peer r. The block propagates exponentially during the measurement time interval $\tau = t - t_0$, resulting in propagation bandwidth $v^{t_0,t}_{p,r,b} = 14 \cdot 256$ KBytes$/30$ s ≈ 120 KBytes/s, assuming 256-KByte blocks and $\tau = 30$ seconds.

an estimate of the benefit that results from a single block transfer from one peer to another.

Formally, for the upload of block b from peer p to peer r, where the transfer completes at time t_0, we define a *block propagation tree* $T^{t_0,t}_{p,r,b}$ rooted at r with a directed edge from p_1 to p_2 if r is an ancestor of p_1, and p_1 finishes uploading b to p_2 at time t' such that $t_0 < t' \leq t$. Thus, $T^{t_0,t}_{p,r,b}$ is essentially an implicit multicast tree rooted at peer r for block b during the time interval $\tau = t - t_0$. The block propagation bandwidth, then, is

$$v^{t_0,t}_{p,r,b} = \left|T^{t_0,t}_{p,r,b}\right| \cdot \text{size}(b)/(t - t_0),$$

the download bandwidth enabled by p's tracked transfer to r over the time interval τ. Figure 4 shows an example propagation of a block and the resulting propagation tree. Assuming 256-KByte blocks and a τ of 30 seconds, the example block propagation bandwidth is approximately 120 KBytes/s.

Block propagation bandwidth enables peers to compare the relative benefits of their block uploads to competing swarms over a common time interval τ. To illustrate the metric and its relation to

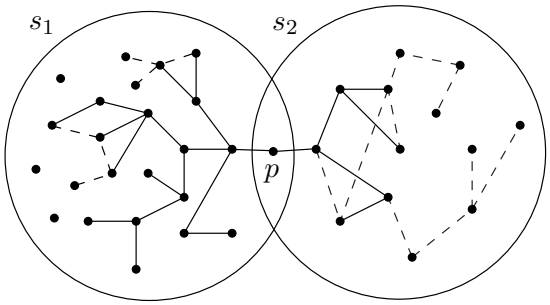

Figure 5: Block propagations in two competing swarms. Solid edges indicate the propagation of a particular block uploaded by peer p. Dashed edges indicate transfers of other blocks that compete with p's block for peers' upload bandwidth. p's block propagates more widely in swarm s_1 than in s_2.

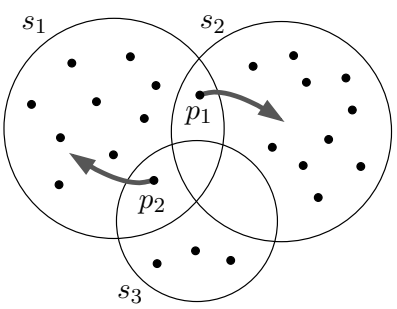

Figure 6: Three competing swarms. Peer p_1 in swarms s_1 and s_2 and peer p_2 in swarms s_1 and s_3 converge on the allocation of bandwidth indicated by the gray arrows. p_2's CPM value for s_3 is smaller than for s_1 due to the swarms' sizes; p_1 allocates its bandwidth to s_2, where its blocks do not compete with p_2's uploads.

the value of a block upload, consider the block propagations shown in Figure 5. Peer p is a member of swarms s_1 and s_2, to which p uploads tracked blocks. On average, peers in s_1 distribute their blocks more widely than peers in s_2, as indicated by solid edges. Dashed edges indicate peer-to-peer transfers of other blocks, which compete for peers' upload bandwidth. The higher average block propagation in s_1 can be due to several factors, including swarm size, competing uploads, peer behavior, and network conditions.

4.2 Content Propagation Metric

Block propagation bandwidth captures the utility of a given block upload, which may suffer from a high rate of fluctuation depending on that block's relative rarity and peer r, the peer that receives the block in the tracked transfer. To compensate for such fluctuations, the CPM is based on a statistical sample of blocks that are disseminated by each peer.

The CPM captures the utility of a peer to a given swarm based on its recent uploads. A peer's CPM value for a particular swarm is computed from block propagation bandwidths obtained within a recent time interval $\pi = t' - t$. Formally, let

$$V_{p,s}^{t,t'} = \{v_{p,r_1,b_1}^{t_1,t_1'}, v_{p,r_2,b_2}^{t_2,t_2'}, \dots\}$$

be the set of all block propagation measurements where blocks $b_1, b_2, \dots$ are from swarm s and $t < t_i' \leq t'$ for all i. Then,

$$\mathrm{CPM}_{p,s}^{t,t'} = \Big(\sum_{v \in V_{p,s}^{t,t'}} v \Big) / |V_{p,s}^{t,t'}|,$$

the average of the measurements. We define

$$\mathrm{CPM}_{p,s} = \mathrm{CPM}_{p,s}^{t^*-\pi,t^*},$$

with the times omitted, to be p's current value for swarm s during the most recent time interval π, where t^* is the current time. Each value $\mathrm{CPM}_{p,s}$ is implemented as a rolling average that is continually updated as new block propagation bandwidths become available and old measurements become stale.

The CPM distills the salient properties of response curves for deciding to which swarms peers should upload their blocks in order to yield high aggregate bandwidth. A peer's CPM value for a particular swarm approximates the instantaneous slope of the swarm's response curve at its point of operation. Whereas Antfarm measures a response curve and uses its slope to predict a swarm's bandwidth yield for any amount of seeder bandwidth, the CPM directly measures the slope of the response curve without the need to explicitly generate the curve.

To illustrate the CPM, consider the bandwidth allocation of two peers originating content for three new swarms with identical network conditions and no competition from other uploaders, as depicted in Figure 6. Swarms s_1 and s_2 distribute popular content, with new downloaders joining at a higher rate than swarm s_3. Peer p_1 possesses content for s_1 and s_2, and peer p_2 possesses content for s_1 and s_3. After uploading a few blocks and measuring CPM values, p_2 identifies s_1 as the swarm that benefits more from its bandwidth due to, in this example, its larger size. p_1 likewise measures a high CPM value for s_1, but blocks uploaded by p_2 interfere with p_1's uploads, causing both peers' CPM values for s_1 to diminish. Consequently, p_1 allocates its bandwidth to s_2, which lacks the competition of p_2's uploads.

As swarm dynamics change, CPM values shift to adjust peers' bandwidth allocations. Continuing the above example, after s_1 has received sufficiently many blocks, its peers may be able to sustain high aggregate bandwidth without support from p_2. In this case, p_2's block uploads to the swarm would compete with a large number of uploads from the peers themselves, causing p_2's blocks to propagate less. In turn, p_2's CPM value for s_3 may exceed its CPM value for s_1, causing p_2 to allocate its bandwidth to s_3 instead.

