EFFICIENT CONTENT DISTRIBUTION WITH MANAGED SWARMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ryan S. Peterson

January 2012

EFFICIENT CONTENT DISTRIBUTION WITH MANAGED SWARMS

Ryan S. Peterson, Ph.D.

Cornell University 2012

Content distribution has become increasingly important as people have become more reliant on Internet services to provide large multimedia content. Efficiently distributing content is a complex and difficult problem: large content libraries are often distributed across many physical hosts, and each host has its own bandwidth and storage constraints.

Peer-to-peer and peer-assisted download systems further complicate content distribution. By contributing their own bandwidth, end users can improve overall performance and reduce load on servers, but end users have their own motivations and incentives that are not necessarily aligned with those of content distributors. Consequently, existing content distributors either opt to serve content exclusively from hosts under their direct control, and thus neglect the large pool of resources that end users can offer, or they allow end users to contribute bandwidth at the expense of sacrificing complete control over available resources.

This thesis introduces a new approach to content distribution that achieves high performance for distributing bulk content, based on *managed swarms*. Managed swarms efficiently allocate bandwidth from origin servers, in-network caches, and end users to achieve system-wide performance objectives. Managed swarming systems are characterized by the presence of a logically centralized coordinator that maintains a global view of the system and directs hosts toward an efficient use of bandwidth. The coordinator allocates bandwidth from

each host based on empirical measurements of swarm behavior combined with a new model of swarm dynamics. The new model enables the coordinator to predict how swarms will respond to changes in bandwidth based on past measurements of their performance.

In this thesis, we focus on the global objective of maximizing download bandwidth across end users in the system. To that end, we introduce two algorithms that the coordinator can use to compute efficient allocations of bandwidth for each host that result in high download speeds for clients.

We have implemented a scalable coordinator that uses these algorithms to maximize system-wide aggregate bandwidth. The coordinator actively measures swarm dynamics and uses the data to calculate, for each host, a bandwidth allocation among the swarms competing for the host's bandwidth. Extensive simulations and a live deployment show that managed swarms significantly outperform centralized distribution services as well as completely decentralized peer-to-peer systems.

BIOGRAPHICAL SKETCH

Ryan S. Peterson was born in Lafayette, Indiana, where he lived his first two years at Purdue University previewing graduate student life. At age two, Ryan moved with his family to Tucson, Arizona, where he spent his childhood building forts among cacti. Another move to Princeton, New Jersey in 1998 paved the way to an undergraduate career at Princeton University, where Ryan graduated with Honors in 2005 with a Bachelor of Science in Engineering degree in computer science and a certificate in linguistics. Despite enrolling in several computer science courses taught by his father, Ryan remained in the field and attended Cornell University to pursue a Ph.D. At Cornell, Ryan developed an interest in large distributed systems and content distribution, which he worked on under the advisement of Emin Gün Sirer. Ryan is the 2009 recipient of the Google Fellowship in Distributed Systems, and he plans to begin working at Google after he defends this thesis.

to Mom, Dad, and Grandma

ACKNOWLEDGEMENTS

This thesis was a lot of work, and it took a lot of time—too much work and too much time to thank everyone who supported me on this journey. But I'll try.

First and foremost, I would like to thank my advisor Emin Gün Sirer. Gün has incredible energy, which fueled me when I was excited, and motivated me when I was in a rut. More than that, Gün taught me to think clearly: to deeply understand a problem, to search for a solution, and to share my findings.

Hakim Weatherspoon and David Bindel provided invaluable guidance on my research and this thesis. Hakim has contributed considerable time and energy, from helping with the original Antfarm system, to donating computational resources for experiments, to providing comments on this thesis. David's expertise on mathematical optimization helped formalize managed swarms and the problem of efficiently placing content among caches.

I spent much of my time in graduate school making music. I can't express how at home I felt with Cornell's choral community. For that, I would like to thank Scott Tucker, John Rowehl, and the Cornell University Glee Club, Chorus, and Chamber Singers. I made friends and memories through singing that I'll keep for a very long time.

I've always considered my close friends to be my family away from home. I'm lucky to have such a big family. Movie nights, the Grindhouse, the Dollhouse, 130 Crunk, the syslab, Pvt. Bits, waterfall hunting, whisk(e)y tastings....

Finally, I want to thank my real family. Mom and Dad, you've always supported me in whatever I've wanted to do. Mom, you're such an understanding person. Dad, it means a lot to me that you include me in your computer science circle. I look forward to more mini family reunions at conferences. And Grandma, I'm always happy to hear your voice. Your enthusiasm is contagious!

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Content distribution is a critical problem. Large multimedia content accounts for the majority of all Internet traffic [4], video alone making up approximately 40% of traffic, excluding peer-to-peer file sharing [36]. Today, subscription-based video-on-demand accounts for more than 32% of Internet traffic in North America during peak hours, resulting in higher costs than postage for DVD-by-mail service [93, 10]. Such video-on-demand services are rapidly increasing in popularity, causing video traffic to double every two and a half years [36]. Methods to keep up with the high demand for online video while minimizing costs of distribution are necessary.

To address the growing demand for multimedia content, existing content distribution systems employ a wide range of techniques and can be divided into three categories based on their architectures: client-server, peer-to-peer, and hybrid systems. These architectures differ in their placement of content as well as their costs, scalability, and performance in different deployment scenarios.

In the *client-server* approach, clients download content directly from servers operating under the content owner's control. This logical centralization gives the content distributor full control over content downloads: accounting and admission control can be handled by the servers, clients can be prioritized, and bandwidth can be dedicated to desired transfers at fine granularity. The relative ease with which the content distributor can control a client-server system is critical for a commercial service. The drawback of the client-server approach is that the entire burden of distribution rests on the content owners, requiring large up-front costs to deploy and high bandwidth costs to operate. Today's

content distributors with repositories of multimedia content, such as YouTube, Akamai, Netflix, and Facebook, operate large datacenters to distribute that content to their clients, and incur significant bandwidth costs in the process. Consequently, the client-server approach can be prohibitively expensive, requiring content providers to seek alternative methods to distribute their content [22].

Peer-to-peer architectures offer an alternative to the client-server approach by shifting the cost of content distribution to the end users and their ISPs. In such systems, clients interested in downloading content contribute their own upload bandwidth to the system in exchange for service. As a result, peer-to-peer systems decrease the bandwidth demand on content originators and enable large-scale distribution of content originating at end users. Existing peer-to-peer systems demonstrate the scalability of the peer-to-peer approach, the most popular of which is BitTorrent [2], which accounts for 43–77% of Internet traffic depending on geographic region [4]. However, peer-to-peer protocols like BitTorrent were designed primarily to avoid centralization and to provide fault tolerance; they do not seek to achieve system-wide objectives, which can result in poor performance. Complete decentralization comes at a cost: nodes in a peer-to-peer system typically act solely on local information, which can lead to emergent system behaviors that make inefficient use of available resources. As a result, peer-to-peer systems have difficulty providing quality-of-service guarantees. Thus, completely decentralized protocols are ill-suited to meet specific system-wide performance objectives, leading to suboptimal performance.

Recently, *hybrid* architectures have emerged that combine centralized components with peer-to-peer transfers. In a hybrid architecture, peers assist a centralized server in the download process in order to decrease the burden placed

on content distributors and to offset bandwidth costs. Measurement studies have shown that hybrid approaches to content distribution could significantly reduce the cost of distribution [55, 33, 67]. To date, systems based on hybrid architectures have primarily focused on reducing the burden that clients place on content servers. However, in addition to shifting bandwidth costs, the centralized component of a hybrid architecture provides a new, untapped opportunity to aggregate system-wide activity and use it to make informed decisions for improving performance. In contrast to completely decentralized systems, where controlling the behavior of the network is difficult, a centralized component has the potential to drastically improve system-wide performance by managing and controlling peers' resources with informed decisions. Yet, few existing hybrid systems leverage centralization to manage peer behavior, and those that do rely on heuristics that lack performance guarantees.

This thesis proposes a novel approach to content distribution for distributing a large set of files to a potentially very large set of clients, through *managed swarms*. Managed swarms endow swarms with centrally computed policies and control so that they can achieve system-wide performance objectives. They incorporate a logically centralized component, called a *coordinator*, to a hybrid architecture that otherwise operates according to decentralized policies. The coordinator controls the behavior of hosts based on swarm dynamics and desired performance criteria. By observing the behavior of swarms, the coordinator steers hosts toward efficient use of available resources in order to maximize a desired performance metric.

The coordinator's global view of the system enables it to optimize network resources for various metrics to meet application-specific performance criteria.

For instance, a system designed to distribute a binary executable to a set of hosts in multiple datacenters might value shipping the file to all target hosts in its entirety as quickly as possible. In such a scenario, the coordinator's goal might be to minimize the download time for the slowest host. In contrast, a video streaming website might aim to maximize the number of clients that achieve a target download bandwidth required for smooth playback.

This thesis primarily focuses on large, bulk downloads, as in the case of multimedia file distribution, where the goal is to maximize global network utilization. In a realistic multimedia file distribution setting, the coordinator is tasked with maximizing system-wide aggregate bandwidth. It does so by judiciously allocating bandwidth, the scarce resource in a content distribution system, from contributing sources. These sources comprise content originators, cache servers, and end users. The problem is further complicated by three issues: content servers may only have a partial set of the content library, cache servers may possess only subsets of the content, and end users may belong to multiple, overlapping swarms.

One way to approach this problem is to analytically model the behavior of swarms. Previous work has developed mathematical models to gain insights into swarm behavior and improve download speeds [108, 83]. Such work predicts peer and user behavior using mathematical distributions to model peer arrivals and departures, peer connectivity, and block transfers. Mathematical models provide a strong foundation for understanding the effects of well-defined properties of swarms and peers on system behavior. However, such models often make simplifying assumptions about user behavior and are difficult to parameterize. Consequently, they cannot account for large variations in

peer arrival rate that result from flash crowds. Peer departures are similarly difficult to model, as they are largely independent of download completion [59]. However, measurement studies have found that altruistic peers that continue to contribute bandwidth after completing their own downloads have an enormous impact on performance [56, 95]. As a result, existing mathematical models require extensive, time-consuming measurements in order to accurately predict swarm behavior in realistic deployments.

This thesis proposes a new model for swarm behavior that replaces hard-to-extract parameters and unrealistic assumptions about user behavior with frequent, lightweight empirical measurements. The model enables accurate prediction of a swarm's aggregate bandwidth, defined as the sum of all member peers' download rates, as a function of the bandwidth injected into the swarm. This model captures two critical behaviors that are essential for accurately estimating the marginal benefit of additional bandwidth to swarms. Specifically, a swarm whose peers have spare upload bandwidth benefit greatly from receiving rare data blocks because it enables its peers to rapidly forward the blocks throughout the swarm. On the other hand, a swarm whose peers have saturated their uplinks are unable to forward additional blocks to neighboring peers; increasing the bandwidth that such a swarm receives from a cache server has limited impact on the swarm's aggregate bandwidth. Consequently, the model enables the coordinator to predict which swarms will exhibit larger increases in aggregate bandwidth if supplied with additional bandwidth.

The logically centralized coordinator, coupled with a novel wire-level protocol, uses our model of swarm dynamics to optimize peer behavior. The coordinator performs three basic tasks. First, it observes swarms to gather information

about swarm behavior, which it uses to model each swarm's response to bandwidth. Second, the coordinator uses the swarm models to compute an efficient allocation of bandwidth. Lastly, the coordinator enacts its decisions by forcing peers to operate according to its computed allocations.

The coordinator's role exposes a fundamental tradeoff between its involvement in guiding peer behavior and its scalability with respect to bandwidth, CPU, and memory. At one extreme, the coordinator could schedule every single block transfer between peers. Coordination at such fine granularity would require significant resources to track each peer's exact bandwidth usage and download state, in addition to peer connectivity and network conditions. Further, even with this detailed data, computing a complete schedule of transfers would pose a large computational burden on the centralized coordinator. At the other extreme, swarms that operate with no guidance from the coordinator degenerate into unmanaged, decentralized swarms, where peer behavior relies solely on local information, and the emergent behavior of the network is uncontrolled.

This thesis advocates combining the coordinator's guidance on allocating bandwidth across competing swarms with clients' local decisions for scheduling transfers among peers within the same swarm. Thus, the coordinator manages peers at the granularity of a swarm and controls inter-swarm bandwidth allocation, while peers perform standard peer-to-peer optimizations for intra-swarm behavior. Optimizations in the latter category include optimistic unchoking for peer discovery, locality-based peer selection, tit-for-tat for bandwidth allocation, and rarest first for block selection, each of which has been shown to improve performance within a single swarm [17, 71, 12]. Overall,

the coordinator ensures that peers allocate bandwidth to the swarms that make most efficient use of bandwidth while peers make decisions that lead to efficient propagation of content that swarms have already received.

The coordinator presented in this thesis guides peers based on data that it acquires using a novel token protocol. This protocol enables the coordinator to make its allocation decisions with minimal network overhead. It relies on small, one-use *tokens* minted by the coordinator that function like a virtual currency. Peers exchange tokens with each other for content blocks and return spent tokens to the coordinator to acquire fresh tokens, thus divulging their recent activity.

The coordinator distributes its resource allocation decisions to peers and uses its global view of the system to ensure that they follow its guidance. The coordinator sends each peer a bandwidth allocation for dividing the peer's upload bandwidth among its competing swarms based on the coordinator's computations. It then polices the system by observing individual peers' activity. Because all peers report to the coordinator regularly, the coordinator has the authority to deny access to any peer that it detects is defecting from the prescribed allocation. The coordinator simply invalidates such a peer's unspent tokens for downloading content blocks, denies it future fresh tokens, and informs other peers of the defector's status.

The token protocol is designed to be scalable and efficient to ensure that peers can interact with the coordinator frequently enough to maintain an accurate model of each swarm's behavior. Consequently, our implementation does not rely on heavyweight cryptographic operations to exchange blocks or tokens. Instead, small tokens minted and verified by the coordinator enable the coordi-

nator to process spent tokens with minimal bandwidth and CPU overhead, resulting in a highly scalable system. An alternative implementation could leverage an encryption scheme to provide additional guarantees, such as user authentication, by forcing peers to establish untappable channels to each other, sign tokens, and prove their identity to the coordinator [115]. Such a scheme may be appropriate for certain deployments of managed swarms where security is the primary concern. In contrast, our implementation prioritizes efficient token exchanges, resulting in a practical, lightweight protocol for obtaining a coarse-grain view of the system.

Based on measurements obtained with the token protocol, we introduce two algorithms that the coordinator can use to compute bandwidth allocations. Both algorithms aim to maximize the system-wide aggregate bandwidth across all swarms, but they differ in the deployment scenarios that they target. The first algorithm, Antfarm [105], applies to deployments with a single, logically centralized source of content, called a *seeder*. The seeder possesses all the content and supplies bandwidth to all swarms, as would be the case for a single datacenter providing a large multimedia library to many clients. Antfarm computes the optimal allocation of the seeder's bandwidth among competing swarms to minimize download times. The second algorithm, V-Formation [107], targets a more realistic range of deployment scenarios. Specifically, it targets deployments with caches in the network, each of which might possess only a partial subset of the content available in the system, in addition to end users that belong to multiple, potentially overlapping swarms. Large-scale video distributors, such as YouTube, where clients request movies from a large set of datacenters, exemplify V-Formation's target deployment. Both algorithms achieve

high performance by instructing hosts how to divide their bandwidth among competing swarms.

Introducing a logically centralized component to an otherwise decentralized system has benefits that reach beyond achieving high aggregate bandwidth. Centralization enables a content distributor to provision for quality-of-service guarantees by ensuring that particular swarms or peers receive a target bandwidth; to aggregate statistics from peers for accounting and tracking; and to specify policies, metrics, and optimizations at a single location based on a global view of the system's activity. QoS guarantees enable content distributors to apportion hosts' bandwidth to account for service-level agreements or target multimedia bitrates. To enact such guarantees and other policies that content distributors define, the coordinator aggregates data on peer behavior and uses it to compute policy-specific bandwidth allocations. Centralized accounting empowers the coordinator to detect that peers act according to content distributors' policies, and, through its involvement in peer-to-peer block transfers, enables the coordinator to prohibit misbehaving peers from disrupting service. Overall, centralization offers content distributors a degree of control over their content and the bandwidth of participating hosts that is difficult to achieve in decentralized systems. However, centralization comes at a cost: logical coordination can lead to a performance bottleneck, and the implementation of a scalable coordinator is critical to the success of a hybrid system. This thesis demonstrates that the benefits of centralization can outweigh the costs and that a judicious implementation can scale to large deployments of millions of peers and hundreds of thousands of swarms.

Overall, this thesis makes three major contributions. First, it introduces a new class of hybrid content distribution systems based on managed swarms. Managed swarms combine peer-assisted transfers with lightweight coordination to enable a logically centralized coordinator to steer the system toward a globally efficient use of available resources. Second, this thesis describes two novel algorithms that use managed swarms to make efficient use of bandwidth in order to maximize a global performance metric. The algorithms leverage swarm characteristics in order to compute efficient allocations of bandwidth. Finally, this thesis describes a full implementation and deployment of both algorithms, including practical considerations for building scalable peer-assisted download systems that manage swarms, a new wire-level protocol that provides accountability and enforcement of peer behavior, and an evaluation of the benefit of managed swarms over existing content distribution systems.

We have built, deployed, and operated a system that embodies the architecture described in this thesis [3]. The system implements a content distribution system that enables end users to contribute their own content. It is built on top of and is backward-compatible with the BitTorrent protocol. We have written new client software that executes on each peer, as well as a physically distributed coordinator that scales to thousands of peers. The coordinator operates as an augmented BitTorrent tracker: it records peer arrivals and departures, sends peers lists of swarm members to which the peers can connect in order to form a mesh network for each swarm, and manages swarms using the algorithms outlined above to maximize aggregate bandwidth. Our deployment employs cache servers to improve content availability and download speed for peers by automatically joining swarms to download popular content. The coordinator uses the same algorithms to allocate bandwidth from cache servers

as from end users. Overall, our implementation demonstrates the feasibility of managed swarms for efficiently distributing large libraries of multimedia content in realistic deployment scenarios.