The CPM provides peers information to allocate bandwidth based on current swarm dynamics. It might be tempting to instead use a global rarest policy, where peers request rare blocks from neighbors regardless of swarm, and peers in multiple swarms preferentially satisfy requests for blocks that are rarest within their respective swarms. However, such a policy operates solely based on the number of replicas of each block, and disregards swarm dynamics and peer behavior.

A peer's CPM value provides an accurate estimate of the peer's value to a swarm relative to competing swarms. The CPM captures the average benefit that peers' recent block uploads had on their swarms, providing a useful prediction of the value of future block uploads.

4.3 Robustness of the CPM

The CPM handles changes in swarm membership and highly dynamic swarms, and achieves high performance in deployments with swarms of vastly different sizes. The CPM naturally dampens oscillations to converge on a stable allocation of bandwidth. We discuss these issues in turn.

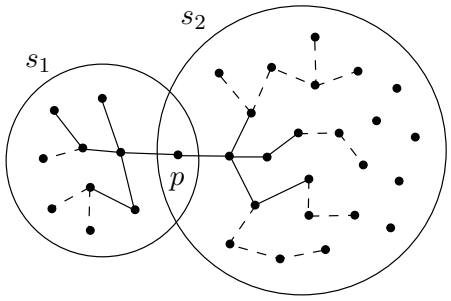

Figure 7: Measurement time interval. Edges indicate the propagation of blocks originating at peer p in swarms s_1 and s_2. Solid edges show the propagations within a time interval τ that is too small for p to differentiate its benefit to the competing swarms. Using a larger τ, indicated by dashed edges, makes it clear that s_2 receives more benefit from p's blocks.

4.3.1 Probing Swarms

Highly dynamic swarms pose two challenges for determining efficient bandwidth allocations. First, when peers join swarms for which they have no block propagation data, they are unable to compute the marginal benefit to the system from uploading blocks to the new swarm versus uploading blocks to a competing swarm. Second, swarms with high peer churn can respond very differently to a peer's contributions from one moment to the next. Consequently, such swarms can regularly invalidate many peers' CPM values, causing them to operate suboptimally.

Probing swarms enables calculation of CPM values for these problematic swarms with minimal overhead. To probe a swarm, a peer temporarily prioritizes requests for blocks in that swarm above other block requests until it has uploaded a small, constant number of blocks to the swarm. Data blocks are typically 128–256 KBytes, and we have found two block uploads to be sufficient for computing provisional CPM values to adapt to highly dynamic swarms.

4.3.2 Measurement Time Interval

The CPM measures the initial surge of block exchanges that occurs when a peer injects blocks into a swarm. The growth of a block's propagation tree reflects the swarm's demand for the block with respect to block availability, peer behavior, and network conditions within the vicinity of its tracked uploader. The wide range of swarm behavior means that using a globally constant time interval τ for measuring block propagations from all peers is insufficient.

Figure 7 gives an intuition of how the choice of measurement time interval affects a peer's ability to differentiate among competing swarms. Swarm s_1 is significantly smaller than s_2, but, assuming comparable network conditions and competition for blocks, using a small τ prevents p from recognizing that s_2 receives more benefit from each block. The propagation trees for small τ, represented by solid edges, are nearly identical in the two swarms, causing p to allocate its bandwidth equally between them. Increasing τ enables p to discover s_2's ability to achieve higher aggregate bandwidth than s_1 for each block.

Measuring block propagation with a τ that is unnecessarily large likewise decreases performance. A large measurement time interval increases the delay between the time p finishes uploading a block and the time p has an updated CPM value that incorporates the newly measured block propagations. Thus, choosing a suitable τ is a tradeoff between system performance, measured as aggregate bandwidth, and system adaptability, or the time required for the system to converge on a new allocation of bandwidth in highly dynamic deployments.

Our implementation of V-Formation addresses the CPM's sensitivity to the measurement time interval by choosing an interval for each peer that teases apart the peer's highest-valued swarms. The system maintains a measurement time interval τ_p specific to each peer p. Based on recent block propagation data, the system adjusts τ_p in order to account for changes in size of p's swarms. To do this, the system periodically uses its record of p's recent block uploads to measure block propagation bandwidths for three different values of τ: τ_p, $\tau_p^{\text{low}} = 1/2 \cdot \tau_p$, and $\tau_p^{\text{high}} = 2 \cdot \tau_p$. It then updates τ_p with the smallest of the three time intervals for which p achieves different CPM values for its two swarms with the largest CPM values, corresponding to the swarms for which p has the greatest impact. Thus, the system continuously and iteratively computes τ_p, adjusting its value over time.

4.3.3 Stabilization

The CPM mitigates oscillations in bandwidth allocations despite complex interactions among peers that influence multiple swarms. First, changes in CPM values only affect the bandwidth allocations of peers that belong to multiple swarms. The remaining majority of peers propagate blocks within their respective swarms regardless of CPM values, dampening the effects of shifting bandwidth allocations on a swarm's aggregate bandwidth. Second, a peer's CPM values for competing swarms regulate the peer's bandwidth allocation among the swarms. A peer's CPM value for a swarm naturally decreases as the peer uploads to the swarm because the uploads increase competition for downloading peers' upload bandwidth. Once the CPM value drops below the CPM value of a competing swarm, the uploading peer allocates its bandwidth elsewhere, leaving the swarm with sufficient content to temporarily maintain a steady aggregate bandwidth. In Section 6, we show that system aggregate bandwidth converges stably when there are multiple swarms vying for bandwidth from cache servers with limited upload capacity.

5. IMPLEMENTATION

We have implemented our approach to content distribution based on the CPM in a system called V-Formation. V-Formation adopts a hybrid architecture that combines peer-to-peer exchanges with bandwidth from optional cache servers managed by a logically centralized coordinator. The coordinator tracks block exchanges in swarms based on the transfer of tokens and uses its measurements to compute CPM values. Tokens are unforgeable credits minted by the coordinator that function as a virtual currency; peers exchange tokens with each other for content blocks and reveal spent tokens to the coordinator as proof of contribution. Each token can only be spent once, and the coordinator verifies that each token is spent for a block within the swarm for which it was minted. V-Formation augments Antfarm's token protocol to include an identifier for the specific block for which a token was exchanged.

This section first discusses the operation of the coordinator, then the operation of peers based on the coordinator's guidance. For simplicity, the discussion assumes compliant peers that follow the protocol as proscribed; we address incentive compatibility in Section 5.4.

5.1 Coordinator

V-Formation's logically centralized coordinator consists of three components: web servers, processors, and a shared state layer (Figure 8). First, web servers process requests from peers to dispense

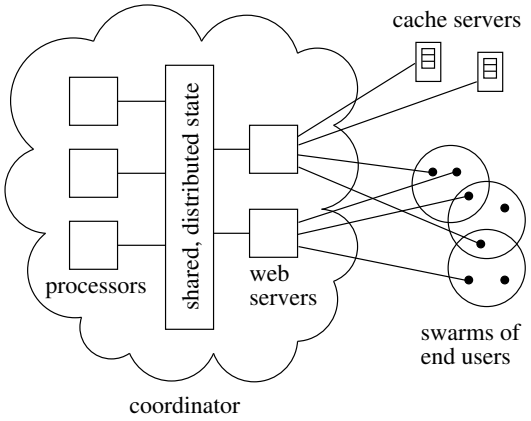

Figure 8: The V-Formation coordinator. Web servers communicate with peers (including cache servers) to gather information on swarm dynamics. Processors use swarm dynamics to compute peer bandwidth allocations. Web servers and processors communicate via a distributed state layer.

fresh tokens, collect spent tokens, and notify peers of computed bandwidth allocations. Second, *processors* use token exchanges aggregated by the web servers to calculate peer bandwidth allocations. Third, a distributed state layer grants web servers and processors read and write access for consolidating block exchange information and bandwidth allocations. The web servers, processors, and state can be distributed across multiple physical machines, or they can run on a single physical host for smaller deployments. We describe the operations of web servers and processors in turn, followed by a summary of all information stored in the distributed state layer.

5.1.1 Web Servers

V-Formation's peer-facing web servers function as an augmented tracker for facilitating swarms, similar a BitTorrent tracker. Web servers accept three types of requests asynchronously from peers: announce, get_tokens, and deposit_tokens. Peers issue periodic announce requests for each of their swarms to obtain addresses of other peers and to discover how to allocate bandwidth to swarms. An announce request contains the requesting peer's identifier and a swarm's identifier, represented as a 20-byte hash. A web server responds to an announce request with addresses of a random set of peers and the requesting peer's most recent CPM value for the swarm. Announce intervals are dynamically adjusted to achieve a constant CPU utilization of the web servers.

The get_tokens and deposit_tokens requests facilitate the exchange of fresh and spent tokens between peers and the coordinator, respectively. The coordinator maintains a credit balance for each peer that represents the total number of tokens that the peer can obtain across all swarms. A get_tokens request deducts from the issuer's credit balance in exchange for fresh tokens for a particular swarm, to be exchanged for blocks within a specific time interval. When a peer receives a token from another peer in exchange for a block, the peer sends it to the coordinator in a deposit_tokens request. The web server verifies the token's authenticity, increases the peer's balance accordingly, and, if the coordinator is tracking the block referenced in the token, records the block exchange in the state layer.

Web servers record block exchanges in *block exchange forests* for use by processors. A block exchange forest contains a set of peer identifiers as vertices and recent block exchanges as timestamped, directed edges. Each forest is specific to a particular block in a particular swarm; web servers build a separate forest for each block that the coordinator is tracking. To add a block exchange to a forest, a web server simply adds an edge from the block's sender to its recipient, timestamped using the coordinator's clock upon receiving the token.

5.1.2 Processors

Processors continuously iterate over block exchange forests to extract block propagation bandwidths. A single forest contains a propagation bandwidth for each edge timestamped prior to the current time minus τ, the time interval over which block propagations are measured. A processor extracts a propagation bandwidth for each such edge, prunes those edges from the forest, and records the new propagation bandwidths in the state layer.

Before extracting propagation bandwidths, the processor adjusts the timestamps on the forest's edges such that no edge is timestamped later than any edge in its subtree. Such an inconsistency occurs if a peer deposits its spent tokens before an ancestor in a block propagation tree deposits its own tokens for the same block. To make the adjustment, the processor recurses on each of the forest's roots, setting each timestamp to the minimum of its own timestamp and the earliest timestamp in its subtree. This makes the forest reflect the constraint that a peer can only upload a block after it has received that block.

To extract block propagation bandwidths, the processor recurses on each vertex in a forest, summing up the number of edges in each subtree. The processor only calculates and reports propagation bandwidths for edges older than the measurement time interval τ. It removes the corresponding edges from the forest and appends each new propagation bandwidth to a list in the state layer of propagation bandwidths for the tracked uploader's contribution to the forest's swarm. Each new propagation bandwidth is timestamped with the time on the forest's tracked transfer edge, equal to the time that the tracked transfer completed.

Web servers compute CPM values for a peer's swarms by averaging the propagation bandwidths in the list that are timestamped within a recent time interval π, a global constant set to five minutes in our implementation. In an announce response, web servers report the most recent CPM value, or, if there are no recent propagation bandwidths, instruct the requester to upload blocks to the swarm to obtain fresh measurements.

Operating on block exchange forests is a highly parallel task. Each forest represents exchanges of a single block for a single swarm, enabling processors to operate on block exchange forests in isolation. Multiple processors coordinate their behavior through the state by atomically reading and incrementing the swarm and block identifiers for the next forest to process. Thus, increasing the number of processor machines linearly increases the supported processing workload of the coordinator.

If high load renders the coordinator unable to process all block propagation forests, the coordinator sheds load by decreasing the fraction of blocks that it tracks. The coordinator maintains a dynamically adjusted parameter that dictates the fraction of blocks to track, enabling web servers and processors to independently determine whether a particular block should be tracked. Web servers do not insert forest edges for untracked blocks, and processors do not iterate over forests of untracked blocks. The coordinator adjusts the parameter such that for each swarm, some block is processed with a target frequency.

5.1.3 Distributed State Layer

V-Formation uses memcached to implement a distributed, shared state layer for web servers and processors. The coordinator's state is linear in the number of swarms it supports and in the number of peers.

The coordinator maintains data structures for each of the swarms it supports as well as for each peer in the system. Since this state is stored in memcached, it is distributed across multiple servers. Atomic compare and swap operations supported by memcached enable nodes to update this state quickly and concurrently. Since all such state is soft and can be recreated through remeasurement, if necessary, it need not be stored on disk. All lookups are performed with a specific key, so the memcached key-value store suffices, and an expensive relational database insertion is unnecessary.

For each peer, the state layer maintains its address, port, identifier, credit balance, and the set of swarms to which it belongs. For each swarm, the state layer keeps a swarm identifier and the set of peers in the swarm. To make bandwidth allocations, for each peer the state layer records its current τ for computing block propagation bandwidths, its current CPM value for each swarm, and a history of block propagation bandwidths for each measured block over the past time interval π. Recent block transfers for measured blocks are stored as block propagation forests linear in size to the number of peers and edges that they contain. Forests are pruned as block propagation bandwidths are extracted, based on peers' values of τ. Lastly, the state layer maintains a single value representing the next block that a processor should analyze, which processors advance each time they read it.

5.2 Peers

Peers interact with the coordinator by issuing announce requests periodically for each swarm, get_token requests when their fresh tokens are nearly depleted, and deposit_token requests when they possess spent tokens from other peers. They interact with other peers through block requests for the rarest blocks among directly connected neighbors, and they satisfy block requests according to their CPM values for competing swarms. Upon receiving a block, a peer responds with a fresh token embedded with the block's identifier.