The rest of this thesis is structured as follows. Chapter 2 introduces the content distribution architecture based on managed swarms, formalizes a metric for efficiency in managed swarming systems, and describes how to allocate bandwidth efficiently among swarms. Chapter 3 details our implementation of a managed swarming system for distributing user-contributed content using the Antfarm and V-Formation algorithms. It also describes alternative methods of aggregating summaries of peer activity at the coordinator and two implementations of the wire protocol for accounting and managing tokens. Chapter 4 evaluates managed swarms using both allocation algorithms. It compares system-wide performance against existing content distribution systems and examines several microbenchmarks for evaluating the systems in a range of realistic deployment scenarios and load on the coordinator. Chapter 5 discusses related work and Chapter 6 concludes.

CHAPTER 2

RESOURCE ALLOCATION IN MANAGED SWARMS

This chapter provides an overview of managed swarms and describes how they model swarm behavior and allocate content among competing swarms. It first describes the managed swarming hybrid architecture and compares it to architectures of existing content distribution systems. Then, it defines the general multi-swarm content distribution problem, which encompasses the deployment scenarios that managed swarms target. This problem formalizes efficiency for all content distribution networks that supplement bandwidth from server-class hosts with bandwidth contributed by end users. Next, this chapter presents a new model for swarm behavior that enables managed swarms to achieve high performance by finding an efficient allocation of a content distributor's resources. The new model provides a foundation for understanding the impact that a content distributor's bandwidth from deployed servers has on the overall performance of a download system. Finally, this chapter explores how the model informs hosts how to allocate their bandwidth in two classes of deployments that are common among content distribution networks today.

2.1 Architecture

Managed swarming systems augment the architectures of existing content distribution networks to maximize the performance that content distributors can achieve from their existing infrastructure. Such networks comprise potentially very large content libraries that can be fragmented across geographically distributed hosts. These hosts might be located in datacenters for efficient resource management or distributed near the edge of the network for improved latency

to end users. Other content distribution networks might task end users to locally store pieces of the content library in order to reduce storage costs for the distributors; still others might allow end users to distribute their own content from their local machines.

Managed swarming systems enable content distributors to combine their existing resources, including both servers and caches, with peer-to-peer transfers and a logically centralized coordinator that allocates system resources. All hosts that upload or download content, or *peers*, are organized into swarms. Each swarm is an unstructured mesh network that facilitates the download of a single piece of content, such as movie file. Within each swarm, peers request *blocks* of content from each other and upload blocks in response to such requests. A single, logically centralized coordinator oversees the system's operation and dictates how peers in managed swarms allocate their resources in order to achieve a global performance goal. We defer the discussion of a scalable coordinator implementation to Chapter 3.

The managed swarming architecture is defined by the logically centralized coordinator that manages the bandwidth of hosts organized into swarms. In general, the hosts that the coordinator oversees are organized according to a hybrid architecture, characterized by bandwidth contributions from servers, datacenters, caches, and end users. A deployment may omit caches or end users from the coordinator's control depending on available resources and deployment requirements. Figure 2.1 illustrates a typical deployment of a managed swarming system.

In a sample deployment, the majority of peers might consist of end users, which download content from and upload content to the swarms to which they

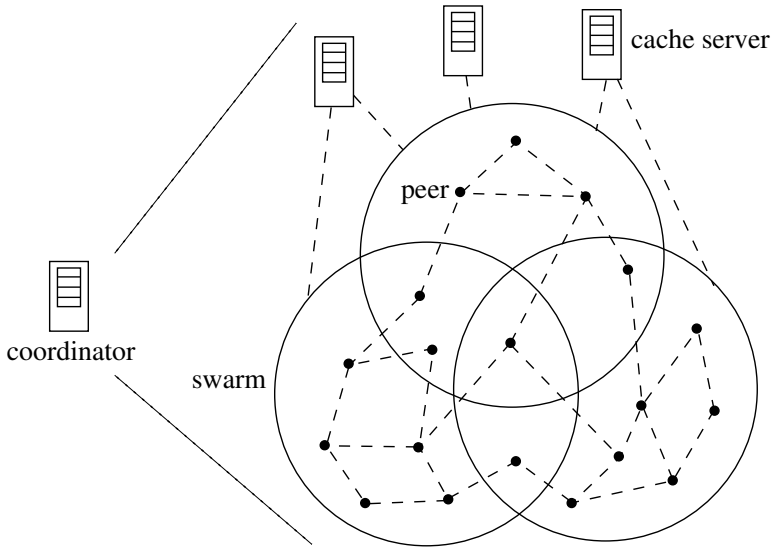

Figure 2.1: **Managed swarming architecture.** Peers, which includes all hosts that upload or download content, belong to arbitrary sets of potentially overlapping swarms. A logically centralized coordinator guides hosts towards an efficient allocation of bandwidth.

belong. End users generally join swarms for content that they are interested in downloading and to share new content that they would like to contribute to the system. Remaining peers provide additional bandwidth to swarms, and their swarm memberships are determined by the content that they possess for seeding swarms. Such peers are generally server-class hosts that operate under the control of the coordinator and the content originator, but they may also be caches deployed by ISPs or other third parties to improve network performance.

In an alternative deployment of managed swarms, all peers might reside in the network to provide an efficient download service for end users. In such a deployment, end users do not contribute their own upload bandwidth, and instead connect to proxies in the network to download content. Consequently, end users need not run specialized software to join swarms and download content. The in-network peers leverage managed swarms to distribute content within the distribution network and to position content among caches.

The management of content that is available for download further affects the design of content distribution systems. Content distribution deployments can be divided into two classes, depending on where content originates. In *closed* systems, the system operator manages all content, adding and removing the content that users are able to download. The Netflix commercial service is an example of a closed content distribution system, where clients are able to watch only the movies that the service currently offers. On the other hand, *open* content distribution systems allow users to contribute content for others to download. BitTorrent and YouTube are examples of open systems.

The managed swarming architecture is applicable to both open and closed systems. The choice of architecture affects the efficiency with which users can download content and, in open systems, contribute their own content. For example, although both YouTube and BitTorrent are open systems, YouTube requires users to upload videos in their entirety to servers, which then distribute the movies to clients using a client-server architecture. BitTorrent users sharing their own original content can upload directly to peers interested in it, but their own upload capacity can limit the speed of other peers' downloads. The algorithms proposed in this thesis provide efficient content distribution for both systems, and they take advantage of a hybrid architecture to enable efficient downloads from servers that possess content as well as from peers that contribute their own content.

The architecture subsumes previous swarming protocols such as BitTorrent, whose architecture is shown in Figure 2.2. For such deployments of multiple, non-overlapping swarms, managed swarms perform identically to BitTorrent. While BitTorrent peers may join multiple swarms, the protocol operates inde-

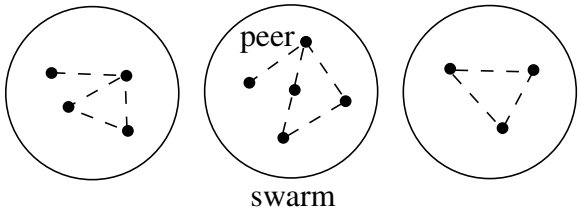

Figure 2.2: **BitTorrent architecture.** BitTorrent swarms are logically isolated; peers make bandwidth allocation decisions independently for each swarm.

pendently for each swarm. Consequently, peers dedicate bandwidth to swarms based on the behavior of individual directly-connected neighbors in the mesh network, resulting in an emergent system behavior that can be far from optimal.

2.2 Problem Definitions

The coordinator provides a logically centralized platform for guiding hosts toward an efficient use of resources based on a global performance goal. By framing a performance goal as an optimization problem, a content distributor can use the coordinator to compute an efficient utilization of resources based on the coordinator's system-wide measurements.

In this thesis, we define and describe two specific optimization problems that a coordinator can address to efficiently allocate hosts' bandwidth. Both problems aim to maximize network utilization across hosts. This overarching goal addresses a content distributor's desire to maximize the total amount of content that the system distributes to hosts. A content provider may choose to optimize for a metric other than system-wide aggregate bandwidth, but algorithms that optimize for alternative metrics are outside the scope of this thesis.

2.2.1 General Multi-swarm Content Distribution Problem

The main problem that we discuss in this thesis involves resource management in the wide variety of content distribution networks that we have observed, including open and closed systems, and systems in which the content library is centralized or distributed. This problem, called the *general multi-swarm content distribution problem*, is the central topic of this thesis. 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 places no restrictions on the location of content and peer memberships in swarms. Bandwidth can be provided by content originators, cache servers, and end users. Furthermore, content servers may only have a partial set of the content library, cache servers may possess only subsets of the content, and end users may belong to multiple, overlapping swarms.

The general multi-swarm content distribution problem, coupled with the managed swarming architecture, defines a flexible set of deployments and provides a metric to evaluate system performance for content distributors who aim to maximize network utilization.

2.2.2 Single-seeder Multi-swarm Content Distribution Problem

The single-seeder multi-swarm content distribution problem encompasses a common subset of the scenarios addressed by the general multi-swarm content distribution problem in which the content distributor maintains a logically centralized content library. This specialized problem's reduced complexity renders it easier to solve than its general variant. Consequently, the coordinator is able to efficiently find the *optimal* allocation of bandwidth in the multi-swarm environment, resulting in high performance for a content distributor that uses a modest bandwidth capacity to disseminate a small content library.

Formally, the single-seeder multi-swarm content distribution problem is defined similarly to the general variant of the problem. A deployment consists of a set of peers P, a set of swarms S, and a mapping $M : P \rightarrow S$, representing to which single swarm each peer belongs. In addition, one designated seeder t possesses all content and is a member of all swarms in S. The single-seeder multi-swarm content distribution problem, then, is to determine the upload bandwidth $U_{t,s}$ that t should allocate to swarm s for all S in order to maximize global aggregate bandwidth $\sum_{p \in P} D_p$, where D_p is the download bandwidth of peer p.

The restrictions that the single-seeder multi-swarm content distribution problem place on the deployment of a managed swarming system in turn affect the deployment architecture and the feasibility of the optimization problem. The architecture (Figure 2.3) restricts the content distributor to a single logically centralized seeder that houses the entire library; however, the seeder need not be restricted to a single physical host. Rather, the seeder itself may comprise any number of distributed replicas of the library for fault tolerance. The restric-

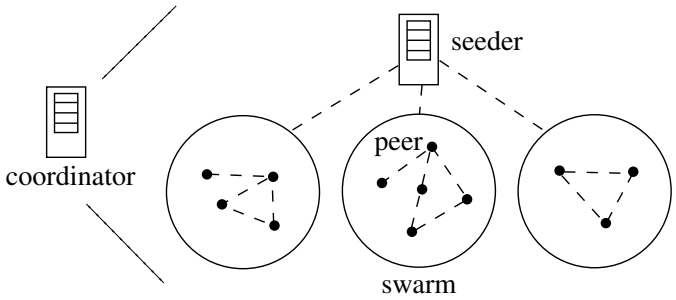

Figure 2.3: **Single-seeder architecture.** Single-seeder managed swarming deployments allocate bandwidth from a logically centralized origin server across multiple swarms.

tion of using a single seeder only stipulates that any seeder host belonging to multiple swarms possess the content library in its entirety.

2.3 Modeling Swarm Behavior

To address the multi-swarm content distribution problems, we have developed a new model of swarm behavior. The model compactly captures the properties of swarms relevant to bandwidth allocation decisions through empirical measurements.

All bandwidth in a swarming system is generated by individual block transfers between peers. A single block transfer from one peer to another benefits the block's recipient, a downloader of the swarm's content. However, the benefit of a single block transfer far exceeds the benefit that the recipient derives from the block. This is because when a peer receives a block, it enables the recipient to further propagate the block to other peers in the swarm that do not yet have it. Thus, a block transfer can potentially lead to a cascade of further transfers which creates high aggregate bandwidth across the swarm as the block propagates.

Many factors influence a swarm's ability to propagate blocks efficiently. The distribution of blocks that the swarm's peers already possess shapes the demand for new blocks. Rare blocks are more likely to propagate quickly throughout a swarm due to their high demand, but even rare blocks will propagate slowly if peers have sufficiently many blocks of high demand that compete for peers' upload bandwidth. Further, swarm size, peer connectivity, and network conditions among peers influence the ability of peers to satisfy block requests, regardless of block rarity.

Our model of swarm behavior captures a swarm's ability to generate aggregate bandwidth with a *response curve*. A response curve represents a swarm's total bandwidth as a function of the bandwidth provided to the swarm by a designated host called a seeder. Response curves fold all factors that affect a swarm's ability to propagate blocks into a single function based on direct measurements of overall swarm performance.

Response curves embody the critical properties of each swarm and have a characteristic shape—a fact that we exploit in this thesis. Figure 2.4 illustrates the characteristic form of the response curve for a homogeneous swarm with static membership; for illustration purposes in this example, peer download capacities exceed upload capacities, and the set of peers does not change throughout the download. When the seeder bandwidth is limited, the peers in the swarm have unused upload and download capacity. In this regime of operation (region A), the swarm's aggregate bandwidth increases rapidly with the seeder bandwidth, since peers can use their spare upload bandwidth to forward new blocks to other peers. Each individual block the seeders feed into the swarm will be shared among many peers, highly leveraging the bandwidth commit-

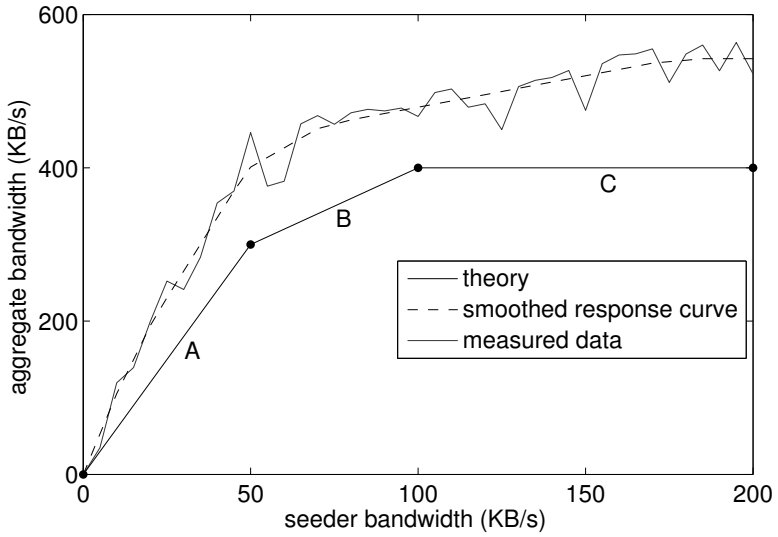

Figure 2.4: **Response curves of a theoretical homogeneous swarm and a measured heterogeneous swarm on PlanetLab.** Aggregate bandwidth increases rapidly as seeder bandwidth increases (A) until peer uplink capacity is exhausted (B) and reaches its maximum when downlinks are saturated (C).

ted by the seeder. Once the peers in a swarm have saturated their uplinks, the marginal benefit from additional seeder bandwidth drops significantly. In this regime (region B), any additional bandwidth that a peer receives only benefits that peer, since saturated upload links render it unable to forward the data to other peers. Finally, once downlinks of swarm participants are saturated (region C), the swarm has reached its maximum aggregate bandwidth. Further bandwidth provided by the seeders will not impact download latency. If download capacities are lower than upload capacities, region B will simply not exist, yielding a response curve with only two regions.

Response curves have two key properties that make them useful for allocating bandwidth. First, response curves are monotonic: a swarm's aggregate bandwidth will never decrease as a result of increasing the seeder bandwidth to the swarm. Monotonicity prevents optimization algorithms from terminating

at suboptimal local maxima. Second, response curves are concave; that is, their derivatives monotonically decrease over possible seeder bandwidths. Concavity implies that a swarm's aggregate bandwidth exhibits diminishing returns as the seeders increase their bandwidth to the swarm. When the seeders increase their bandwidth beyond a swarm-specific threshold, the peers' uplinks and downlinks saturate, decreasing their ability to receive and forward data from the seeders and other peers.

Real-life swarms are more complex than the idealized swarms discussed above in that they may comprise heterogeneous hosts and exhibit peer churn. They nevertheless exhibit several critical properties. In heterogeneous swarms, where peer uplinks and downlinks are non-uniform, the transitions between the disparate regions of the response curves are smoother. This is because different peers' upload and download capacities saturate at different points, smoothing the discontinuous transition seen in a homogeneous swarm. In addition, real swarms exhibit peer churn, where peers can join at any time and leave due to failure, cancellation, or completion. Such membership changes shift the response curve because their influence affects the swarm's dynamics, but do not violate the monotonicity and concavity properties outlined above.

Response curves form the basis for allocating bandwidth among swarms that compete for limited bandwidth. For simple content distribution networks that have a logically centralized content library, response curves alone provide sufficient information for making allocation decisions. In more complex deployments that have distributed content libraries, the response curve model of swarm behavior motivates an alternative algorithm for allocating bandwidth from multiple hosts.

2.4 Optimal Distribution of a Logically Centralized Library

Managed swarming deployments can use response curves to derive an optimal allocation of bandwidth for content distributors with relatively small content libraries. In particular, response curves contain sufficient information to allocate bandwidth from a single logically centralized seeder to a set of disjoint swarms, and provide an efficient solution to the single-seeder multi-swarm content distribution problem.

The monotonicity and concavity of swarms' response curves form the basis of their value for computing bandwidth allocations from a seeder. These properties prevent search algorithms from terminating at suboptimal local maxima, and enable the coordinator to use response curves as input to convex optimization algorithms. Given a response curve for each swarm in a managed swarming deployment, "climbing" each of the curves, always preferring the steepest curve, until all seeder bandwidth has been allocated maximizes the global benefit of the seeder's upload bandwidth. The resulting *point of operation* on each curve represents the amount of bandwidth the seeder plans to feed to each swarm and the expected aggregate bandwidth within each swarm based on the seeder bandwidth. Given each swarm's measured response curve, this allocation of seeder bandwidth is optimal [92]: decreasing the seeder bandwidth to one swarm in favor of another will not improve the overall performance of the system. The allocation of seeder bandwidth ensures that the content distributor achieves the highest performance possible from its servers' bandwidth.