To allocate bandwidth among competing swarms, peers prioritize their swarms based on the CPM values contained in announce responses. Upon receiving a CPM value, a peer updates a local prioritized list of its swarms. When a peer has received multiple outstanding block requests from peers in different swarms, it satisfies the request from the swarm with the largest CPM value.

If the coordinator lacks current information on recent block exchanges for any swarm to which a peer belongs, the coordinator reports that the peer should probe the swarm. The special probe flag instructs a peer to upload a small, constant number of blocks to the swarm for the coordinator to track.

5.3 QoS Guarantees

In some cases, it may be desirable to guarantee a minimum bandwidth for particular swarms in order to meet service-level agreements or to provide a certain quality of service to designated swarms. V-Formation enables system administrators to specify lower bounds for swarms where necessary, sacrificing overall system performance in favor of control. V-Formation satisfies such requests by diverting upload bandwidth from members of the swarm whose CPM values suggest that they should upload to other competing swarms instead. In order to minimize impact on overall system performance, the coordinator iteratively reassigns bandwidth from peers with low CPM values for competing swarms until the target service level has been achieved.

5.4 Security

Securing a peer-assisted content distribution system from malicious users or free-loaders is significantly more tractable than securing pure peer-to-peer systems. The coordinator in a peer-assisted systems serves as a trusted entity in the system that can detect attacks and enforce access control when coupled with a secure wire protocol.

The V-Formation wire protocol is an extension of the Antfarm wire protocol [32] and shares the same security guarantees. It makes standard cryptographic assumptions on the infeasibility of reversing a one-way cryptographic hash function. It also assumes that packets cannot be read or modified by untrusted third parties at the IP level. Such an attack is difficult without collusion from ISPs, and a successful attack only influences peers whose packets can be snooped and spoofed. We support SSL on peer-to-peer and peer-to-coordinator exchanges, respectively. While a formal treatment of the security properties of the protocol is beyond the scope of this paper, prior work [35] has established the feasibility of a secure, cryptographic wire protocol using a trusted, logically centralized server.

V-Formation's token protocol incentivizes peers to report accurate and timely information about block exchanges to the coordinator. Fresh tokens contain unforgeable identifiers known only to the coordinator and the peers for which they are minted. When a peer receives a token in exchange for a block, the token's recipient embeds the identifier of its original owner and verifies the block identifier embedded in the token before depositing the token at the coordinator. The coordinator verifies that the token's spender and depositor are members of the swarm for which the token was minted, according to its records. In addition, the coordinator checks that the token was deposited before its expiration time, ensuring that it represents a recent block exchange. If a check fails, at least one of the two peers known to have touched the token are at fault, and the coordinator marks both peers possible culprits.

Peers are incentivized to deposit tokens soon after receiving them in exchange for blocks, resulting in accurate block propagation forests at the coordinator. All tokens must be deposited at the coordinator before they expire, placing an upper bound on the deviance of a block exchange's timestamp. Peers with small credit balances are incentivized to deposit tokens earlier than their expiration times in order to receive fresh tokens to spend. Consequently, when the coordinator adjusts timestamps on forest edges before extracting propagation bandwidths, promptly deposited tokens result in early timestamps that percolate up the forest trees, replacing timestamps of exchanges whose tokens were deposited significantly after the block exchanges actually occurred.

6. EVALUATION

We have implemented the full V-Formation protocol described in this paper, both in a deployed system that is actively running on FlixQ [2], and in a simulator for fine-grain analysis of its performance. Through a deployment on PlanetLab [6] and extensive simulations, we compare V-Formation's performance to Antfarm, BitTorrent, and a BitTorrent-like *global rarest* policy where peers request the rarest blocks for which they are interested across all swarms. We also evaluate secondary features of the CPM, such as convergence time to a stable allocation and sensitivity to changing swarm dynamics.

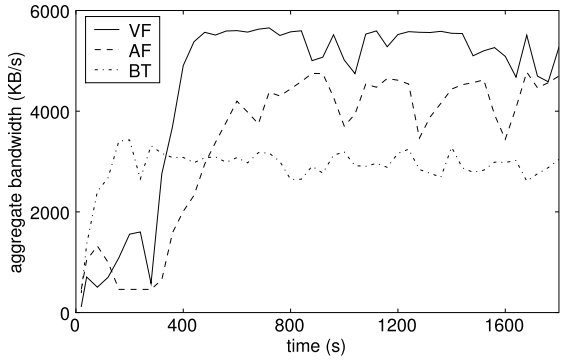

Figure 9: Performance on PlanetLab. 380 nodes in 200 swarms download movies from FlixQ using the V-Formation protocol and the same movies using the Antfarm and BitTorrent protocols.

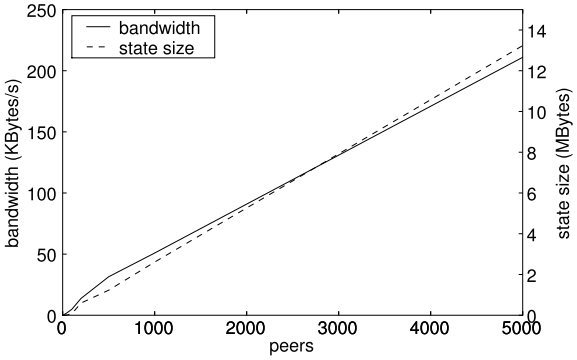

Figure 10: Scalability. Both memory consumption and bandwidth at the coordinator scale linearly with the size of the system.

6.1 Live Deployment

We evaluated our live deployment of V-Formation against Antfarm and version 5.0.9 of the official BitTorrent implementation. Our V-Formation deployment uses a distributed coordinator deployed in the Amazon EC2 cloud. In this experiment, 380 PlanetLab nodes each download one or more of 200 simulated movies, where a random 20% of the downloading nodes join two or more swarms to reflect the results of our BitTorrent trace. Two cache servers running on PlanetLab nodes seed the swarms. We scaled down the upload capacities of the cache servers to 50 KBytes/s each to reflect our relatively small deployment size. Peer upload capacities are drawn from the distribution of BitTorrent peer bandwidth collected by Pouwelse et al. [25]. This distribution specifies a median and 90th percentile peer upload capacity of 30 and 250 KBytes/s, respectively. The peers' download capacities are set 50% higher than their upload capacities to simulate asymmetric links.

The results of the experiment (Figure 9) show the three systems' aggregate bandwidths over time. V-Formation exhibits similar initial behavior as Antfarm, with lower aggregate bandwidth than BitTorrent in the first six minutes as peers probe swarms to determine an efficient allocation of bandwidth. V-Formation transitions to its steady state more quickly than Antfarm as a result of its lightweight probes, and it maintains a significantly higher steady state aggregate bandwidth than Antfarm and BitTorrent.

V-Formation's logically centralized coordinator is a potential performance bottleneck, and a poor implementation could limit the

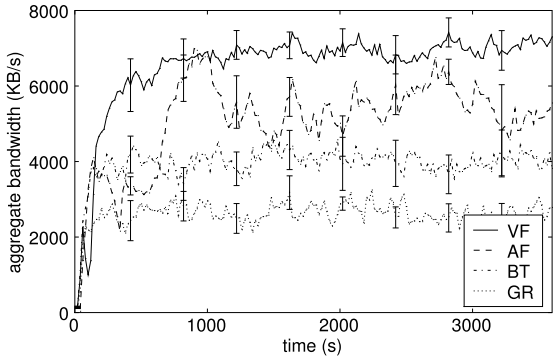

Figure 11: Comparison of protocols. Peers download movies with lengths and popularities randomly drawn from the Internet Movie Database. Peers have link capacities drawn from a distribution determined by a BitTorrent measurement study. Error bars indicate 95% confidence.

system's scalability. The next experiment examines how our implementation of the coordinator scales as a function of the size of the deployment; we found that the coordinator's bandwidth and memory requirements scale linearly with the total number of peers (Figure 10). In this experiment, peers are simulated across hosts in a computer cluster. Each peer is assigned a random bandwidth drawn from the same measured BitTorrent distribution as in the PlanetLab deployment. A peer's bandwidth is proportional to the rate at which the peer simulates receiving blocks from random participants in its swarm. Hosts in the cluster issue realistic deposit_token requests to the coordinator according to these simulated block transfers, as well as periodic announce requests. We made minor modifications to the coordinator to accept deposited tokens as if they were coming from legitimate peers with different IP addresses. In the experiment, three new peers enter the system every second and join a swarm for a 1-GByte file with 256-KByte blocks. The coordinator is distributed over two Amazon EC2 instances, each running a web server, a processor, and a slice of the memcached shared state layer. The reported memory usage includes all CPM, swarm, and peer metadata stored in the state layer, as discussed in Section 5.1.3. Coordinator bandwidth includes all outgoing tokens, CPM values, and responses to announce requests.

6.2 Simulations

Simulation experiments provide an in-depth examination of the CPM and how it affects hosts' bandwidth allocations in V-Formation. We first show the system-wide aggregate bandwidth of V-Formation, Antfarm, BitTorrent, and a global rarest policy for a realistic simulation based on movies in the Internet Movie Database (IMDb). The experiment is based on the number of votes and lengths of 425,000 movies, scaled down to 500 peers and 300 swarms to make simulations feasible. Each swarm facilitates the download of a single movie file, and each swarm's popularity is proportional to the number of votes that its movie has received on IMDb, resulting in a power-law distribution of swarm sizes. Each file's size is based on the movie's length and 1 Mbit/s video compression, common for 480p video. Swarm memberships are assigned iteratively, each of approximately 670 movie downloads randomly assigned either to a peer that has already been assigned one or more downloads, or to a fresh peer with no assigned downloads, the probability of each case calibrated so that 20% of peers belong to multiple swarms to reflect our BitTorrent trace. As in the PlanetLab deployment, nodes draw their bandwidth distribution from the measured BitTorrent band-

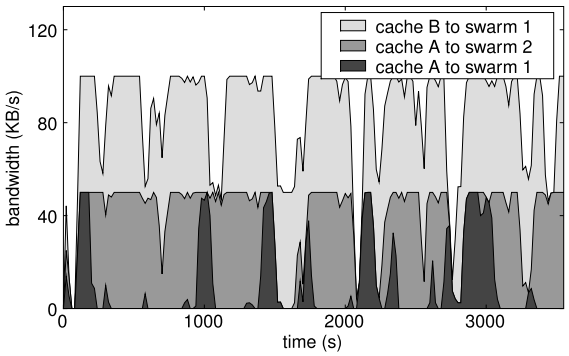

Figure 12: Cache bandwidth to a pair of swarms. Two swarms of similar size achieve comparable aggregate bandwidth even though cache B does not possess content for swarm s_2. Cache A gives more bandwidth to s_2 (medium gray area) than to s_1 (dark gray area) to compensate.

width distribution. Content originates from two cache servers, simulating a distributed cache of movie files. Cache server A contains a copy of every movie; cache server B has only a random 50% of the library's files to mitigate load on cache server A. Both servers upload content at 50 KBytes/s, scaled down from a realistic datacenter bandwidth due to the number of simulated downloaders. We show the system-wide aggregate bandwidth of each protocol over a one-hour run (Figure 11).

Both BitTorrent and the global rarest policy ignore swarm- and system-wide performance. The result is that singleton swarms and small swarms from the long tail receive a high proportion of the servers' bandwidth. Such peers are unable to forward blocks as rapidly as members of larger swarms, resulting in low aggregate bandwidth for pure peer-to-peer approaches. V-Formation achieves 66% higher aggregate bandwidth than BitTorrent.

V-Formation differs from Antfarm in both the time to converge on an allocation of bandwidth and the aggregate bandwidth itself after convergence. Since V-Formation uses lightweight probes to determine bandwidth allocations, it reaches a stable allocation of bandwidth four times faster than Antfarm. Antfarm instead relies on response curves to assess swarms, which requires the Antfarm coordinator to remain at a particular bandwidth allocation for a longer time before it becomes apparent which swarms benefit most from bandwidth.

Further, after the protocols converge, V-Formation and Antfarm achieve different aggregate bandwidths due to constraints imposed by individual peers' bandwidths and swarm memberships. From the experiment shown in Figure 11, consider two swarms s_1 and s_2 for which peers measure comparable CPM values, where s_1's movie is cached on both servers and s_2's movie is only cached on server B. In this scenario, the Antfarm coordinator measures a response curve for each of the two swarms and determines that both swarms should receive approximately equal bandwidth from the servers. However, Antfarm is unable to realize this allocation due to the constraints of peers' upload capacities and their swarm memberships. The coordinator's greedy solution to the assignment problem results in suboptimal performance.

In contrast, the V-Formation coordinator uses each individual peer's benefit to arrive at a more efficient allocation of bandwidth. A representative run of the experiment shows that both swarms receive comparable bandwidth from the cache servers despite the imbalance in the cache servers' content (Figure 12). Cache server B can only upload to swarm s_1 because it only contains one of the movies, as depicted by the light gray area. Cache server A, on the other hand, possesses both movies, so it can upload blocks to either swarm. The dark gray and medium gray areas indicate that swarm s_2 receives more bandwidth from cache server A than s_1. Fluctuations in the caches' bandwidth is due to allocating bandwidth to other swarms in the system as measured CPM values vary over time. Averaged over eight runs of the experiment, cache server A uploads a majority of its bandwidth to the swarm with only one source (with an average of 124 peers) and only 12.6 KBytes/s to the swarm also sourced by cache server B (with an average of 120 peers) in order to offset cache server B's asymmetric contribution of 42.1 KBytes/s to the swarm sourced by both servers. Interactions among swarms similar to the two swarms we have examined account for V-Formation's 30% higher system-wide bandwidth over Antfarm.