The optimization process described above may reach a point at which the seeder has excess bandwidth to award, yet the derivatives of multiple response

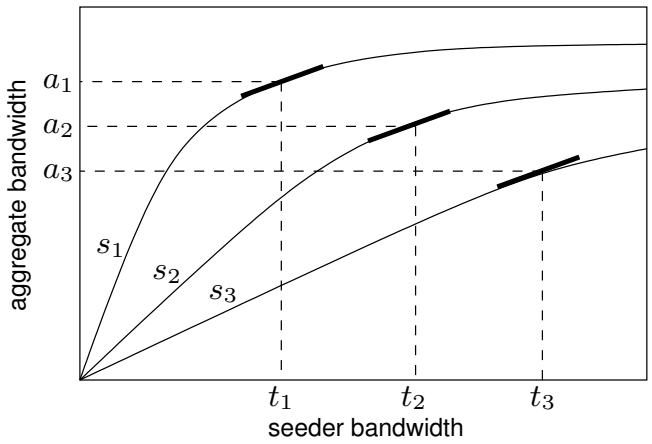

Figure 2.5: **Optimal bandwidth allocation for three concurrent swarms.** The seeder awards bandwidth to swarms by hill-climbing the steepest response curves first until all its available bandwidth has been allocated.

curves are identical, indicating that multiple swarms offer the same global benefit (Figure 2.5). In such cases of equivalent global benefit, a tie-breaker algorithm maximizes the perceived improvement by peers. Suppose that two swarms s_1 and s_2 have response curves with equivalent slopes at seeder bandwidths t_1 and t_2, corresponding to swarm aggregate bandwidths of a_1 and a_2, with $a_1 > a_2$. While this indicates that awarding a block to either swarm would improve average download times across the entire network by an equal amount, the incremental benefit to members of s_1, which already enjoy a higher aggregate throughput, is small compared to the relative improvement that members of s_2 would perceive. Consequently, seeders break ties by awarding their bandwidth to swarms with lower bandwidth when multiple response curves have the same slope. This mechanism ensures that the system maintains its primary goal of maximizing aggregate bandwidth, while the participants receive maximal marginal benefit whenever there is freedom in making a bandwidth allocation that is in line with the primary goal.

This optimization algorithm does not explicitly allocate bandwidth from non-seeder hosts that belong to multiple swarms. In our own implementation of the algorithm, we relax this constraint: peers may belong to more than one swarm, but they will not be designated seeders. As a result, such peers' bandwidth is not guaranteed to be allocated optimally among competing swarms. This relaxation allows for more realistic deployments where several peers may download multiple pieces of content simultaneously, but it can lead to reduced system-wide performance in deployments with a high amount of swarm overlap.

Overall, response curves for each swarm are sufficient for computing the optimal bandwidth allocation from the logically centralized seeder. As a result, small content distributors with limited bandwidth can be assured that their resources are used efficiently.

2.5 Efficient Distribution of a Distributed Library

Content distributors with large libraries generally split their content among many physical hosts to improve reliability by eliminating single points of failure, augment storage capacity by distributing content libraries across disks, and increase performance by positioning hosts near clients that are likely to download content. The general multi-swarm content distribution problem addresses the goals of content owners that use a distributed array of hosts to share a large, fragmented content library.

By relaxing the constraints in swarm memberships to allow for overlapping swarms and cache servers that possess partial subsets of the content, the band-

width optimization problem becomes more complex and difficult to solve. Consequently, a straightforward, centralized computation on response curves becomes impractical for large deployments. Instead, we use observations about swarm behavior based on response curves to motivate and develop an alternative method of allocating bandwidth that results in a globally efficient use of distributed resources.

Our approach is to consider the impact that each individual peer has on the performance of the swarms to which it belongs. The performance of any content distribution network is entirely dependent on the aggregate of the decisions of each individual host. However, predicting how an individual host affects a single swarm, let alone the system as a whole, is a complex problem with many variables and interdependencies. Overlapping swarms and cache servers with partial subsets of the content library only complicate matters because the bandwidth allocation decisions that one host makes can vastly affect the way that swarms behave when they receive bandwidth from other hosts.

To determine the actual benefit that a single peer has on a swarm based on its block uploads, we introduce a new metric called the *Content Propagation Metric (CPM)*. The CPM enables hosts to continuously approximate target points of swarms' response curves and to use the computed points to converge on an efficient allocation of bandwidth from origin servers, in-network cache servers, and end users.

The key insight behind the CPM is to capture how quickly a host's uploaded content propagates transitively throughout 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 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.

This section describes the CPM in detail. It first discusses the information that response curves contain for solving the general multi-swarm content distribution problem. Then it defines the CPM and 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 Chapter 3.

2.5.1 Extracting Information from Response Curves

A response curve provides an accurate model of a swarm's behavior with respect to bandwidth provided by a particular host, which we designate the seeder. In realistic deployments, however, peers belong to multiple swarms, which significantly complicates the bandwidth allocation problem. 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. Every such peer is faced with choosing which swarms should receive their bandwidth, and their decisions can have dramatic effects on the performance of the system.

Allowing swarms to overlap introduces new challenges for measuring response curves because response curves assume independence among peers' decisions. This assumption holds in the single-seeder content distribution prob-

lem, but the general variant of the problem violates this assumption: when a peer shifts its upload bandwidth from one swarm to another, it invalidates the response curves for both swarms. This dependence makes it difficult to obtain accurate measurements. Measuring a single data point in a response curve requires a peer to operate at a particular bandwidth for sufficiently long that the swarm's aggregate bandwidth stabilizes. During that time, another peer's shift in point of operation during the time interval will perturb the measured value.

To adapt the response curve model for deployments of the general multi-swarm content distribution problem, we recognize that response curves capture far more information than is necessary to compute an efficient allocation of bandwidth. We take advantage of the insight that the coordinator, when computing allocations, only uses measurements from each response curve's current point of operation.

The critical piece of information of a response curve is its slope at the point of operation. The slope indicates the instantaneous increase in bandwidth that a particular swarm will generate from a small amount of additional bandwidth. Further, the monotonicity and submodularity of response curves imply that a response curve's shape is predictable near the point of operation. Specifically, the slope can only decrease as a swarm is given more bandwidth, which caps the aggregate bandwidth that the swarm can generate from future increases in seeder bandwidth.

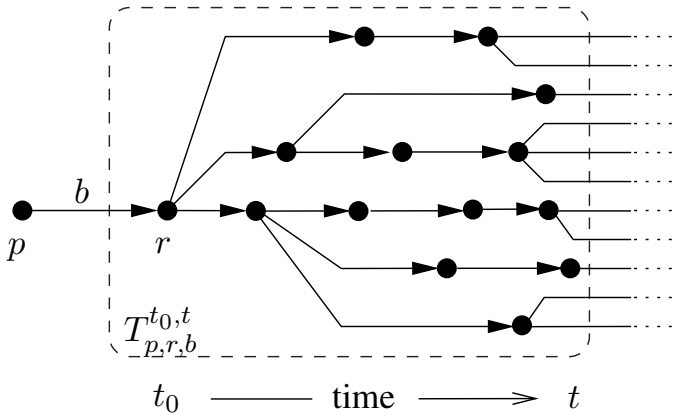

Figure 2.6: **Propagation of a block.** The dashed box indicates the propagation tree that results from peer p's tracked transfer of block b to peer r. The block propagates exponentially during the measurement time interval $\tau = t - t_0$, resulting in propagation bandwidth $v^{t_0,t}_{p,r,b} = 14 \cdot 256$ KBytes/30 s $\approx$ 120 KBytes/s, assuming 256-KByte blocks and $\tau = 30$ seconds.

2.5.2 Block Propagation Bandwidth

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. The CPM approximates the instantaneous slope of a swarm's response curve at its point of operation. Whereas a response curve represents a swarm's response to bandwidth over a range of bandwidths from a single seeder, the CPM provides a means to measure the slope of a response curve without the need to explicitly generate the curve. We define the CPM in terms of an intermediate metric called the *block propagation bandwidth*.

Block propagation bandwidth is a metric that captures 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 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

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

the download bandwidth enabled by p's tracked transfer to r over the time interval τ. Figure 2.6 shows an example propagation of a block and the resulting propagation tree.

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 the value of a block upload, consider the block propagations shown in Figure 2.7. 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.

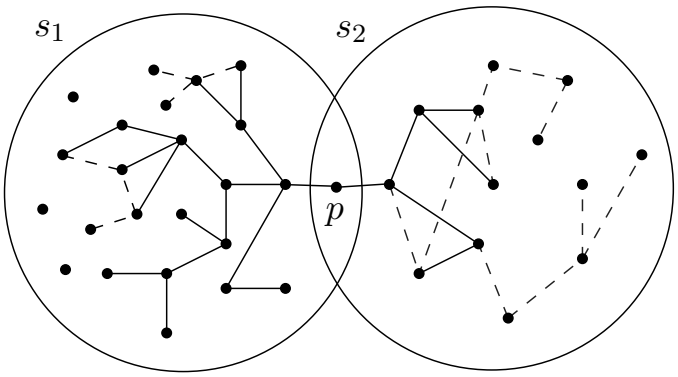

Figure 2.7: **Block propagations in two competing swarms.** Solid edges indicate the propagation of a particular block uploaded by peer p. Dashed edges indicate transfers of other blocks that compete with p's block for peers' upload bandwidth. p's block propagates more widely in swarm s_1 than in s_2.

2.5.3 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,

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

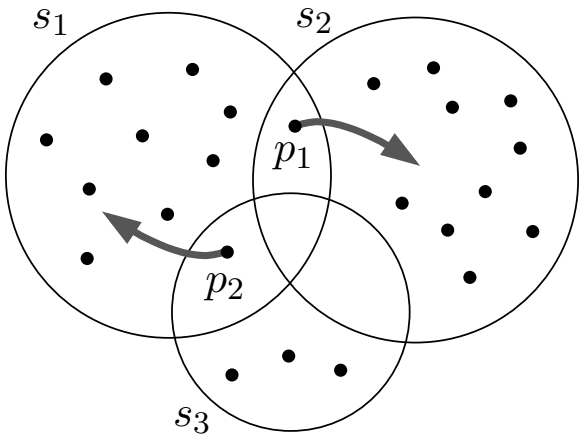

Figure 2.8: **Three competing swarms.** Peer p_1 in swarms s_1 and s_2 and peer p_2 in swarms s_1 and s_3 converge on the allocation of bandwidth indicated by the gray arrows. p_2's CPM value for s_3 is smaller than for s_1 due to the swarms' sizes; p_1 allocates its bandwidth to s_2, where its blocks do not compete with p_2's uploads.

the 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.

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 2.8. 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 will identify s_1 as the swarm that benefits more from its bandwidth due to, in this example, its larger size. p_1 likewise will measure a high CPM value for s_1, but blocks uploaded by p_2 will interfere with p_1's uploads, causing both peers'

CPM values for s_1 to diminish. Consequently, p_1 will allocate 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 will 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 use, instead, heuristics such as 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.

2.5.4 Robustness of the CPM

A metric for allocating bandwidth needs to handle changes in swarm membership and highly dynamic swarms, and achieve high performance in deploy-

ments with swarms of vastly different sizes. It also needs to dampen oscillations to converge on a stable allocation of bandwidth. We discuss characteristics of the CPM and how it handles these issues in turn.

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 of 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 of 128–256 KBytes. We have found two block uploads to be sufficient for computing provisional CPM values to adapt to highly dynamic swarms.

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,

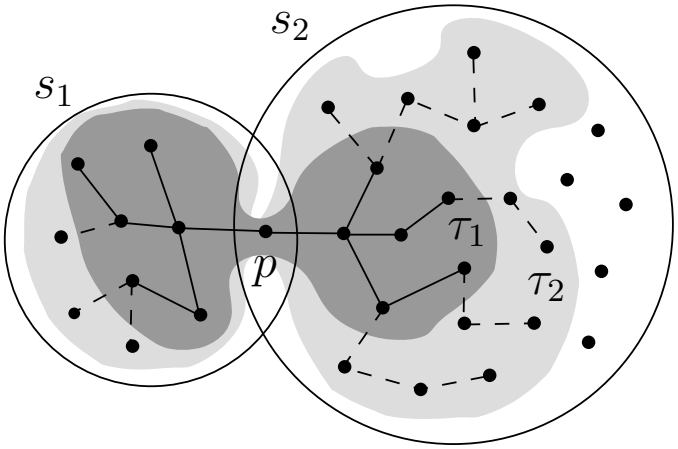

Figure 2.9: **Measurement time interval.** Edges indicate the propagation of blocks originating at peer p in swarms s_1 and s_2. Solid edges and dark shading show the propagations within a time interval τ_1 that is too small for p to differentiate its benefit to the competing swarms. Using a larger time interval τ_2, indicated by dashed edges in the lightly shaded region, makes it clear that s_2 receives more benefit from p's blocks.

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 2.9 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 time interval τ_1 prevents p from recognizing that s_2 receives more benefit from each block. The propagation trees for τ_1 are nearly identical in the two swarms, causing p to allocate its bandwidth equally between them. Increasing the time interval to τ_2 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.

To address the CPM's sensitivity to the measurement time interval, we choose an interval for each peer that teases apart the peer's highest-valued swarms. The coordinator in a deployment that uses the CPM 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.

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 Chapter 4, we show that aggregate bandwidth converges stably when there are multiple swarms vying for bandwidth from cache servers with limited upload capacity.

2.6 Summary

Content distribution systems based on managed swarms enable a logically centralized coordinator to optimize resources in order to achieve a global goal. One such goal, and the focus of this thesis, is to maximize the average download bandwidth across peers. The general multi-swarm content distribution problem formalizes this goal. A new model of swarm behavior based on response curves, which combine mathematical properties with empirical measurements, provides a foundation for addressing the general multi-swarm content distribution problem.

Small content distributors can use response curves directly to optimize the bandwidth usage in a specialized set of deployment scenarios where a content library resides at a logically centralized seeder. For large content distributors, which often deploy cache servers to increase performance and reliability, we introduce the CPM, a new metric that enables each host to measure its benefit to the system and adjust its allocations accordingly. With the CPM, a content dis-

tributor can achieve efficient use of resources in deployments with cache servers that possess only subsets of the content, as well as peers that belong to multiple, potentially overlapping swarms.

CHAPTER 3

IMPLEMENTING MANAGED SWARMS

This chapter describes our implementation of a managed swarming system. We discuss two algorithms, Antfarm and V-Formation, that address the single-seeder and general multi-swarm content distribution problems, respectively. We also describe the design of the logically centralized coordinator and cache servers, as well as two implementations of the underlying wire protocol for handling tokens exchanges and block transfers: one designed from the ground up to support Antfarm, and one based on and backward compatible with BitTorrent for standardization and interoperability. This chapter then discusses the security and scalability of managed swarming deployments. Finally, we discuss engineering challenges that we faced while building our live deployment of a managed swarming system, called FlixQ [3], and how we addressed them.

3.1 Wire Protocol

Hosts in a deployment of managed swarms communicate with each other using a wire protocol. The protocol specifies the format and semantics of the messages that peers send to other peers, such as block requests and responses and token exchanges, as well as messages sent between peers and the logically centralized coordinator, including as tokens, bandwidth allocations, and lists of swarm members.

The wire protocol's primary goal is to shuttle blocks, tokens, and allocations among peers and the coordinator, but several secondary concerns drive its design as well. First, the wire protocol should incur minimal overhead as every

block transfer is accompanied by several token exchanges. To this end, we have designed tokens to be small random strings that do not rely on cryptographic operations for signing or verification. Second, the protocol should enable the coordinator to use its global view of the system to monitor peers, detect peers that do not adhere to the protocol, and ban such peers from the system. Finally, the protocol should enable peers to employ standard swarm optimizations for uploading blocks within a swarm for block transfers that the coordinator does not explicitly dictate. Such optimizations, which are common among BitTorrent clients, include optimistic unchoking, block auctions, and rarest-first block selection.

For historical reasons, we have implemented two wire protocols for managed swarms that satisfy these goals: a prototype designed from the ground up that specifies messages for an Antfarm deployment and a production version running on our FlixQ deployment that is based on and backward compatible with the BitTorrent protocol. Both protocols enable peers to request blocks from each other in exchange for tokens, request fresh tokens from and return spent tokens to the coordinator, and receive bandwidth allocations and peers lists from the coordinator. The protocols differ predominantly in peers' communication with the coordinator, which the BitTorrent-compatible protocol addresses with HTTP requests rather than with streams of messages sent over persistent TCP connections. The second wire protocol augments the BitTorrent wire protocol, which simplified the implementation of our managed swarming client software because we were able to build it on top of an existing open-source BitTorrent client.

This section describes the messages supported by each wire protocol. We leave discussion of how peers and the coordinator behave with respect to sending and receiving messages to Sections 3.2 and 3.4, respectively.

3.1.1 Prototype Wire Protocol

We built and included a custom prototype wire protocol with our deployment of Antfarm. The protocol establishes persistent TCP connections between each peer and all its neighbors, as well as from each peer to a physical server in the logically centralized coordinator. This allows fast transmission of small messages for shuttling tokens and bandwidth allocations.

The basic data transmission protocol has three phases consisting of peer and block selection, data-for-token exchange, and bandwidth allocation. Table 3.1 lists the complete wire protocol.

Peers and the coordinator create persistent TCP connections by sending handshake messages. After establishing a connection to the coordinator, peers manage their swarm memberships by sending the coordinator join_swarm and leave_swarm messages.

Between peers, the protocol facilitates block selection and transmission using messages that are analogous to, but distinct from, BitTorrent protocol messages. In particular, a peer in a managed swarming system advertises the blocks that it possesses to its neighbors using a bitfield message when the peer first connects to a neighbor, and with incremental updates via have_block messages as the peer acquires additional blocks. Based on neighbors' block updates, a peer then

Connections
handshake Sent by peers to establish connections; includes the identifier of a file the sender wants to download and the public port of the sender.
handshake_response Sent in response to a handshake.
join_swarm Sent to the coordinator to become a swarm member.
leave_swarm Sent to the coordinator to be removed from a swarm.
time_request Sent by a peer to the coordinator to get the system time.
time_response Sent in response to a time_request; contains the time according to the coordinator.
Node state
choke Informs the recipient that the sender is not accepting block requests from the recipient.
unchoke Informs the recipient that the sender is now accepting block requests from the recipient.
interested Informs the recipient that it has at least one block that the sender needs.
not_interested Informs the recipient that the recipient does not have any blocks that the sender needs.
have_block A notification sent to directly-connected peers when a peer receives a new block.
bitfield Contains a bitfield of all the blocks the sender possesses. Normally sent after establishing a new connection.
Block transfers
request A request for a specific block.
block A block of file data, sent in response to a request.
Swarm info
peer_request Sent by a peer to the coordinator to request a set of peers in the swarm.
peer_response A set of peers' addresses and ports.
good_peers Sent periodically by the coordinator to each peer to notify them of peers to unchoke.
bad_peers A notification containing a set of peers the coordinator has identified as malicious.
allocation Sent by the coordinator to inform peers of the desired allocation of their upload bandwidth.
Token management
new_tokens Sent by the coordinator to deliver a set of fresh tokens to a peer.
token_receipt Receipt for a block transfer; sent from one peer to another in response to a block message.
token_ledger Contains a set of spent tokens sent to the coordinator in exchange for fresh tokens.
token_replace Contains a set of fresh tokens sent to the coordinator in exchange for new tokens with later expiration times.