To provide further insight into V-Formation's bandwidth allocation algorithm, we empirically show how V-Formation allocates bandwidth to competing swarms. We set up a scenario where peer p_1 possesses content for two swarms s_1 and s_2 with 25 downloaders each, and peer p_2 possesses content for s_1 and a small swarm s_3 with only three downloaders (Figure 6). All peers have upload and download capacities of 50 KBytes/s. The two peers p_1 and p_2 converge on an efficient, stable allocation (Figure 13). The left-hand graph shows p_1's CPM values for its swarms over time; the right-hand graph shows the same for p_2. When the simulation begins, p_1 and p_2 probe their respective swarms to obtain initial CPM values. They both measure comparable CPM values for s_1, which are similar to p_1's initial measurement of s_2. p_2 quickly discovers that s_3 receives little benefit from its block uploads, so it allocates its bandwidth to s_1. The competition that p_2's uploads create diminishes p_1's CPM value for s_1, causing it to dedicate its bandwidth to s_2. This sequence of events matches the expected behavior of the V-Formation protocol, with peer p_1 preferentially providing bandwidth to s_2 as s_1 can be sourced by both p_1 and p_2. The periodic fluctuations of measured CPM values are the result of probing; CPM values go stale after five minutes of no activity, at which time peers probe swarms for new block propagation bandwidths.

In order to differentiate peers' effects on competing swarms, the coordinator adjusts the block propagation measurement time interval τ for each peer. We measure a single peer p's block propagation bandwidths for three competing swarms s_1, s_2, and s_3, as well as the aggregate bandwidth that results from using each value. Swarm s_1 has 30 downloaders, and s_2 and s_3 each have 20 downloaders. Swarm s_3 has an additional source of content whose uploads compete with p's uploads. All peers have upload and download capacities of 50 KBytes/s. The coordinator chooses a value for τ that enables it to differentiate among swarms (Figure 14). The left-hand graph shows the resulting CPM values as a function of the coordinator's choice of τ. All three swarms exhibit comparable CPM values for small τ, but with sufficiently large τ, the swarms' different behaviors become prominent. The right-hand graph shows the system-wide aggregate bandwidth that results from each value of τ, with values 30 seconds and above providing approximately equal aggregate bandwidth. The vertical dashed line in the graph indicates the coordinator's dynamic choice of τ for determining p's bandwidth allocation. The coordinator chooses the smallest τ such that it is able to distinguish p's contribution to the swarms that receive the most benefit from p's blocks. The selected τ is safely above 30 seconds, enabling the system to operate at a high aggregate bandwidth.

The next two experiments evaluate how the CPM enables V-Formation to converge on a stable allocation of bandwidth in the

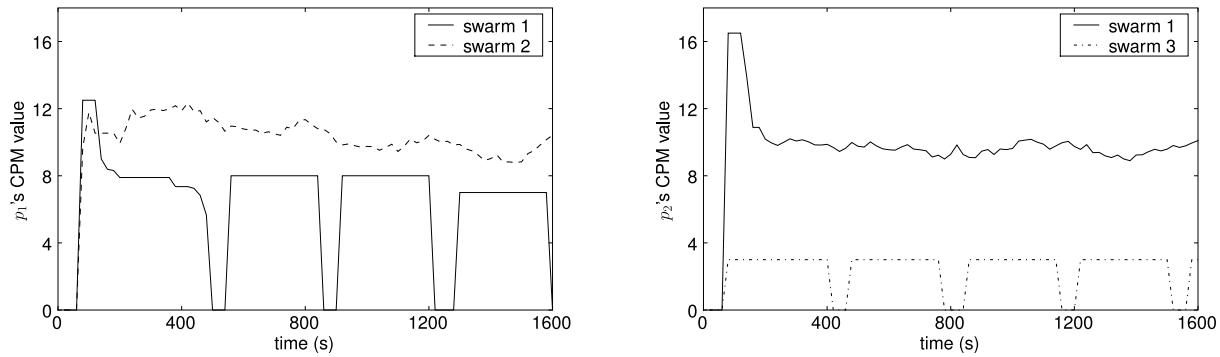

Figure 13: CPM values for competing swarms. Peer p_1 has cached content for two large swarms s_1 and s_2 (left), and peer p_2 has content for s_1 and a small swarm s_3 (right). p_2 creates competition for block uploads in s_1, causing p_1 to upload to s_2.

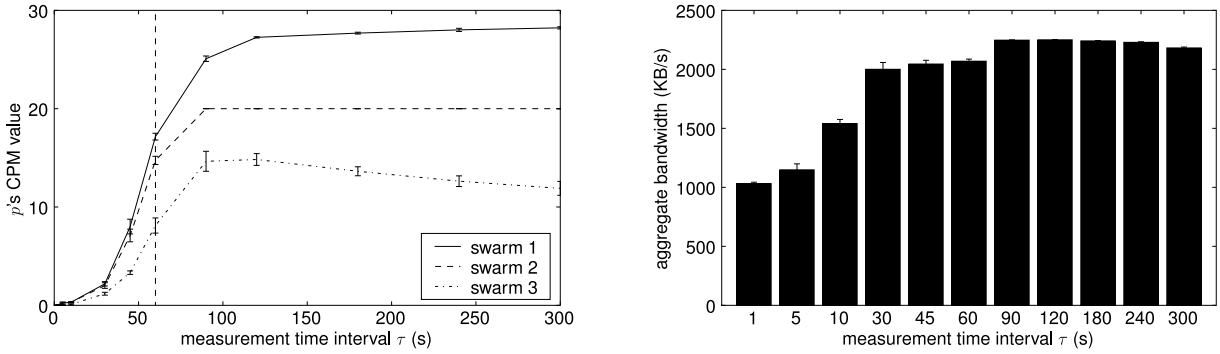

Figure 14: Sensitivity to the measurement time interval. Competing swarms are indistinguishable for small τ. Sufficiently large τ enables peer p to discover the swarm that receives the most benefit from its blocks. The vertical dashed line shows the coordinator's choice of τ based on the measurements. Error bars indicate 95% confidence.

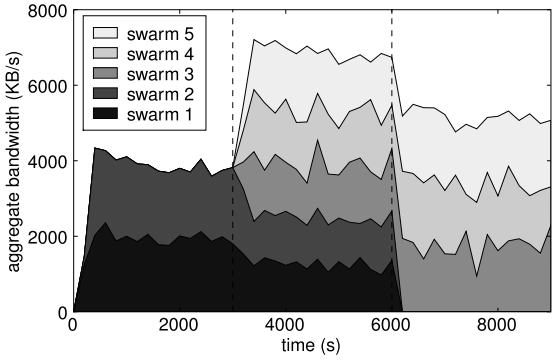

Figure 15: Convergence and stability. Three cache servers concurrently upload to two, five, and three identical swarms, indicated by shaded areas. The protocol adapts quickly to the changes introduced at the dotted lines, achieving equal aggregate bandwidth across swarms in each interval.

presence of churn. In both experiments, three cache servers initially provide content to two identical swarms s_1 and s_2, each with 50 peers. Again, peers have asymmetric upload and download links drawn from the same measured BitTorrent distribution. Due to symmetry, both swarms receive an even split of the servers' bandwidth and achieve equal aggregate bandwidth (Figure 15). At 3000 seconds, identical swarms s_3, s_4, and s_5 simultaneously join. The cache servers adjust their allocations to maintain proportional swarm aggregate bandwidths by slightly sacrificing the aggregate bandwidths of s_1 and s_2 to bootstrap the new swarms. The caches' bandwidth is too small to saturate peers' upload capacities across the five swarms, but V-Formation manages to converge on equal aggregate bandwidths despite limited cache bandwidth. At 6000 seconds, all peers in s_1 and s_2 leave simultaneously; the remaining swarms each achieve an equal increase in aggregate bandwidth. V-Formation adapts within five minutes to dramatic changes in swarm memberships by choosing an appropriate block propagation time measurement interval τ that enables hosts to efficiently detect the swarms that benefit most from their bandwidth (Figure 16). Cache servers continuously shift their allocations based on changing CPM values, and swarms dampen the effect that the fluctuating allocations have on the swarms' aggregate bandwidths.

7. RELATED WORK

Past work on content distribution falls into two categories: content distribution networks in general and peer-to-peer swarming systems in particular.

Content distribution networks leverage distributed hosts to alleviate load at content origin servers and to improve latency and bandwidth performance for clients. Akamai [18] is an infrastructure-based CDN that many content providers use to distribute their content. ECHOS [20] proposes introducing infrastructure at the Internet's periphery to cache content near clients. Choffnes et al. [11] reduce cross-ISP traffic in peer-to-peer systems by harvesting data

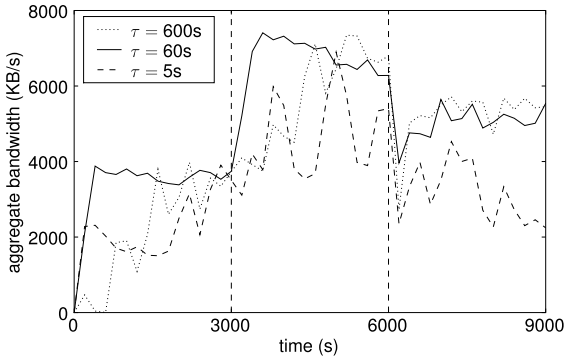

Figure 16: Time measurement interval and churn. Swarm memberships change drastically at the two vertical dashed lines. A measurement time interval τ that is too large or too small results in suboptimal performance. V-Formation chooses τ to maximize aggregate bandwidth.

from existing CDNs for locality information. Numerous CDNs rely on consistent hashing within distributed hash tables to replicate content for faster downloads and higher availability [14, 38]; in general, cooperative web caching dissipates load at servers [8,15, 17, 37]. CoBlitz [26] uses proxies to disseminate content in pieces and assemble it near downloaders. V-Formation uses the CPM to make efficient use of bandwidth from all hosts, including server-class machines that cache content in the network.

Swarming systems leverage bandwidth contributed by peers in a mesh network for increased scalability and resiliency. BitTorrent [9], a swarming protocol that contributes to much of the Internet's traffic [25, 36], encourages peers to contribute their bandwidth using a bilateral tit-for-tat mechanism. Much prior work has measured and modeled BitTorrent's performance and tit-for-tat strategy, showing that it is susceptible to Sybil attacks [12] and is easily gamed [21, 27, 34]. New protocols have responded to BitTorrent's incentive mechanism that emphasize fairness using cryptographic- or auction-based mechanisms [24, 33, 35]. Such protocols encourage peers to contribute bandwidth in order to be rewarded with a proportional or equal number of content blocks. Fairness is not a primary concern of V-Formation; while some protocols rely on fairness for incentive compatibility, V-Formation enforces behavior with its coordinator, freeing it to maximizing system-wide performance instead.

Many swarming protocols rely on cryptographic virtual currencies to incentivize peers to contribute bandwidth. Dandelion [35] and BAR Gossip [22] employ tamper-resistant protocols that ensure that peers are honest. Similarly, microcurrencies [10,23,30,39] rely on cryptographic tokens exchanged between parties to enforce correct behavior. V-Formation extends the lightweight token protocol introduced by Antfarm [32] to incentivize peers to follow the prescribed protocol and risk being discovered and blacklisted if they deviate.

One-Hop Reputations [28] and Contracts [29] use propagation trees of depth one to increase peer accountability in bulk download and live streaming systems, respectively. The protocols introduce peer incentivization strategies that outperform BitTorrent's bilateral tit-for-tat approach. The CPM is defined by content propagation trees that measure the benefit of a host to particular swarms.

Freedman et al. [13] propose a protocol that manages downloads in a multi-file system. Peers use a distributed algorithm to determine the relative values of content files and the market-based supply and demand for content blocks at each peer according to available network resources. The protocol enables ISPs to set a cost on transferring data over specific network links. It enables peers to adjust block prices based on local content demand; V-Formation, on the other hand, takes a holistic view of the relative contributions that peers bring to their swarms.

The work most similar to V-Formation is Antfarm [32], a content distribution system that measures swarms' responses to seeder bandwidth in order to optimize its uploads among competing swarms. Antfarm does not efficiently allocate bandwidth from multiple origin servers or from in-networking cache servers, which belong to multiple, overlapping swarms. In contrast, V-Formation accounts for bandwidth from all hosts based on real-time measurements using the CPM.

8. CONCLUSIONS

This paper introduced the Content Propagation Metric, which enables content distribution systems to make efficient use of bandwidth from all sources, including content distributors' origin servers, in-network caches, and clients. The CPM captures how quickly content propagates throughout swarms, providing hosts a basis of comparison that they can use to preferentially upload content to swarms that exhibit high marginal utility. A new content distribution system called V-Formation, based on a hybrid, peer-assisted architecture, uses the CPM to allocate hosts' bandwidth among competing swarms. V-Formation achieves a global performance goal of maximizing system-wide aggregate bandwidth by using the CPM to guide hosts toward an efficient use of resources. The CPM naturally handles dynamic swarm and peer behavior, and enables V-Formation to stabilize quickly on an efficient allocation of bandwidth. The flexibility of the CPM makes V-Formation efficient and scalable, rendering it well-suited to address the increasing demand for online media content.