Table 3.1: **Prototype wire protocol.** A comprehensive list of peer-peer and coordinator-peer messages. The protocol comprises messages to establish connections, notify peers of progress and status, exchange blocks, and handle tokens.

notifies neighbors whether it is interested in downloading blocks from them. Peers respond by indicating whether they will allow the peer to request blocks with choke and unchoke messages.

Another set of messages allow the coordinator to distribute swarm metadata to peers. Peers discover other peers via the coordinator's peer_response message, which contains a subset of the swarm's peers. The good_peers and bad_peers messages enable the coordinator to send peers subsets of the swarm's peers based on their past behavior. The coordinator uses these messages to reward and punish peers; the message recipient is encouraged to exchange blocks with good peers and to avoid bad peers.

The coordinator informs peers of its bandwidth allocation decisions with an allocation message. The message contains the absolute bandwidths that the recipient peer should allocate to each swarm to which it belongs until the peer receives the next allocation message.

Finally, a set of messages facilitate token exchanges among peers and between peers and the coordinator. The coordinator sends fresh tokens to peers in new_tokens messages, peers send each other tokens in exchange for blocks in token_receipt messages, and token_ledger messages enable peers to deposit spent tokens at the coordinator.

3.1.2 BitTorrent-Compatible Wire Protocol

We designed a second wire protocol that is backward compatible with the BitTorrent protocol for our own open, large scale content distribution system. For

messages between peers, the protocol uses pure BitTorrent messages, with two modifications. First, when establishing a connection with a new peer, a managed swarming peer informs the other peer that it expects to receive tokens in exchange for uploaded blocks. Second, a new token message encapsulates tokens sent in exchange for a block. Using this augmented BitTorrent protocol among peers, managed swarming peers can optionally connect to traditional BitTorrent peers, with the knowledge that such peers do not possess tokens.

To communicate with the coordinator, peers issue HTTP requests similar to requests that BitTorrent peers send centralized BitTorrent trackers to join swarms and obtain lists of peers. Table 3.2 lists the full peer-coordinator protocol:

announce (request) An HTTP GET request modified from BitTorrent's announce request sent by a peer to the coordinator to join or leave a swarm, and to request a list of peers in the swarm. Parameters: - *content_id* Identifier for a swarm and its content - *peer_id* The announcing peer's unique identifier - *port* The peer's public port - *event* Swarm membership and content download state; one of "completed", "stopped", or "started" - *left* Number of bytes the peer has left to download - *upload_bw* The peer's upload capacity available for allocation
Table continued on next page...

Table continued from previous page...

announce (response) The coordinator's response to an announce request. Returned values:

- *interval* The number of seconds the peer should wait before issuing another announce request
- *peer_list* A list of IP addresses and public ports of other peers in the swarm
- *cpm* The peer's current CPM value for the swarm
- *probe* A Boolean value indicating whether the peer should probe the swarm so the coordinator can calculate a fresh CPM value
- *stride* Determines which blocks the coordinator tracks to calculate CPM values: a block is tracked if and only if the block identifier modulo the stride is 0; used by the peer to prioritize incoming block requests when probing a swarm

get_tokens (request) An HTTP GET request sent by a peer to request fresh tokens for a particular swarm. Parameters:

- *content_id* Identifier for a swarm and its content
- *peer_id* The requesting peer's unique identifier
- *port* The peer's public port
- *num_tokens* The number of tokens that the peer wants

get_tokens (response) The coordinator's response, containing a generator that the peer can use to create a specific number of fresh tokens for a limited time. Returned values:

- *token_generator* A secret key that the peer uses to sequentially generate valid tokens
- *epoch* The current epoch, used by the coordinator to accept or reject tokens based on the age of their generator
- *start_serial* The sequence number at which the peer should number the tokens that it generates
- *num_tokens* The number of valid tokens the token generator will yield
- *min_request_interval* The minimum amount of time before the peer can request more tokens

Table continued on next page...

Table continued from previous page. . .
deposit_tokens (request) An HTTP POST request sent by a peer to deposit spent tokens. Parameters: - *content_id* Identifier for a swarm and its content - *peer_id* The requesting peer's unique identifier - *port* The peer's public port - *tokens (POST data)* A packed concatenation of the peer's ledger (spent tokens that the peer received in exchange for blocks)
deposit_tokens (response) The coordinator's response. Returned values: - *num_tokens* The number of successfully deposited tokens - *ban_ips* A list of addresses of peers from which the requesting peer received invalid tokens; the requesting peer should no longer exchange blocks with returned peers

Table 3.2: **BitTorrent-compatible wire protocol.** A list of coordinator-peer messages.

Basing a managed swarming wire protocol on an existing, widely used protocol eases adoption, accessibility, and implementation of managed swarming systems. First, it enables BitTorrent clients to interoperate with managed swarming systems. Although BitTorrent peers cannot participate in token exchanges or allocate bandwidth according to the coordinator's calculations, existing BitTorrent deployments contain a wealth of content that could potentially draw users toward managed swarms as an alternative. Additionally, layering the protocol on top of the existing, well-established BitTorrent protocol lowers the barrier for third-party developers to adopt the managed swarming approach. Finally, basing our protocol on a widely adopted protocol enabled us to use existing, open-source implementations of BitTorrent's peer-to-peer protocol as a basis for our own implementation.

The coordinator employs peer-facing web servers that collectively function as an augmented tracker for facilitating swarms, similar to a BitTorrent tracker.

Web servers accept three types of asynchronous requests from peers. Peers issue announce requests periodically for each swarm, get_tokens requests when their fresh tokens are nearly depleted, and deposit_tokens requests when they possess spent tokens from other peers.

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 set of peers and the requesting peer's most recent CPM value for the swarm. The coordinator adjusts announce intervals dynamically to achieve a constant CPU utilization on the web servers. If the coordinator does not have sufficient data to determine a peer's CPM value, the response asks the peer to probe the swarm so that the coordinator obtains fresh block exchange information.

The get_tokens and deposit_tokens requests facilitate the exchange of fresh and spent tokens, respectively, between peers and the coordinator. The coordinator maintains a credit balance for each peer that represents the total number of tokens that the peer can obtain across all swarms. When a peer issues a get_tokens request, the coordinator updates the peer's credit balance and sends the peer an equal number of fresh tokens for a particular swarm, to be exchanged for blocks within a specific time interval. Peers with a balance of zero may continue to request blocks from peers, but its requests will be prioritized strictly below requests from peers that have tokens. This mechanism enables peers to obtain blocks, which they can use to acquire tokens in the future, but it only uses peers' spare upload bandwidth, and therefore does not interfere with com-

puted bandwidth allocations. 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 coordinator verifies the token's authenticity, increases the peer's balance accordingly, and records the block transfer for calculating future bandwidth allocations.

The following sections explore how peers, cache servers, and the coordinator operate on top of the wire protocol in managed swarming systems.

3.2 Peers

Peers behave similarly regardless of the bandwidth allocation algorithm and the underlying wire protocol. Within a swarm, peers operate identically to BitTorrent peers and adopt common mechanisms from BitTorrent clients, including optimistic unchoking for peer discovery, tit-for-tat for bandwidth allocation, and rarest first for block selection. The coordinator's bandwidth allocation decisions, then, affect peer behavior by instructing peers how to prioritize requests for blocks from peers in competing swarms.

Peers use the coordinator's bandwidth allocation decisions to choose which block requests to satisfy, and in what order. Each peer will receive a stream of incoming block requests from peers in multiple swarms. Peers participating in a deployment that uses the Antfarm algorithm allocate their bandwidth among competing swarms according to absolute bandwidth values from the coordinator. Peers in V-Formation deployments prioritize their swarms based on the CPM values contained in announce responses. Upon receiving an updated CPM value, a peer updates a local list of its swarms, strictly ordered by CPM value.

When a peer has received multiple outstanding block requests from peers in different swarms, it satisfies a request from the swarm with the largest CPM value.

Guided by the coordinator's inter-swarm bandwidth allocations, peers determine their own criteria for selecting specific peers and blocks to exchange. This enables peers to perform optimizations based on local information, reducing the burden on the centralized coordinator. The default peer behavior in managed swarming deployments for peer and block selection is identical to BitTorrent. Peers retain a prioritized list of other peers with which to exchange data blocks (to *unchoke*). The priority order is determined by maintaining a rolling average of the bandwidth achieved through that peer's history of interactions. Peers choose blocks using a rarest-first algorithm; they maintain a bitmap of blocks held by each connected peer constructed from block acquisition notifications sent by peers after each block transfer. Since swarming systems that rely solely on local information and randomized interactions may operate at reduced efficiency due to lack of information [67], the coordinator uses its global knowledge to influence peer selection. The coordinator monitors each peer's upload history and identifies underutilized peers. It sends lists of such peers as candidates for data exchange. This is an advisory notification that causes the recipient to increase the priority of the named, underutilized peers. This is a no-cost optimization; a peer is under no obligation to follow the recommendations and the protocol's correctness does not depend on the peer-selection algorithm. This process of aiding peer selection is also improved by the use of network proximity measures [94, 39, 75].

Once a peer (receiver) has chosen another peer (sender) and determined a suitable block for download, it sends a block-exchange request. If the sender has unchoked the receiver, it sends the requested block to the receiver. Upon completion of the transfer, a non-malicious receiver checks the hash of the block against the hash specified in the swarm description and sends an unexpired token to the sender of the data block. Each peer maintains a *purse* of unused tokens issued by the coordinator for use by that peer, and a *ledger* of tokens received from other peers in exchange for data blocks. Tokens flow from the purse of the receiver to the ledger of the sender.

Peers communicate periodically with the coordinator to refresh their purses and ledgers. Each unexpired token in the ledger entitles the peer to a fresh token for its purse. If a newly received token in the ledger is going to expire before the next scheduled refresh, or if the purse contains nearly expired unspent tokens, the peer can preemptively redeem selected tokens for new tokens with later expiration times.

3.3 Cache Servers

Content distribution systems may employ cache servers to improve content availability and download times. Operators of the content distribution systems can deploy such cache servers themselves, or ISPs can choose to deploy their own cache servers to provide better service for their customers or reduce inter-ISP traffic. In either case, the managed swarming coordinator measures cache servers' effects on their swarms, and uses the data to allocate their bandwidth efficiently among swarms.

Antfarm and V-Formation handle cache servers differently, as discussed for the single-seeder and general multi-swarm content distribution problems, respectively. Antfarm explicitly designates servers that possess the content library in its entirety as seeders, and includes such servers' upload bandwidth in its allocation of seeder bandwidth. Thus, additional physical machines that constitute the logically centralized seeder function to increase the seeder's total bandwidth that the algorithm optimally allocates among swarms. Antfarm treats cache servers that do not possess all the content as normal peers, and the coordinator does not manage bandwidth from such servers, even if the servers belong to multiple swarms.

In V-Formation, in contrast, the coordinator manages bandwidth from all hosts, regardless of swarm membership. As a result, content distributors and ISPs have more freedom to deploy cache servers that only contain a partial subset of the system's content library.

V-Formation treats cache servers and end users identically: both are simply peers that can belong to any number of swarms, either to download content or to provide blocks to downloading peers. V-Formation, then, determines a cache server's bandwidth allocation in the same way that it does for an end user: the coordinator computes a CPM value for each swarm to which the cache server belongs, and instructs the cache server to strictly prioritize swarms for which its bandwidth has the highest impact. Because the coordinator uses the same criteria for all hosts when making its bandwidth allocation decisions, V-Formation grants network operators and ISPs the flexibility to deploy cache servers: caches can contain all content to improve availability, or they can contain only a se-

lected subset, such as the most popular items, to improve download speeds for clients.

Cache servers in managed swarming deployments effectively support both open and closed content distribution systems. In Antfarm, cache servers are managed optimally when they contain all the system's content. Thus, they naturally support a closed content distribution system, where the content distributor manages the content available for download. Alternatively, an Antfarm deployment can function as an open content distribution system where users are required to upload content to the logically centralized seeder before it becomes available for download, similar to YouTube. V-Formation, on the other hand, does not make a distinction between cache servers and end users. As a result, V-Formation is more suitable for an open content distribution system, where users share original content directly from their computers, and cache servers aid in the distribution content by caching popular content as it propagates throughout the network.

3.4 Coordinator

The logically centralized coordinator is the heart of a managed swarming system. Its function is to track block transfers among peers, compute efficient bandwidth allocations, notify hosts how they should allocate their bandwidth, and provide accountability and management to ensure that hosts follow the prescribed allocations.

The coordinator collects statistics on peer network behavior, computes response curves and bandwidth allocations for each peer and seeder, and steers

each swarm toward an efficient point of operation. It affects each swarm's point of operation through manipulation of the token supply and direct interaction with cooperative peers. Finally, it keeps track of malicious and uncooperative participants, excising them from the network when their misbehavior affects performance.

The primary task of the coordinator is to monitor network characteristics and swarm dynamics by keeping track of tokens for each data block transaction between peers. Each token the coordinator receives informs the coordinator of the swarm in which a transaction occurred, the specific peers involved in the transaction, and a window of time in which the data block was transferred based on the token's minting and expiration times.

In large deployments, the coordinator may be distributed across physical machines, and the organization of the physical machines affects the coordinator's efficiency for its various tasks. This thesis describes two organizations for a distributed coordinator that differ in how information about block transfers are aggregated and where bandwidth allocations are computed. A coordinator with a *hierarchical organization* and *centralized computation* is ideal for Antfarm, where the coordinator periodically aggregates information and computes a bandwidth allocation for only a single logically centralized seeder. In contrast, V-Formation computes bandwidth allocations for each of potentially many hosts, each with different swarm memberships and bandwidth capacities. As a result, a coordinator with a *flat organization* that is optimized for *distributed computation* is more suitable because it provides a natural method to continuously aggregate block transfers, and it computes bandwidth allocations using several physical machines.

We describe both coordinator organizations in detail and discuss the advantages of each for coordinating peer behavior using the two bandwidth allocation algorithms. Then, we describe the design of a completely decentralized coordination scheme in which the coordinator's tasks are delegated to the peers themselves. The result is a decentralized managed swarming system for use in trusted environments when centralization is infeasible.

3.4.1 Hierarchical Organization, Centralized Computation

A distributed coordinator organized hierarchically comprises two types of servers: a single server, called the coordinator's *root server*, centrally computes all bandwidth allocations based on recent token exchanges. The remaining servers, called *token servers*, communicate directly with peers. They are tasked with issuing tokens, collecting tokens back from peers, and periodically sending each peer's upload and download rates to the root server. When a new peer enters the system, it contacts the root server. The root server redirects each peer to a token coordinator based on a hash of the peer's IP address. When a token coordinator receives a spent token from an assigned peer, it applies the same hash function to the IP address of the token's original owner, a field in the token itself, and verifies the token with the token coordinator that issued it. Thus, each token exchanged between peers involves at most two of the coordinator's token servers.

A hierarchically organized coordinator that computes bandwidth allocations at a single root server is well suited for the single-seeder multi-swarm content distribution problem. There are two key reasons for this. First, the coordinator

aggregates all block transfer information to a single host, which then possesses all data required to determine the optimal allocation of bandwidth from the logically centralized seeder. This renders the bandwidth allocation algorithm tractable without the need for a distributed optimization algorithm. Second, because there is only a single seeder, the coordinator only periodically computes new bandwidth allocations. Thus, token servers need to send summaries of token exchange data to the root server only when the root server needs to compute allocations, resulting in low network overhead within the coordinator.

The coordinator's centralized computation of bandwidth allocations makes it ideal for managed swarming systems based on Antfarm. The remainder of this section explores the coordinator's role in a deployment of Antfarm, which highlights how the coordinator collects tokens from peers using token servers, computes bandwidth allocations at the root server, and notifies peers and the seeder of the coordinator's decisions.

Based on token exchanges, each coordinator token server maintains two key parameters for each peer p assigned to it: the set of swarms S_p that p is a member of and a rolling average of its upload bandwidth u_p, which the token server updates based on tokens that it receives, and sends to the root server upon request. The root server keeps track of the set of all physical hosts T that comprise the logically centralized seeder, as specified by the network operator. The coordinator assumes that all hosts in T are members of all swarms, and that they each possess the system's content in its entirety. Finally, for each swarm s, the root server maintains a set P_s of peers in s and a *response scatterplot* for s, represented as a collection of data points that each expire 30 minutes after they are measured.

The coordinator chooses swarms to grant bandwidth based on collected swarm statistics. The response scatterplots are not immediately suitable for use in computing bandwidth allocation, as they contain artifacts due to measurement errors and changes over time, creating false local minima and maxima. The coordinator generates a response curve from a response scatterplot for swarm s, which we denote as a function ρ_s that maps the seeder bandwidth that s receives to the expected aggregate bandwidth that s will generate.

The coordinator computes ρ_s from the response scatterplot by fitting a piecewise linear function. Given a response scatterplot of (seeder bandwidth, aggregate bandwidth) pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, with $x_1 < x_2 < \cdots < x_n$, the coordinator computes a new set of points $(x_1, y'_1), (x_2, y'_2), \ldots, (x_n, y'_n)$ with the same x values. The new aggregate bandwidth values $y'_1, y'_2, \ldots, y'_n$ are chosen according to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=1}^{n} (y_i - y'_i)^2 \\ \text{such that} \quad & \text{slope}_i \geq 0, \quad 1 \leq i < n, \\ & \text{slope}_{i+1} \leq \text{slope}_i, \quad 1 \leq i < n-1, \end{aligned}$$

where $\text{slope}_i = (y'_{i+1} - y'_i)/(x_{i+1} - x_i)$. The response curve ρ_s, then, is the function created by connecting consecutive generated points with line segments. The resulting response curve minimizes least squares and respects the monotonicity and concavity constraints of theoretical response curves, making them suitable input to Antfarm's bandwidth optimization algorithm.