9. ACKNOWLEDGMENTS

We would like to thank our shepherd Michael J. Freedman and the anonymous reviewers for their insights and comments. This material is based upon work supported by the National Science Foundation under Grant No. 0546568 and the 2009 Google Fellowship in Distributed Systems.

10. REFERENCES

[1] BitTorrent. http://bittorrent.com.

[2] FlixQ.com – Videosharing For The Masses. http://flixq.com.

[3] Ipoque – Internet Studies http://www.ipoque.com/resources/internet-studies.

[4] Velocix – New Generation Content Delivery Network http://www.velocix.com.

[5] Verivue – OneVantage Content Delivery Solution http://verivue.com.

[6] A. C. Bavier, M. Bowman, B. N. Chun, D. E. Culler, S. Karlin, S. Muir, L. L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support For Planetary-scale Network Services. *Symposium on Networked System Design and Implementation*, San Francisco, CA, March 2004.

[7] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain Approach To Reliable Distribution Of Bulk Data. *SIGCOMM Conference*, Vancouver, Canada, August 1998.

[8] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A Hierarchical Internet Object

Cache. *USENIX Annual Technical Conference*, San Diego, CA, January 1996.

[9] B. Cohen. Incentives Build Robustness In BitTorrent. *Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, May 2003.

[10] J. Camp, M. Sirbu, and J. D. Tygar. Token And Notational Money In Electronic Commerce. *USENIX Workshop on Electronic Commerce*, New York, NY, July 1995.

[11] D. R. Choffnes and F. E. Bustamante. Taming The Torrent: A Practical Approach To Reducing Cross-ISP Traffic In Peer-to-Peer Systems. *SIGCOMM Conference*, Seattle, WA, August 2008.

[12] J. R. Douceur. The Sybil Attack. *International Workshop on Peer-to-Peer Systems*, Springer Lecture Notes in Computer Science 2429, Cambridge, MA, March 2002.

[13] M. J. Freedman, C. Aperjis, and R. Johari. Prices Are Right: Managing Resources And Incentives In Peer-Assisted Content Distribution. *International Workshop on Peer-to-Peer Systems*, Tampa Bay, FL, February 2008.

[14] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing Content Publication With Coral. *Symposium on Networked System Design and Implementation*, San Francisco, CA, March 2004.

[15] S. Gadde, J. S. Chase, and M. Rabinovich. Web Caching And Content Distribution: A View From The Interior. *Computer Communications*, 24(2), May 2001.

[16] C. Huang, J. Li, and K. W. Ross. Can Internet Video-on-Demand Be Profitable? *SIGCOMM Conference*, 2007.

[17] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching With Consistent Hashing. *International World Wide Web Conference*, Toronto, Canada, May 1999.

[18] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent Hashing And Random Trees: Distributed Caching Protocols For Relieving Hot Spots On The World Wide Web. *ACM Symposium on Theory of Computing*, El Paso, TX, May 1997.

[19] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent Is An Auction: Analyzing And Improving BitTorrent's Incentives. *SIGCOMM Conference*, Seattle, WA, August 2008.

[20] N. Laoutaris, P. Rodriguez, and L. Massoulie. Echos: Edge Capacity Hosting Overlays Of Nano Data Centers. *ACM SIGCOMM: Computer Communication Review*, 38, January 2008.

[21] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding In BitTorrent Is Cheap. *Workshop on Hot Topics in Networks*, Irvine, CA, November 2006.

[22] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. Bar Gossip. *Symposium on Operating System Design and Implementation*, Seattle, WA, November 2006.

[23] M. Manasse. The Millicent Protocol For Electronic Commerce. *USENIX Workshop on Electronic Commerce*, New York, NY, August 1995.

[24] T.-W. Ngan, D. S. Wallach, and a. P. Druschel. Enforcing Fair Sharing Of Peer-to-Peer Resources. *International Workshop on Peer-to-Peer Systems*, Springer Lecture Notes in Computer Science 2735, Berkeley, CA, February 2003.

[25] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The BitTorrent P2P File-Sharing System: Measurements And Analysis. *International Workshop on Peer-to-Peer Systems*, Springer Lecture Notes in Computer Science 3640, Ithaca, NY, February 2005.

[26] K. Park and V. S. Pai. Scale And Performance In Coblitz Large-file Distribution Service. *Symposium on Networked System Design and Implementation*, San Jose, CA, May 2006.

[27] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do Incentives Build Robustness In BitTorrent? *Symposium on Networked System Design and Implementation*, Cambridge, MA, April 2007.

[28] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One Hop Reputations For Peer To Peer File Sharing Workloads. *Symposium on Networked System Design and Implementation*, San Francisco, CA, April 2008.

[29] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe. Contracts: Practical Contribution Incentives For P2P Live Streaming. *Symposium on Networked System Design and Implementation*, San Jose, CA, April 2010.

[30] T. Poutanen, H. Hinton, and M. Stumm. Netcents: A Lightweight Protocol For Secure Micropayments. *USENIX Workshop on Electronic Commerce*, Boston, MA, August 1998.

[31] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips. *International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[32] R. S. Peterson and E. G. Sirer. Antfarm: Efficient Content Distribution With Managed Swarms. *Symposium on Networked System Design and Implementation*, Boston, MA, April 2009.

[33] A. Sherman, J. Nieh, and C. Stein. Fairtorrent: Bringing Fairness To Peer-to-Peer Systems. *Conference on emerging Networking EXperiments and Technologies*, Rome, Italy, December 2009.

[34] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding In BitTorrent Networks With The Large View Exploit. *International Workshop on Peer-to-Peer Systems*, Bellevue, WA, February 2007.

[35] M. Sirivianos, X. Yang, and S. Jarecki. Dandelion: Cooperative Content Distribution With Robust Incentives. *USENIX Annual Technical Conference*, Santa Clara, CA, June 2007.

[36] S. Saroiu, P. K. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis Of Internet Content Delivery Systems. *Symposium on Operating System Design and Implementation*, Boston, MA, December 2002.

[37] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On The Scale And Performance Of Cooperative Web Proxy Caching. *Symposium on Operating Systems Principles*, Kiawah Island, SC, December 1999.

[38] L. Wang, K. Park, R. Pang, V. S. Pai, and L. L. Peterson. Reliability And Security In The Codeen Content Distribution Network. *USENIX Annual Technical Conference*, Boston, MA, June 2004.

[39] P. Wayner. *Digital Cash: Commerce On The Net*. Morgan Kaufmann, April 1996.