The coordinator sends the seeder hosts in T explicit bandwidth allocations for each swarm based on the optimal solution to the single-seeder multi-swarm content distribution problem. In addition, the coordinator calculates each peer's upload capacity based on deposited tokens and uses it to send end users per-

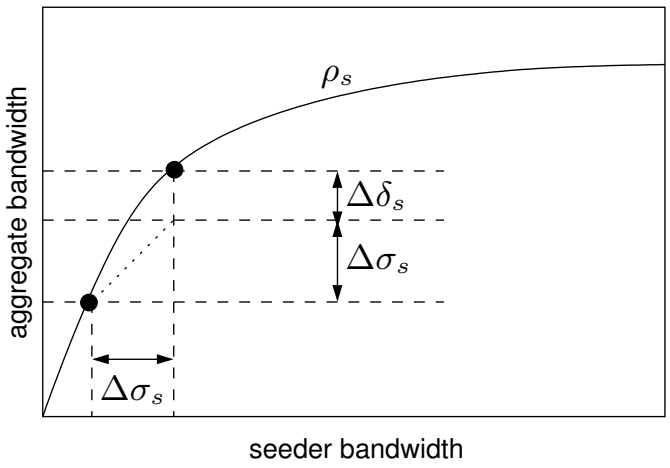

Figure 3.1: **Allocating bandwidth with response curves.** The black dots denote the allocation of bandwidth for swarm s before and after one iteration of allocation. For each $\Delta\sigma$ tasked to the seeder by the hill-climbing algorithm, a randomly selected peer with spare upload capacity is tasked with allocating a corresponding $\Delta\delta$. The dotted line has a slope of 1, accounting for the seeder's contribution to the swarm's aggregate bandwidth.

swarm allocations. These allocations are not guaranteed to be optimal; the coordinator generates them with a greedy algorithm to distribute bandwidth from peers that are members of multiple swarms. The mechanism is an optimization, but Antfarm provides no guarantees of the system's efficiency when there is high overlap among swarms.

Based on the computed response curves, the coordinator computes the amount of bandwidth that each seeder host and peer should dedicate to each swarm, represented as two matrices σ and δ. For each swarm s, $\sigma_{t,s}$ captures the amount of bandwidth seeder host t will dedicate to s, and $\delta_{p,s}$ captures the amount of bandwidth peer p is expected to dedicate to s. This determines the critical allocation of seeder upload bandwidth $\sigma_s = \sum_{t \in T} \sigma_{t,s}$ to swarm s in order to achieve a swarm aggregate bandwidth $\rho_s(\sigma_s)$, based on s's response curve. The component of swarm bandwidth coming from peer-to-peer, non-seeder block exchanges, then, is $\delta_s = \rho_s(\sigma_s) - \sigma_s = \sum_{p \in P_s} \delta_{p,s}$. The coordi-

nator computes this allocation periodically (every five minutes in our current implementation). In computing σ and δ, the coordinator operates under the constraint that $\sum_{s \in S_p} \delta_{p,s}$ cannot exceed p's upload capacity u_p. The coordinator determines σ and δ iteratively. Initially, $\sigma_{t,s} = \delta_{p,s} = 0$ for all peers p, seeder hosts t, and swarms s. The coordinator determines the allocation of bandwidth through a greedy hill-climbing algorithm using the computed response curves and its knowledge of the seeder hosts' upload capacities. This process is illustrated in Figure 3.1. The coordinator allocates bandwidth in discrete units to the swarms whose response curves have the highest gradient, breaking ties in favor of the swarm s with the lowest value of $\rho_s(\sigma_s)$, as described in Chapter 2.4. For each increase in seeder bandwidth σ_s to swarm s by an amount $\Delta\sigma_s$, the algorithm computes the anticipated increase in swarm aggregate bandwidth $\Delta\rho_s = \rho_s(\sigma_s) - \rho_s(\sigma_s - \Delta\sigma_s)$. It then chooses a peer at random from P_s with spare upload bandwidth and tasks it with uploading to s an additional $\Delta\delta_s = \Delta\rho_s - \Delta\sigma_s$. The coordinator continues the process until all seeder bandwidth has been allocated. The final peer allocation δ ensures that peer transfers within each swarm achieve the previously measured aggregate bandwidth based on the seeder's allocation σ.

Computation of bandwidth allocation is not a highly time-critical task. Delays in network measurements and peer interactions imply inherent delays between computing an allocation and seeing a change in the network. Since the latency of computing the bandwidth allocation is dwarfed by the latency of data exchange, the computation can be performed in the background. The optimization algorithm is linear in the number of peers and grows according to $O(n \lg n)$ with the number of swarms, enabling the system to scale. The primary metric

that determines the quality of the solution is the freshness of data on swarm dynamics.

To keep the coordinator's view of the network accurate, Antfarm's token protocol incentivizes peers to report statistics to the coordinator in a timely manner. A token's expiration time (five minutes in the current implementation) and spender-specificity force peers to return tokens to the coordinator in order to receive bandwidth in the future. The circulation of tokens reveals sufficient information to the coordinator to perform the allocation described above.

Token-based economies can suffer from inflation, deflation, and bankruptcy if left unmonitored. Based on analyses of scrip systems [69], the Antfarm coordinator maintains a constant number of tokens per swarm per peer (30 in the current implementation). New peers receive an initial allowance of 30 tokens. As unspent tokens expire, the coordinator redistributes an equal number of new tokens to random peers to prevent a token deficit when peers depart with positive token balances. Token unforgeability prohibits deflation, and token redistribution enables bankrupt peers to acquire new blocks and reintegrate themselves into the swarm.

The coordinator rewards peers that contribute to the system both directly, by offering seeder bandwidth to peers that have donated bandwidth to peers, and indirectly, by encouraging peers to request blocks from underutilized peers. For the latter, the coordinator uses data from tokens to identify top uploaders with spare bandwidth. Then, to each peer, the coordinator periodically sends a list containing a subset of the identified peers, where each peer is included with a probability proportional to its upload bandwidth. When peers receive these

lists, they can choose to preferentially unchoke those peers and request blocks from them.

Peer churn and changes in network conditions cause response curves to become stale over time. In addition, transient measurement errors can skew response curves, causing the system to operate suboptimally. Antfarm maintains response curves by actively exploring the swarm's response at different seeder bandwidths. In each epoch, the coordinator randomly perturbs the current bandwidth allocation by a small amount for each swarm, on the order of 5 KBytes/s. Such variances provide additional data points on the response scatterplot, enabling the system to overcome false local minima due to transient effects.

In our current Antfarm implementation, the coordinator does not enforce peers' compliance with the coordinator's directives in allocating their upload bandwidth. A peer can shift bandwidth away from one swarm in favor of another swarm at its discretion. When a peer deviates from its prescribed bandwidth allocation, it results in a small shift in the affected swarms' response to bandwidth. The coordinator updates its response curves as it takes new measurements, and the peer's decisions are reflected in future bandwidth allocations that the coordinator computes from the updated response curves.

Overall, a hierarchical coordinator computes bandwidth allocations at a single host, and the coordinator periodically updates its bandwidth allocations for a set of seeder hosts specified by the network operator.

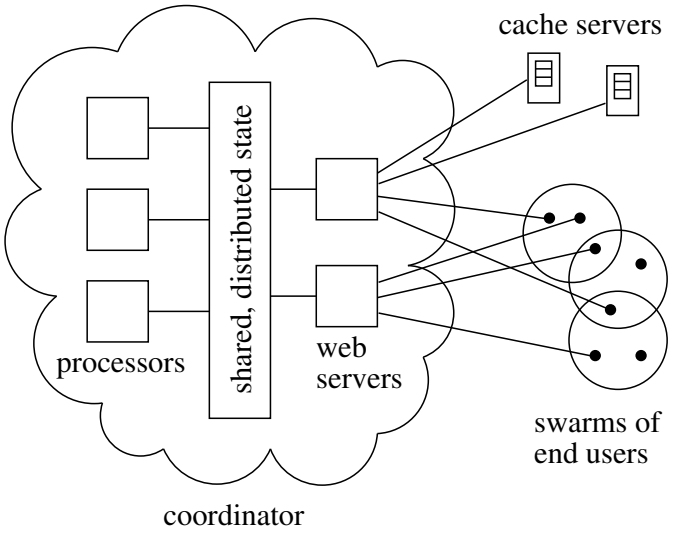

Figure 3.2: **Distributed computation in a flat coordinator.** Web servers communicate with peers (including cache servers) to gather information on swarm dynamics. Processors use swarm dynamics to compute peer bandwidth allocations. Web servers and processors communicate via a distributed state layer to maintain consistency.

3.4.2 Flat Organization, Distributed Computation

The general multi-swarm content distribution problem is larger scale than the single-seeder variant because it requires the coordinator to compute allocations from multiple contributors rather than from only a single seeder. To address the larger scale, we have designed a sharded solution that calculates bandwidth allocations in a distributed manner. Instead of periodically aggregating coarse-grained summaries to a single host that performs all computations, we introduce a flat coordinator, where each of a potentially large set of hosts continuously performs a share of the bandwidth allocation computation as new block exchange information becomes available. The coordinator's design is well suited for managed swarming deployments that use V-Formation, and the remainder of this section uses V-Formation to describe and motivate a flat coordinator.

The logically centralized coordinator consists of three components: web servers, processors, and a shared state layer (Figure 3.2). First, web servers process requests from peers to dispense 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, implemented as a key-value store, 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 hosts, 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 the information stored in the distributed state layer.

Web Servers

The coordinator's peer-facing web servers function as an augmented tracker for facilitating swarms, similar to a BitTorrent tracker. Web servers handle three types of asynchronous requests from peers: announce, get_tokens, and deposit_tokens. The format and semantics of these messages is described in Chapter 3.1.2. To summarize, web servers update swarm and peer metadata, distribute lists of peers, issue tokens, adjust peers' credit balances, and accept deposits of spent tokens. When a web server receives a deposited token from a peer, it 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, annotated with the block's identifier and timestamped using the coordinator's clock upon receiving the token.

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, counting 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 lists of the uploading peers' block propagation bandwidths, maintained in the state layer. Each new propagation bandwidth is timestamped with the time on the forest's tracked transfer edge, equal to the time at which the tracked transfer was reported to the coordinator.

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 hosts linearly increases the supported processing workload of the coordinator.

If high load renders the coordinator unable to process all block propagation forests using available processor hosts, 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.

Distributed State Layer

Our flat coordinator uses memcached to implement a distributed, shared state layer for web servers and processors, which it uses to maintain 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, the state layer records, for each peer, 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, whose sizes are linear in 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 on each read.

3.4.3 Decentralized Coordination

In deployments of managed swarms in trusted environments, peers can collectively replace the centralized coordinator and operate in a completely decentralized manner. The resulting system can achieve the same aggregate bandwidth as V-Formation. In the decentralized protocol, peers compute their own CPM values based on messages from downstream peers instead of obtaining them from a coordinator.

Decentralized coordination requires that hosts are honest without a centralized entity that provides accountability and detects malicious behavior. The decentralized protocol sacrifices the use of tokens, as well as the accompanying global view of the network and mechanisms for penalizing misbehaving peers that a centralized coordinator provides. Thus, decentralized coordination is appropriate for deployments within a trusted environment where peers are not incentivized to game the system, and where introducing a logically centralized component is infeasible.

The decentralized protocol translates the centralized coordinator's remaining tasks of tracking block transfers, computing block propagation forests, and extracting CPM values, into distributed operations that the peers perform. In the protocol, peers prepend each transferred content block with a header con-

taining an *ancestor list*. An ancestor list specifies the block transfer's ancestors in the block's propagation trees. More precisely, an ancestor list contains the addresses of peers tracking the block's propagation, each paired with an expiration time that indicates when the tracking period ends.

Upon receiving a block, a peer first removes expired entries from the beginning of the block's ancestor list, as determined by the expiration timestamps paired with each listed peer. The peer then sends a block_propagation message containing the block number to each peer in the ancestor list to notify each ancestor of the recent block transfer. Finally, before forwarding the block to satisfy downstream block requests, the peer optionally appends itself to the block's ancestor list if it wishes to track the block's propagation for its own measurements.

When a peer tracks the propagation of a block by adding an entry to the block's ancestor list, the peer will receive one block_propagation message for each downstream transfer of the block within the block tracking time specified by the entry's expiration time. A peer simply counts the number of block_propagation messages that it receives to calculate its block propagation bandwidth, and it maintains a rolling average of block propagation bandwidths for each swarm in order to compute its CPM value for each swarm. Thus, each peer performs a similar computation as a centralized coordinator using V-Formation to determine its own bandwidth allocation among competing swarms. Like the centralized coordinator's approach, a peer can randomly track a subset of the blocks that it uploads, and it can adjust the expiration times of each tracked block to control the measurement time interval τ.

Overall, decentralized coordination of managed swarms offers the advantages of a managed swarming system without the need for a logically central-

ized coordinator. Without the coordinator, however, the system requires that hosts follow the protocol without an omniscient observer to provide incentives for good behavior.

3.5 Security

A formal treatment of the security properties of the wire protocols is beyond the scope of this thesis. Past work on similar, though heavier-weight, protocols [115] has established the feasibility of a secure wire protocol. Consequently, the focus of this section is to enunciate our assumptions, describe the overall goals of the protocol, provide design alternatives, and outline how to mitigate attacks targeting managed swarming algorithms.

Our wire protocols make standard cryptographic assumptions on the difficulty of reversing one-way hashes and assumes that peers cannot snoop on or impersonate other peers at the IP level. Violation of the first assumption would render our wire protocols, as well as most cryptographic algorithms, insecure; consequently, much effort has gone into the design of secure hash functions. Violation of the second assumption is unlikely without ISP collusion, and damage is limited to IP addresses that an attacker can successfully snoop and masquerade.

Managed swarms require peers to contribute bandwidth to their swarms, engage in legitimate transactions with other peers, and report accurate statistics to the coordinator. The token protocol, coupled with verification at the coordinator, ensures detection of dishonest peers with relatively low overhead.

In order to measure accurate response curves, the coordinator verifies that all token transactions occur within the intended swarm, by the intended peer, and within the intended period of time. The coordinator detects token forgery upon receiving an invalid token from a peer by verifying deposited tokens to ensure that the coordinator minted them; Chapter 3.7.2 describes the implementation details of token management. Similarly, the coordinator compares its own record of the intended sender with the spender as reported by the peer returning the token to prevent peers from spending maliciously obtained tokens. Peers are required to report the actual spender in order to receive credit for the deposited token. The coordinator detects all counterfeit tokens, but when it detects an invalid token, it is unable to determine whether the culprit is the peer that deposited the token or the peer that spent the token. Therefore, it notifies both peers of the forgery so the honest peer can blacklist the culprit.

To hold peers accountable for their actions when the coordinator is unable to precisely identify malicious peers, we employ a *strikes* system to record and act on undesirable behavior. Peers maintain a tally of strikes against other peers and disconnect from peers that have exceeded a threshold. By default, misbehaviors that can stem from network congestion, such as a late response to a block request or payment with a recently expired token, result in one strike against the offending peer. Circulating a counterfeit token results in automatic termination of the connection. In general, when the coordinator is unable to determine the identity of a malicious peer, it appeals to the strikes system rather than erroneously penalizing an honest peer. While it is possible to build a centralized reputation system for peers, our current implementation avoids this to reduce burden on the coordinator.

Using cryptographically signed tokens can provide stronger guarantees at the cost of additional overhead and complexity. In such a scheme, the coordinator can sign all minted tokens before issuing them to peers, enabling peers to verify that they are exchanging legitimate tokens with each other during each transaction. In addition, if the spender of a token were required to sign the token before sending it, peers could prove the identities of token double-spenders. Token signatures would prevent malicious peers from snooping packets and tampering with tokens without the recipient's knowledge. We do not implement a cryptographic scheme, and instead opt to use a lightweight token scheme for scalability.

It is possible for peers to collude in order to coerce the coordinator into providing their swarm with more bandwidth. In particular, peers could band together and send each other large numbers of tokens without sending each other blocks in exchange. The resulting inflated estimate of that swarm's aggregate bandwidth can lead the system to deviate from a desired allocation. Several techniques mitigate such attacks. First, the coordinator never issues more tokens than strictly necessary to download the file, thereby bounding the impact of fake transactions by the number of Sybils. Second, forcing participants to register with a form of hard identity, such as credit card numbers, can mitigate Sybils [23]. Finally, the coordinator can mandate that peers trade with a diverse set of peers, reducing the effect of collusion among a small fraction of the swarm. Although the token protocol does not eliminate the possibility of malicious behavior, its simplicity and ability to detect malicious activity limit the harm peers can inflict.

3.6 Scalability

Coordinators of managed swarming systems are optimized to ensure that the logical centralization does not pose a CPU or bandwidth scalability bottleneck. In this section, we first discuss the scalability of token management. The section then explores the differing scalability properties of coordinators with hierarchical and flat organizations, respectively.

Shuttling tokens to and from the coordinator for each data block transfer is the primary consumer of the coordinator's bandwidth. To reduce the burden, managed swarming systems do not rely on public-key cryptography to issue or exchange tokens. The protocols minimize the size of tokens on the wire, transmitting only relevant fields when tokens change hands. Only a token's identifier, file reference, and expiration time are sent on the wire when the coordinator sends fresh tokens, and only the identifier and expiration time are sent on the wire when a peer sends another peer a token. Spent tokens sent back to the coordinator are represented with only the token's identifier and the identifier of the peer that spent the token. Using 4-byte token identifiers, each token exchange requires less than 24 bytes of total bandwidth and less than 16 bytes of bandwidth, plus HTTP overhead, at the coordinator for each data block of around 32–256 KBytes. The HTTP overhead is negligible because peers deposit tokens in bulk. The BitTorrent-compatible wire protocol further reduces the coordinator's bandwidth requirements by sending peers secret keys that allow them to generate their own tokens instead of sending fresh tokens on the wire. Chapter 3.7 describes this optimization in detail.

Managed swarms use highly compact token identifiers to reduce bandwidth. A 4-byte identifier is sufficient to disincentivize forgery because the coordinator will detect a malicious peer's attempt to forge a token upon its first failure to produce a legitimate token. In the event that a peer correctly guesses an active token's identifier, it is unlikely to correctly identify the token's intended spender. In the worst case, should a peer successfully forge a token, it will only gain one data block for its efforts, whereas failures will lead to remedial action against the peer, described in Chapter 3.5. Thus, with 4-byte token identifiers, several million peers, and several hundred million tokens, the likelihood of a successful, undetected token forgery is around 10^{-8} when tokens are uniformly distributed. With a skewed token distribution where some peers have 100 times more tokens than the average peer, the likelihood might rise as high as 10^{-6}. Downloading ten blocks with forged tokens is as likely as discovering a collision for a cryptographically secure hash function.

The coordinator's scalability depends on its organization and how computation is distributed among its servers. For hierarchically organized coordinators, token management is an embarrassingly parallel task. Token servers send at most one message per received token to another token server to verify the token's authenticity. As a result, each token server requires constant bandwidth regardless of the number of token servers. The root server, which aggregates summaries from token servers, only receives each peer's upload and download bandwidths during each epoch of approximately five minutes.

A hierarchical coordinator performs all computation at the root server using Antfarm, resulting in a potential CPU bottleneck. The root server performs two periodic CPU-bound tasks: it computes response curves from scatterplots,

and it allocates seeder and peer bandwidth. These tasks are computed centrally in order to derive bandwidth allocations based on the most recent measurements. Our implementation on a 2.2 GHz CPU with 3 GBytes of memory takes 6 seconds to perform these computations for 1,000,000 peers and 10,000 swarms whose popularities follow a realistic Zipf distribution. The root server can easily be replicated to mask network and host failures.

In contrast, a coordinator with a flat organization maintains a consistent view across all web servers and processors with a distributed state layer. Each deposited token results in at most two atomic writes to the state layer, one to update the depositing peer's balance, and one to update the block's propagation tree if the coordinator is tracking the block. The coordinator enables the system to scale, but extremely large deployments can cause contention on the state layer. Chapter 3.7 describes several optimizations to the state layer to reduce the load.

Our implementation of flat coordinators uses V-Formation to allocate bandwidth. V-Formation relies on a distributed set of processors in the coordinator to analyze block propagation trees, which eliminates the potential of a CPU bottleneck. Processors, like web servers, only contend for the state layer; they otherwise operate independently.

Chapter 4 shows that distributing the coordinator incurs negligible overhead and that the parallel nature of token management enables the system to grow linearly with the number of coordinator servers for both hierarchical and flat coordinators.

3.7 Engineering Challenges

When building any large system, there are always unforeseen challenges. FlixQ, our open content distribution system based on managed swarms [3], is no exception. This section discusses the challenges that we faced, and, in doing so, presents details of the organization and implementation of our own managed swarming deployment. We first discuss FlixQ's infrastructure. We then describe challenges building a scalable version of the V-Formation bandwidth allocation algorithm. Finally, we describe FlixQ's user interface, which enables users to add content, search for and download new content, and view downloaded content from a single location.

3.7.1 Infrastructure

FlixQ's server architecture implements a flat coordinator that uses distributed computation to allocate bandwidth, as described in Chapter 3.4.2. We implemented the coordinator in Python using the Django framework to dispatch requests to handler functions that perform the requested operations and render responses. FlixQ uses a MySQL database for persistent storage of shared content and user information.

We have found that FlixQ's MySQL database does not allow the system to scale in either the number of clients or the number of servers that comprise the coordinator. Instead, FlixQ's coordinator uses memcached to implement its distributed state layer. Memcached is a fast, distributed key-value store that is used to maintain swarm memberships, peer's CPMs, and block propagation

forests for tracking block transfers. Our memcached instance spans across our coordinator's web servers and processors. Objects in memcached—peer metadata, swarm metadata, and block propagation trees—are each stored on a single memcached instance determined from a consistent hash of its unique key.

FlixQ servers access memcached values frequently, and therefore it is critical that the key-value store remain consistent in the presence of concurrent writes. To support concurrency, memcached supports a compare-and-swap (CAS) operation that only writes a new value if the key-value binding has not been modified since a previously specified read of the same value. However, the Python bindings for memcached do not export the CAS operations. We extended the interface to support CAS and implemented a read_modify_write method that, given a memcached key and a function, applies the function to the associated value from memcached and replaces the stored value with the function's result. The method calls the given function and attempts to write the new value repeatedly, with exponential backoff, until the CAS succeeds. Finally, once the CAS has succeeded, the method performs side effects specified by the given function.

The read_modify_write method provides correct semantics for concurrent reads and writes, yet it limits scalability in write-heavy workloads due to contention on specific memcached values. By analyzing the coordinator's behavior in realistic deployments, we discovered that the vast majority of read_modify_write invocations were either attempting to write back the same value as it read, or attempting to write back a value with a logically equivalent mutation as a concurrently written version of the value. For instance, the coordinator is designed to recompute CPM values at most once per five minute interval, yet multiple invocations of read_modify_write that were concurrently attempting to update the

same CPM value would all run to completion, interfering with each other until they all succeeded. However, once one of the read_modify_write calls succeeds, the rest are no longer necessary because the newly written CPM value is sufficiently recent.

We implemented two mechanisms to reduce read_modify_write contention. First, atomic functions can raise a new abort exception to cancel the atomic operation mid-execution. The exception enables the function to short-circuit the encapsulating read_modify_write, causing it to return without modifying shared state. Second, read_modify_write accepts a new should_abort parameter that enables the caller to specify a condition that, if it ever holds between two attempts to execute the atomic function, causes automatic cancellation of the operation. Returning to the previous example, an attempt to atomically update a CPM value can be canceled if the value was last updated within the past five minutes, which the caller can specify in the should_abort parameter. With the abort parameter in place, once the first concurrently execution finishes, the remaining invocations will each terminate after at most one CAS failure.

3.7.2 Bandwidth Allocation

The FlixQ coordinator uses V-Formation to calculate bandwidth allocations by tracking block transfers among peers. Peers inform the coordinator of block transfers by depositing spent tokens at any web server in the coordinator. This section highlights a few challenges and implementation details of tracking tokens, selecting which blocks the coordinator tracks to compute bandwidth al-

locations, and how peers probe swarms for which the coordinator lacks recent data.

Token Management

Tokens are small, unforgeable strings that a peer can exchange for a block from another peer. The coordinator tracks the exchange of all tokens to track peer behavior and block exchanges. Therefore, the coordinator must be able to verify the authenticity of deposited tokens to prevent malicious peers from manipulating their service or the service of others. The naive solution, where the coordinator explicitly mints fresh tokens, sends them to peers, and records the tokens in the shared state layer for later verification, requires a considerable amount of outgoing bandwidth and memory to keep a record of all outstanding, unexpired tokens.

FlixQ reduces the coordinator's upload bandwidth and memory requirements by sending peers *token generators* in response to get_tokens requests instead of literal tokens. Token generators are secret keys that a peer can use a token generator create a sequence of valid tokens during the current *epoch*, after which time generated tokens expire.

The coordinator computes generators from a hash of the swarm, the requesting peer's identifier, and the current epoch, as well as a secret that only the coordinator knows. The number of tokens that a token generator can create depends on the number of tokens the peer requested and the peer's credit balance at the time of the request. The coordinator records the number of tokens that each peer is allowed to generate in the shared state layer, which corresponds

to range of legal serial numbers that a peer can use to generate each token. In addition, the coordinator maintains an initially zeroed *serial bitmap* to record which serial numbers the peer has used in the current epoch to prevent token double-spending and double-minting.

To create a token from a generator, a peer hashes the generator with a unique serial number that falls within the legal range of serial numbers specified by the coordinator. The peer also embeds the serial number itself, its own peer identifier, and the swarm's identifier into the token and exchanges it for a block. The token's recipient embeds its own peer identifier as well, along with the exchanged block's number, and deposits the token.

When the coordinator receives the spent token from the other peer, it is able to check its validity by performing the same computations: it recreates the token generator itself, then hashes that with the token's serial number and compares it with the hash in the token. Finally, the coordinator checks and updates the token's serial number in the peer's serial bitmap to ensure that the token is unique.

Tracking Blocks

The coordinator's web servers collect deposited tokens and select a subset of the tokens to build content propagation trees for computing CPM values. To control the amount of CPU required for the coordinator's processors, the coordinator uses an adjustable *stride* parameter that dictates the fraction of blocks that the coordinator tracks. In addition, for sufficiently small swarms, the coordinator can make efficient allocation decisions based on data that the coordinator al-

ready possesses about the swarms rather than tracking blocks in those swarms. The coordinator disables block tracking for such swarms to further reduce its CPU and memory requirements.

The stride parameter precisely determines the set of blocks that the coordinator tracks: a block is tracked if and only if its block identifier modulo the stride is zero, regardless of which swarm the block is from. Thus, a stride of five indicates that the coordinator tracks every fifth block of each piece of content. The stride's determinism enables each web server in the coordinator to autonomously decide whether a given token should result in an entry in the block propagation forest; likewise, each processor can iterate through identifiers of tracked blocks to continuously process block propagation trees.

Stride also determines how peers probe swarms to obtain fresh CPM values. The coordinator sends peers special probe flags to instruct them to probe swarms if the coordinator lacks current information on recent block exchanges. To probe a swarm, a peer prioritizes uploading a small, constant number of blocks to the swarm for the coordinator to track. Because the coordinator only tracks a subset of blocks for computing CPM values, a peer must be selective when choosing which blocks it uploads to probe a swarm. To ensure that the coordinator tracks the blocks that a peer uploads when probing a swarm, the peer uses the stride value to selectively upload blocks that will yield CPM data. As a result, peers expend a minimal amount of bandwidth probing swarms, leaving the remainder of their upload capacity for swarms that are known to propagate content efficiently.

Probing swarms and tracking blocks enables the coordinator to make efficient bandwidth allocation decisions, but for a subset of swarms, the coordi-

nator can use metadata on the swarm in lieu of tracking blocks. In particular, sufficiently small swarms, or swarms that contain sufficiently few downloading peers, have a theoretically bounded maximum CPM value that is less than measured CPM values of other swarms. For instance, a swarm with only three peers cannot produce a block propagation tree of size greater than two; therefore, if a member of that swarm is also a member of a swarm for which it produces block propagation trees of size three, the coordinator will not instruct the peer to probe the smaller swarm. This optimization ensures that peers' bandwidth is not wasted taking unnecessary measurements, which has large impact in deployments with long content popularity tails.

3.7.3 User Interface

FlixQ's user interface dictates how users interact with the system to add content, discover new content, download content, and manage their libraries of shared and downloaded content. The user interface plays a critical role in determining who uses FlixQ and how they use it. It has been a challenge to develop a simple, intuitive interface that facilitates both traditionally web-based operations, such as searching for new content, and functionality that is traditionally managed by standalone software, such as exchanging content blocks with peers and managing downloaded files on disk.

FlixQ uses a purely web-based interface for all operations. To support operations normally handled by standalone software, such as manipulating the file system and opening connections to peers, we developed the FlixQ *sidecar*. The sidecar is software that the user downloads and installs, but which runs

silently in the background and is controlled exclusively through FlixQ's web interface. At its core, our implementation of the sidecar uses an open-source implementation of the BitTorrent protocol, from the BitTornado codebase [1], with modifications to adhere to the V-Formation protocol and to communicate with a locally running browser on the FlixQ website. Communication between the FlixQ website in the user's web browser and the sidecar occurs transparently using asynchronous AJAX requests.

By pushing peer-to-peer functionality into the sidecar, the browser functions only as a user interface; the user can close the browser at any time without disrupting service. An in-browser implementation of the V-Formation protocol would require that the user keep the browser open to engage in data transfers, and by navigating away from the site, the user would lose TCP connections to peers, tokens, and partially downloaded blocks.

In addition to controlling the sidecar, the website facilitates content discovery. The website maintains a MySQL database with metadata about all content. For each piece of content, FlixQ servers store a metadata file analogous to a BitTorrent .torrent file, which specifies the content's metadata and block hashes. To further simplify FlixQ's interface of peer-assisted downloads, the browser and sidecar automatically manage the transfer and processing of metadata files. When users share new content, the sidecar automatically generates a metadata file and uploads it to FlixQ's servers; likewise, when users download a file from the website, the browser instructs his sidecar to download the corresponding metadata files and manage the downloads.

Overall, a website interface simplifies the user experience for managed swarming systems. The streamlined approach combines the flexibility of stan-

dalone software, because of the fully privileged sidecar, with the simplicity of web-based applications, with which many users are familiar.

3.8 Summary

This chapter introduced several engineering tradeoffs in managed swarming systems that affect efficiency, scalability, and interoperability with existing protocols. We discussed two wire protocols and two implementations of the logically centralized coordinator that we have built. The coordinator designs differ in how they aggregate block transfer statistics and whether the computing bandwidth allocations is performed on a single physical host or distributed across multiple servers. In addition, we described the design of a decentralized coordination scheme for replacing the coordinator when using V-Formation in environments where hosts implicitly trust each other. In this scheme, peers track their own block transfers, calculate their own CPM values, and use those values to determine their own efficient allocation of bandwidth among swarms. Finally, this chapter discussed the security and scalability of managed swarms, as well as practical engineering considerations for building a large managed swarming system that we encountered in our own large-scale deployment.

CHAPTER 4

EVALUATION

This chapter explores the performance of managed swarms across a large range of deployment scenarios. It examines managed swarms with respect to the primary goal of maximizing aggregate download bandwidth for end users, as well as secondary goals such as fairness and preventing starvation. Through extensive simulations and PlanetLab [13] deployments, we show how managed swarms react to churn and flash crowds, and how they handle scenarios where bandwidth from cache servers is scarce and plentiful. Our experiments compare the performance of both Antfarm and V-Formation with a client-server approach, BitTorrent version 5.0.9, and a peer-to-peer approach that allocates bandwidth among multiple swarms using a global rarest block selection policy.

4.1 Simulations

We have developed a simulator for fine-grain analysis of managed swarming systems. The simulator implements the full Antfarm and V-Formation protocols described in this thesis, as well as BitTorrent, a BitTorrent-like *global rarest* policy where peers request the rarest blocks for which they are interested across all swarms, and a traditional client-server system. We have calibrated our simulator so that it behaves comparably to live deployments of the protocols. Our simulation results first show the end-to-end performance of managed swarming bandwidth allocations and compare them to other swarming protocols. Then, we present microbenchmark results for Antfarm and V-Formation to better understand how they allocate bandwidth among competing swarms. The microbenchmarks examine how the algorithms converge on stable bandwidth al-

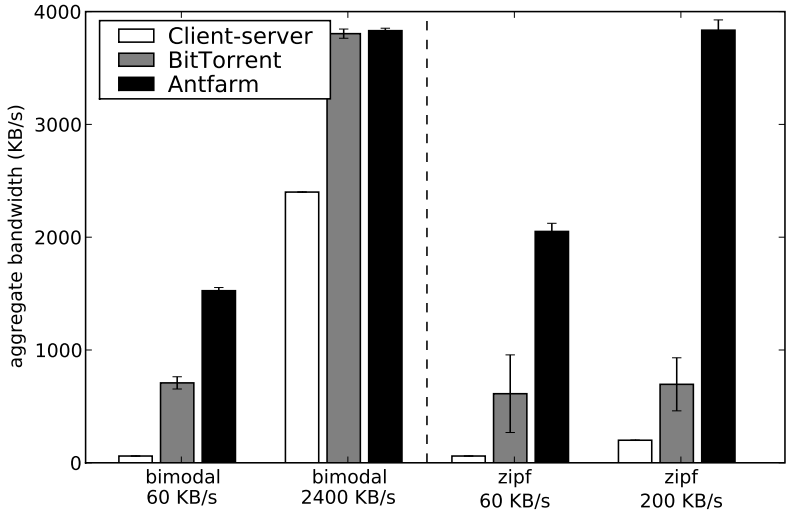

Figure 4.1: **Aggregate bandwidth for a client-server system, BitTorrent, and Antfarm.** When seeder bandwidth is plentiful, even a client-server model can deliver high throughput. When seeder bandwidth is limited, Antfarm outperforms BitTorrent by allocating bandwidth to swarms that receive the most benefit. Error bars indicate 95% confidence.

locations, handle churn, and dynamically adjust parameters to improve performance.

4.1.1 End-to-End Performance

The primary goal of managed swarming deployments in this thesis is to maximize system-wide aggregate bandwidth. We first compare managed swarming systems to traditional swarming systems such as BitTorrent, and to client-server approaches. We use the single-seeder multi-swarm content distribution problem to compare Antfarm to BitTorrent. Then we compare those protocols to V-Formation, which addresses the more flexible general multi-swarm content distribution problem.

The differences between Antfarm and BitTorrent in a multi-swarm setting stem from the way the two protocols allocate their bandwidth to competing swarms. Whereas BitTorrent seeders allocate their bandwidth greedily to peers that absorb the most bandwidth, Antfarm allocates the precious seeder bandwidth preferentially to swarms whose response curves demonstrate the most benefit. As a result, there is a qualitative and significant difference between the two protocols; under some scenarios, BitTorrent can starve swarms and perform much worse than Antfarm, while in others with ample bandwidth, seeder allocation may have little impact on client download times. Figure 4.1 shows Antfarm's performance in comparison to BitTorrent and a traditional client-server system similar to YouTube for two swarm distribution scenarios. In the *bimodal* scenario, there is a single swarm of 30 peers and 30 swarms of one peer each. The *Zipf* scenario comprises swarms of sizes 50, 25, 16, 12, 10, 8, and 5, and 400 singleton participants. Each set of three bars shows the average aggregate bandwidth for a corresponding scenario and seeder bandwidth.

Overall, Antfarm achieves the highest aggregate download bandwidth. In scenarios where there is ample seeder bandwidth, the differences between the three systems are negligible and even a client-server approach is competitive with BitTorrent and Antfarm. As available seeder bandwidth per peer drops, however, swarming drastically outperforms the client-server approach, highlighting the efficiency of peer-to-peer over a client-server system using comparable resources. For the scaled-down but realistic Zipf scenario, Antfarm achieves a factor of 5 higher aggregate download bandwidth than BitTorrent. BitTorrent misallocates bandwidth by preferentially unchoking hosts based on their recent behavior, regardless of their potential to share blocks. In contrast,

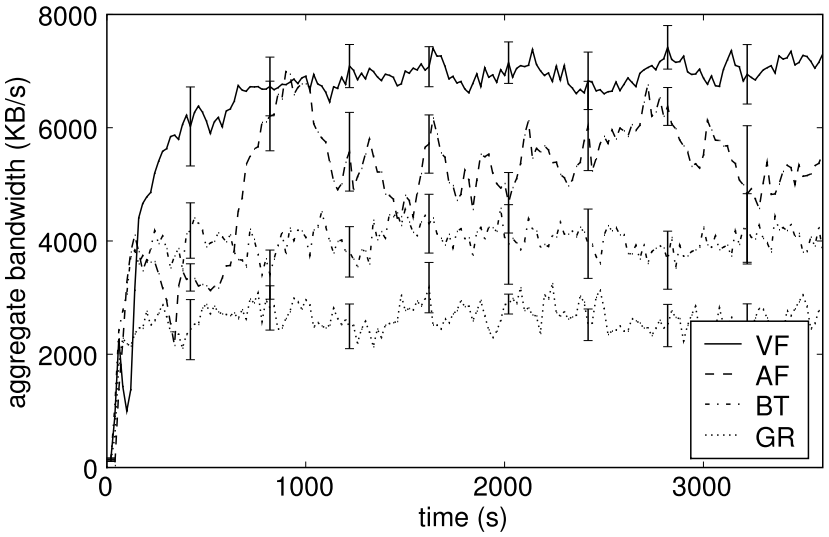

Figure 4.2: **Comparison of protocols over time.** Peers download movies with lengths and popularities randomly drawn from the Internet Movie Database. Peers have link capacities drawn from a distribution determined by a BitTorrent measurement study. Error bars indicate 95% confidence.

Antfarm measures response curves and uses them to steer the single seeder's capacity to swarms where blocks can be further shared among peers.

V-Formation computes CPM values for each host to allocate bandwidth rather than explicitly measuring response curves. We show the system-wide aggregate bandwidth of V-Formation, Antfarm, BitTorrent, and the 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. This is in contrast to the previous experiment, where only a single designated host possessed all the content in the system. Peer upload capacities are drawn from the distribution of BitTorrent peer bandwidth collected by Pouwelse et al. [95]. 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. 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 4.2).

As before, 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 for each host, it reaches a stable allocation of bandwidth four times faster than

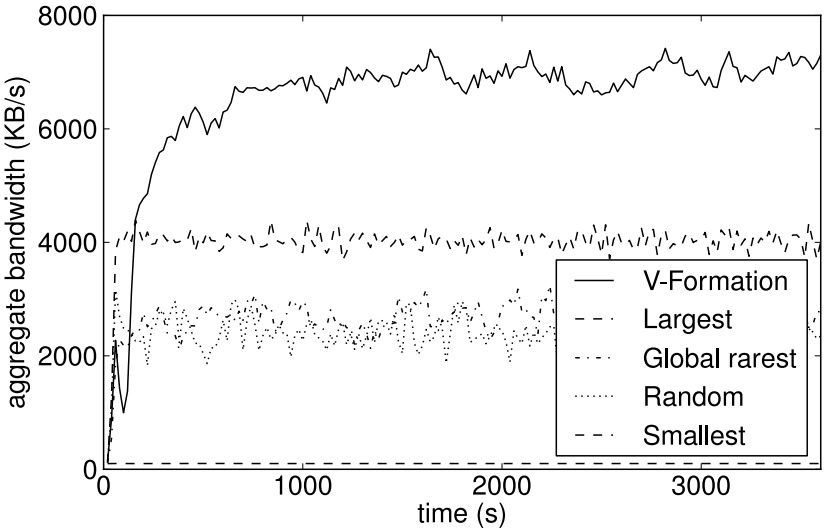

Figure 4.3: **The CPM versus simple heuristics.** V-Formation achieves significantly higher aggregate bandwidth by allocating hosts' bandwidth using the CPM than deployments where hosts instead base their allocation decisions on simple heuristics.

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.

The CPM enables hosts to make principled decisions based on the observed behavior of swarms. We compared the performance of V-Formation using the CPM with the performance of deployments where hosts allocate their bandwidth using simple heuristics based on block popularities and swarm sizes (Figure 4.3). The experimental setup is equivalent to that of Figure 4.2, where swarm popularities are determined from a random sampling of IMDb movies. By enabling hosts to measure their actual benefit to swarms, V-Formation achieves significantly higher aggregate bandwidth than straightforward heuristics such as preferring large swarms over small swarms, always satisfying requests for globally rare blocks over requests for more popular blocks, and distributing bandwidth randomly among peers.

4.1.2 Antfarm Microbenchmarks

The Antfarm coordinator continuously measures and refines response curves for each swarm to maintain high performance as swarm dynamics change. Through a series of small-scale experiments, we examine the strategies that Antfarm uses to allocate bandwidth and avoid starvation.

Figure 4.4 shows Antfarm's bandwidth allocation over time to provide insight into Antfarm's strategy. The top graph shows that when seeder bandwidth is plentiful, Antfarm spends the vast majority of its bandwidth on small swarms since they receive the most marginal benefit. When seeder bandwidth is constrained, as shown in the bottom graph, Antfarm achieves high aggregate bandwidth by preferentially seeding large swarms that can leverage their upload capacity to multiply the benefits from the seeder. As peers of the large swarm complete their downloads at 5000 seconds, the seeder shifts its bandwidth to the singleton swarms. The staircase behavior is due to different swarms completing at different times.

When swarms become large, they are often able to saturate their peers' uplinks, and sometimes even their downlinks, without the aid of seeder bandwidth. Such *self-sufficient* swarms yield flat response curves. Antfarm's allocation strategy naturally avoids dedicating bandwidth to self-sufficient swarms when there are other swarms that can benefit more. In contrast, BitTorrent does not take swarm dynamics into account, and can end up dedicating seeder bandwidth without consideration of available peer bandwidth, leading to a shortage of seeder bandwidth for other, needier swarms. Figure 4.5 shows an exaggerated scenario that illustrates this effect. The figure shows the average download bandwidths of peers in BitTorrent and Antfarm of the two swarms. In this sce-

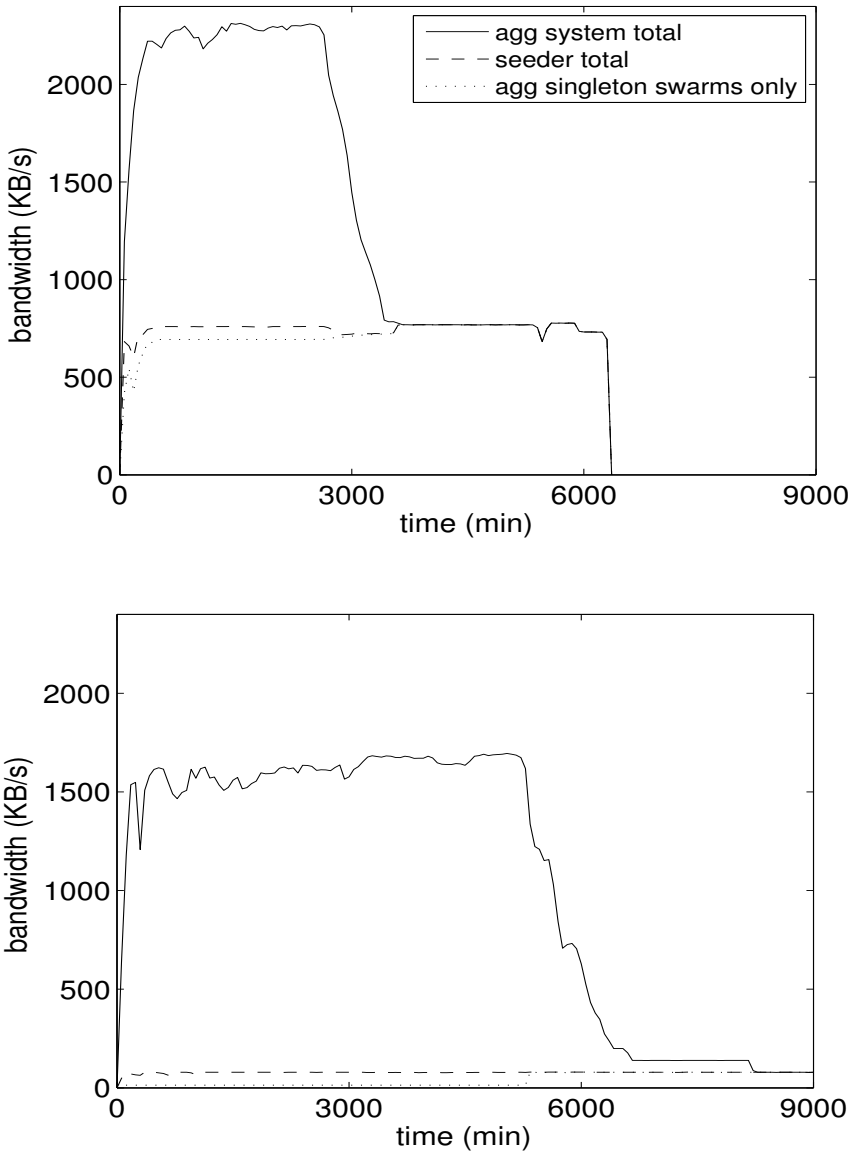

Figure 4.4: **Antfarm's bandwidth allocation over time.** The figures show seeder and aggregate bandwidths of the bimodal experiment with seeder bandwidths of 800 KBytes/s (top) and 80 KBytes/s (bottom). Antfarm follows drastically different bandwidth allocation strategies (dashed and dotted lines) to achieve high throughput (solid lines).

nario, the seeder has a capacity of 100 KBytes/s, and each peer downloads a 10 MB file with 30 KBytes/s download capacity and 10 KBytes/s upload capacity. The self-sufficient swarm saturates peers' uplinks without seeder bandwidth and has a fresh peer arrive every second, resulting in a swarm of approx-

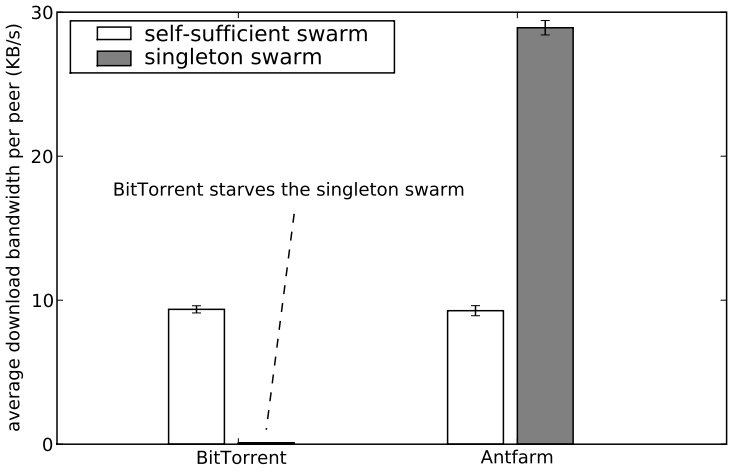

Figure 4.5: **BitTorrent's and Antfarm's allocations to a singleton swarm and a large, self-sufficient swarm.** Even though a self-sufficient swarm can saturate its peers' bandwidth without seeder bandwidth, BitTorrent awards bandwidth to peers in the swarm. In contrast, Antfarm awards seeder bandwidth to the singleton swarm because it receives high marginal benefit.

imately 1000 peers. The Antfarm coordinator determines that the self-sufficient swarm does not benefit from seeder bandwidth, and awards bandwidth to the singleton swarm instead. Under Antfarm, the singleton peer is able to complete its download in an average of six minutes. BitTorrent fails to provide the singleton swarm any bandwidth over the course of the 20-minute simulation.

The problems with BitTorrent's allocation strategy are compounded in larger, more realistic scenarios. While large swarms are often self-sufficient, smaller non-singleton swarms can receive large multiplicative benefits from the seeder because their peers have available upload capacity to forward blocks. In contrast to the previous experiment, which examined the impact on a swarm at the tail end of the popularity distribution, Figure 4.6 illustrates the impact of seeder bandwidth allocation on a file of medium popularity. The figure shows the total amount of seeder bandwidth that Antfarm and BitTorrent al-

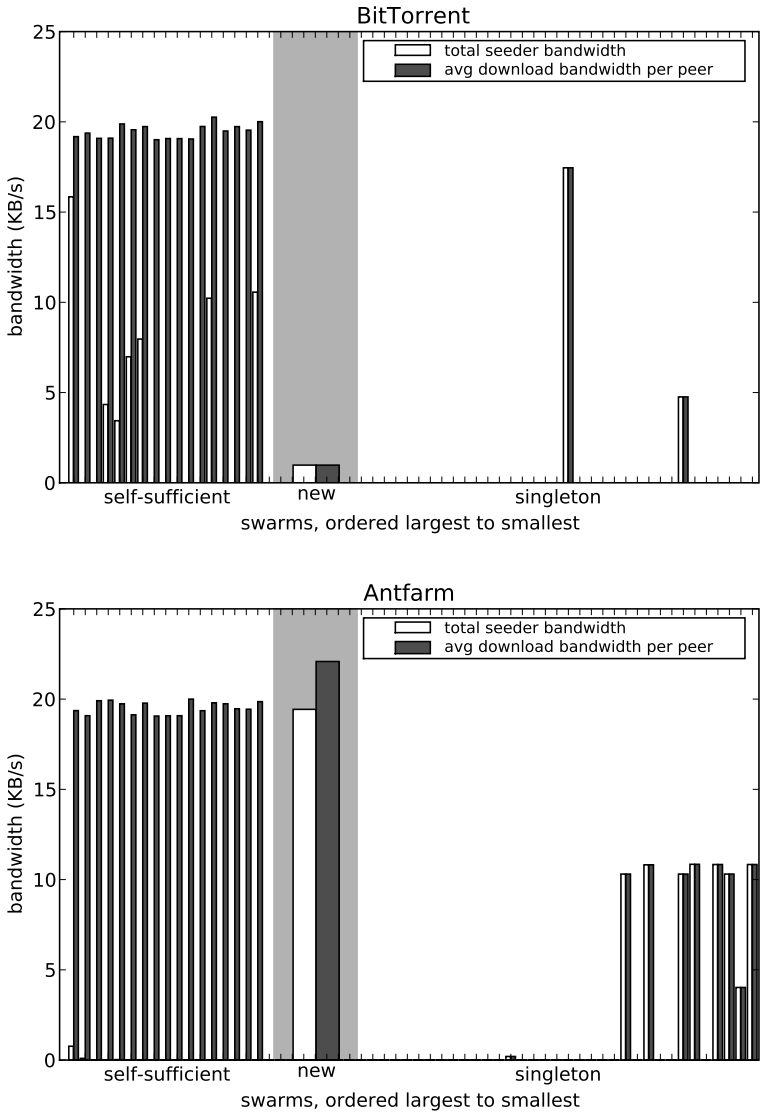

Figure 4.6: **BitTorrent versus Antfarm serving the middle of the popularity distribution.** The shaded region indicates a new swarm of 5 peers. Swarms to its left are self-sufficient; swarms to its right are singletons. BitTorrent (top) starves the new swarm, favoring to dedicate bandwidth to the many peers in self-sufficient swarms. Antfarm (bottom) allocates enough seeder bandwidth to the new swarm to saturate its peers' upload bandwidths, and allocates the rest to singleton swarms because they receive high marginal benefit.

locate to a set of self-sufficient swarms, a new swarm of 5 peers, and 32 singleton swarms. It also shows the resulting average download bandwidths of peers in each swarm. The peers have 30 KBytes/s download capacities and

20 KBytes/s upload capacities, and the self-sufficient swarms have peer inter-arrival times of 3, 6, 12, 24, 50, and 100 seconds. In the top graph, BitTorrent dedicates almost all of its bandwidth to the self-sufficient swarms, whose peers are already saturated, and some randomly to singleton swarms, which are unable to forward blocks. The bottom graph shows that Antfarm awards enough bandwidth to the new swarm to saturate its peers' uplinks and dedicates the rest of its bandwidth evenly among several singleton swarms because they receive high marginal benefit. BitTorrent's optimistic unchoking protocol causes it to dedicate its bandwidth to only a few singleton swarms over the 20 minute simulation. Overall, Antfarm achieves an order of magnitude increase in average download speed for the affected swarms without a corresponding penalty for the popular swarms.

Overall, Antfarm qualitatively outperforms BitTorrent in a multi-torrent setting by allocating bandwidth based on dynamically measured response curves and preferentially serving those swarms that benefit most from seeder bandwidth.

4.1.3 V-Formation Microbenchmarks

Unlike Antfarm, V-Formation addresses the more flexible general multi-swarm content distribution problem by calculating a bandwidth allocation for all hosts that belong to more than one swarm. To show that these hosts do not interfere with each other while the coordinator computes CPM values, this section examines how V-Formation converges on a stable bandwidth allocation in a variety

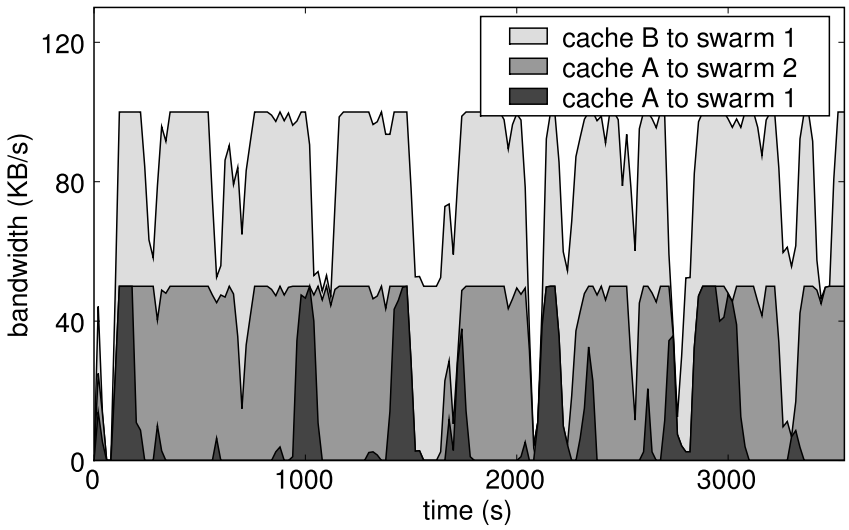

Figure 4.7: **V-Formation's bandwidth allocation to a pair of swarms.** Two swarms of similar size achieve comparable aggregate bandwidth even though cache B does not possess content for swarm s_2. Cache A gives more bandwidth to s_2 (medium gray area) than to s_1 (dark gray area) to compensate.

of scenarios. We also show how V-Formation's measurement time interval τ influences the coordinator's calculations.

In a large deployment, V-Formation and Antfarm can achieve different aggregate bandwidths due to constraints imposed by individual peers' bandwidths and swarm memberships. From the end-to-end experiment shown previously in Figure 4.2, 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 4.7). 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 2.8). 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 4.8). The top graph shows p_1's CPM values for its swarms over time; the bottom graph shows the same for p_2. When the simulation begins, p_1 and p_2 probe their respective swarms

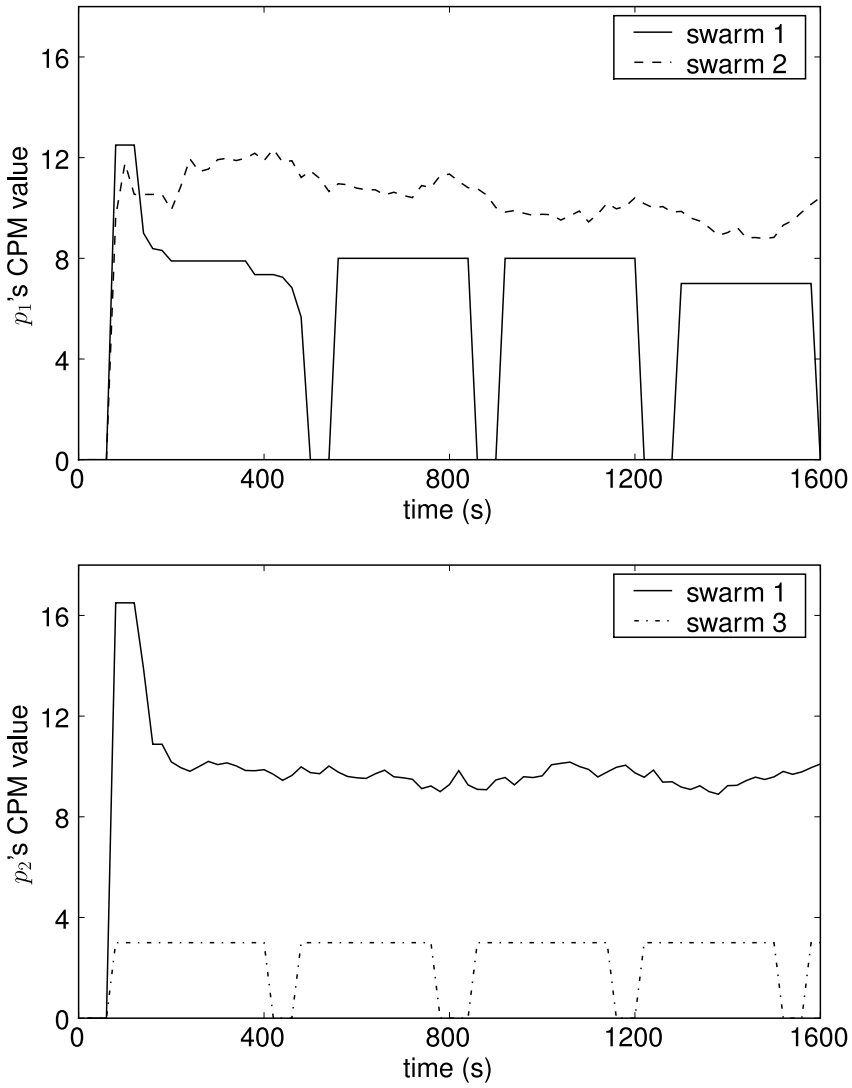

Figure 4.8: **CPM values for competing swarms.** Peer p_1 has cached content for two large swarms s_1 and s_2 (top), and peer p_2 has content for s_1 and a small swarm s_3 (bottom). p_2 creates competition for block uploads in s_1, causing p_1 to upload to s_2.

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 pro-

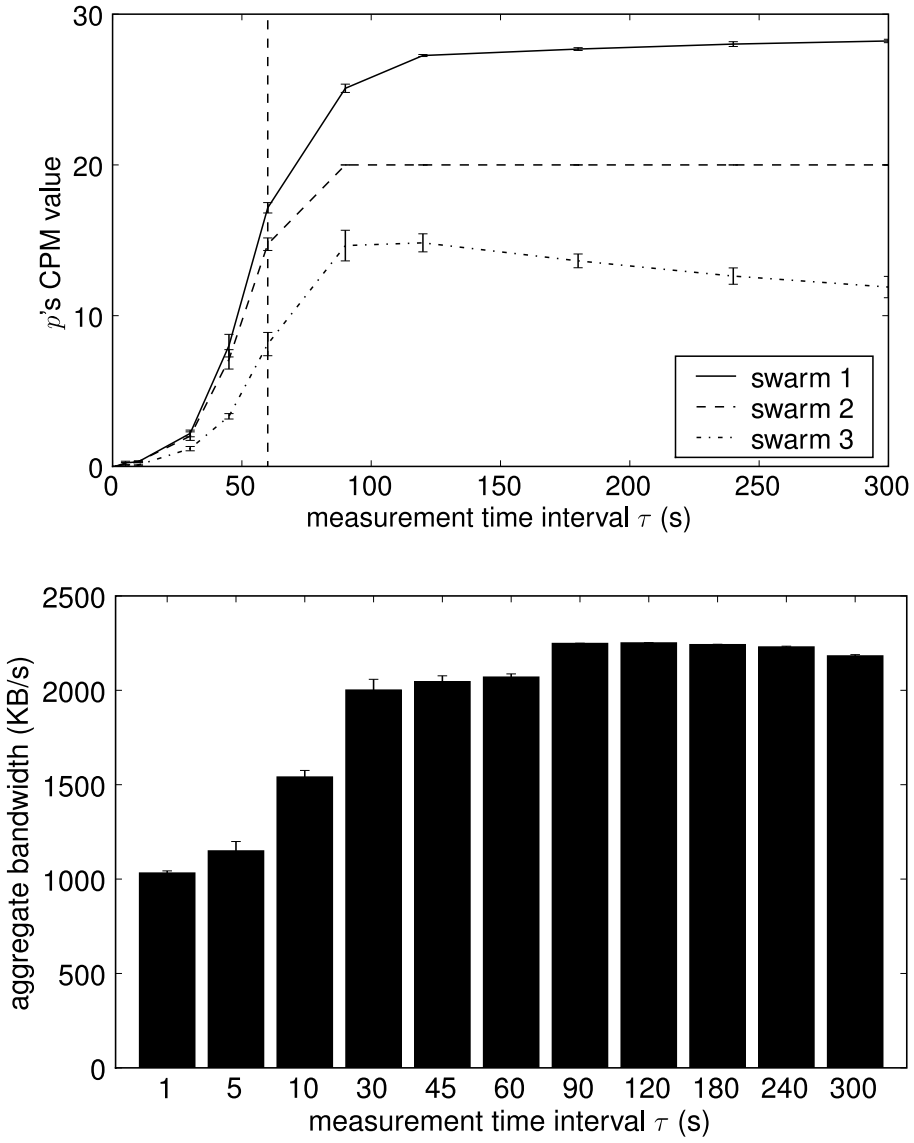

Figure 4.9: **V-Formation's sensitivity to the measurement time interval.** Competing swarms are indistinguishable for small τ. Sufficiently large τ enables peer p to discover the swarm that receives the most benefit from its blocks. The vertical dashed line shows the coordinator's choice of τ based on the measurements. Error bars indicate 95% confidence.

viding 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 4.9). The top 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 bottom 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 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 4.10). At 3000 sec-

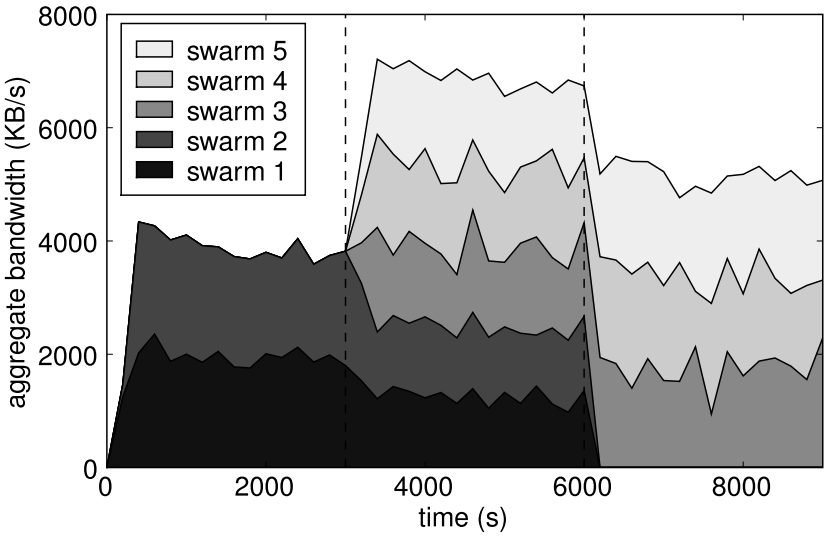

Figure 4.10: **V-Formation's convergence and stability.** Three cache servers concurrently upload to two, five, and three identical swarms, indicated by shaded areas. The protocol adapts quickly to the changes introduced at the dotted lines, achieving equal aggregate bandwidth across swarms in each interval.

onds, 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 4.11). 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.

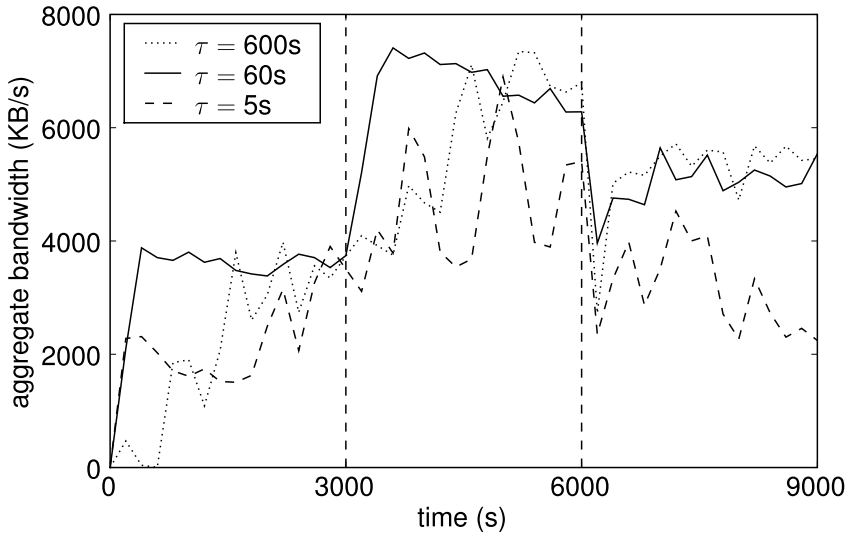

Figure 4.11: **Time measurement interval and churn.** Swarm memberships change drastically at the two vertical dashed lines. A measurement time interval τ that is too large or too small results in suboptimal performance. V-Formation chooses τ to maximize aggregate bandwidth.

4.2 Live Deployment

We deployed both Antfarm and V-Formation on PlanetLab to measure their performance in real swarms. The V-Formation measurements are based on our active deployment of the algorithm in FlixQ, our open content distribution system [3]. We compare the performance of both bandwidth allocation algorithms as well as version 5.0.9 of the official BitTorrent client. Then we examine the scalability of our logically centralized coordinators, both a hierarchical coordinator suitable for Antfarm, and a flat coordinator used by V-Formation.

4.2.1 End-to-End Performance

We first examine the end-to-end performance of Antfarm and V-Formation in realistic deployments on PlanetLab. Antfarm relies on accurate response curves

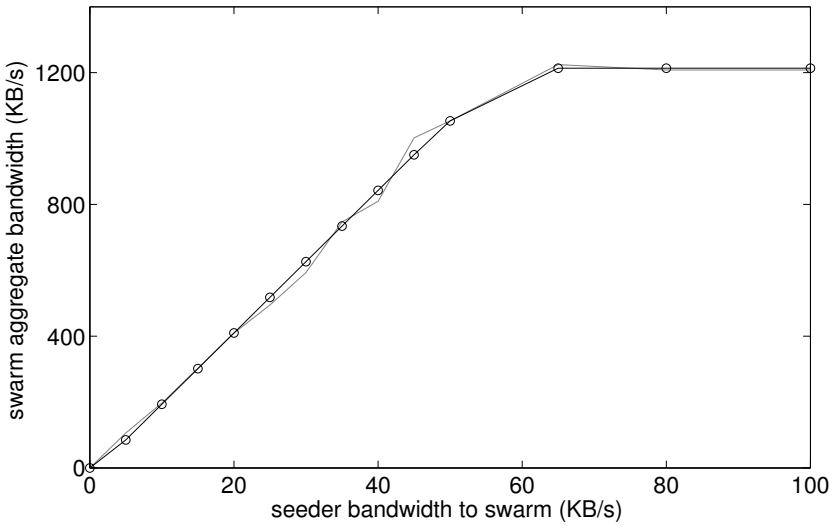

Figure 4.12: **A measured response curve.** The swarm consists of 25 PlanetLab nodes, each with an upload capacity of 50 KBytes/s. Each data point is based on the average swarm aggregate bandwidth over 10 minutes. Real-world response curves confirm simulations.

to make its bandwidth allocations. A response curves measures the designated seeder's impact on a particular swarm. To demonstrate Antfarm's response curves in practice, Figure 4.12 shows a measured response curve of a swarm comprised of 25 PlanetLab nodes, each with an upload capacity of 50 KBytes/s. The graph plots both the response scatterplot and the response curve as computed by the coordinator from the token exchange. The results confirm the simulations and our theoretical response curve model of swarm behavior.

Figure 4.13 compares the aggregate bandwidth achieved by Antfarm, BitTorrent, and traditional client-server downloads in a single-seeder deployment, such as a closed content distribution system. The experiment runs across 300 PlanetLab nodes, each with an upload capacity of 50 KBytes/s. Swarms have size 100, 50, 25, 12, 6, 3, and 1. In practice, the stock BitTorrent implementation uploads only a few hand-picked files concurrently; to evaluate BitTorrent in the presence of many swarms, we measured two values by running multiple

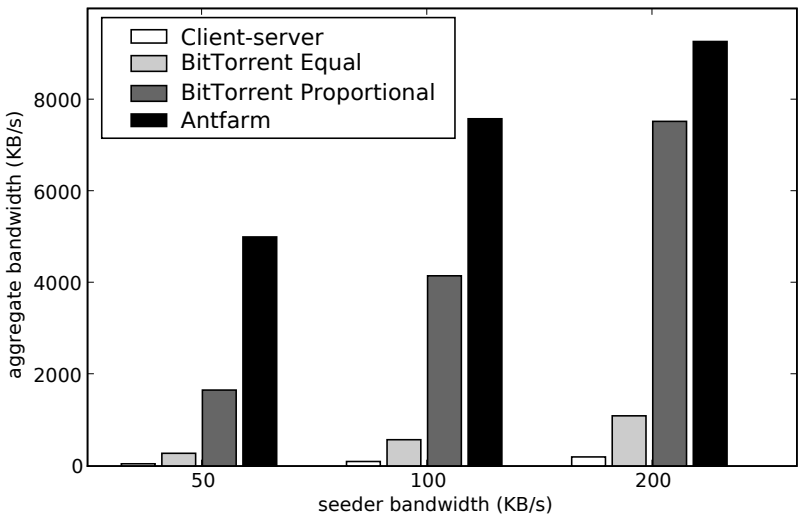

Figure 4.13: **Performance of Antfarm, BitTorrent, and client-server in a closed content distribution system.** 300 PlanetLab nodes are distributed among swarms ranging in size from 1 to 100. Antfarm achieves high average performance by making efficient use of limited seeder bandwidth.

seeder instances, each with its own upload capacity. *BitTorrent Equal* indicates the aggregate system bandwidth when the BitTorrent seeder splits its upload bandwidth equally among all swarms, including singleton swarms. *BitTorrent Proportional* shows performance when the seeder allocates to each swarm an upload bandwidth proportional to the size of the swarm.

Antfarm outperforms BitTorrent by allocating seeder bandwidth to the swarms that receive the most benefit. Antfarm's advantages over BitTorrent become more pronounced in systems with many swarms accompanied by relatively small seeder uplink capacities, a realistic scenario for a distribution center with a large number of files and a bandwidth bottleneck. In these experiments, Antfarm outperforms traditional client-server by a factor of between 50 and 100, BitTorrent Equal by a factor of 8 to 18, and BitTorrent Proportional by a factor of 1.2 to 3.

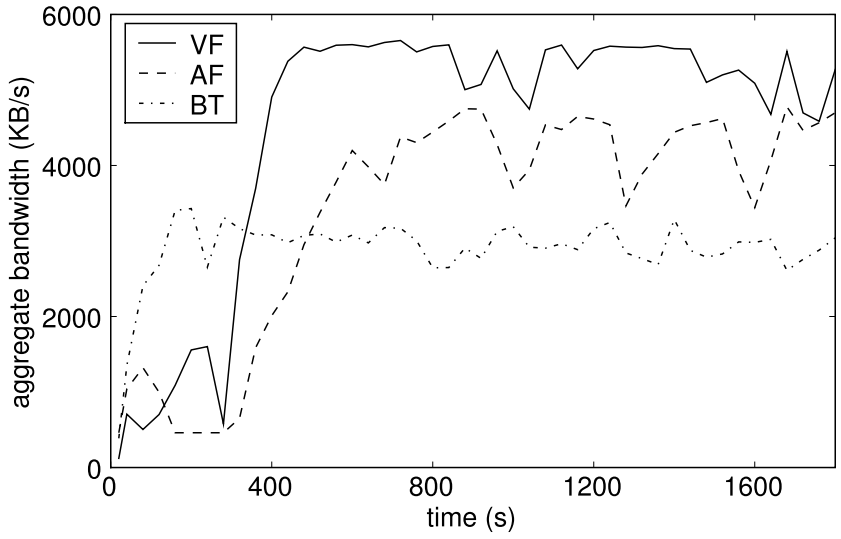

Figure 4.14: **Performance of V-Formation, Antfarm, and BitTorrent in an open content distribution system.** 380 nodes in 200 swarms download movies from FlixQ using the V-Formation protocol and the same movies using the Antfarm and BitTorrent protocols. 20% of peers belong to multiple swarms.

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. Nodes draw their bandwidth distribution from the measured BitTorrent bandwidth distribution. The results of the experiment (Figure 4.14) 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.

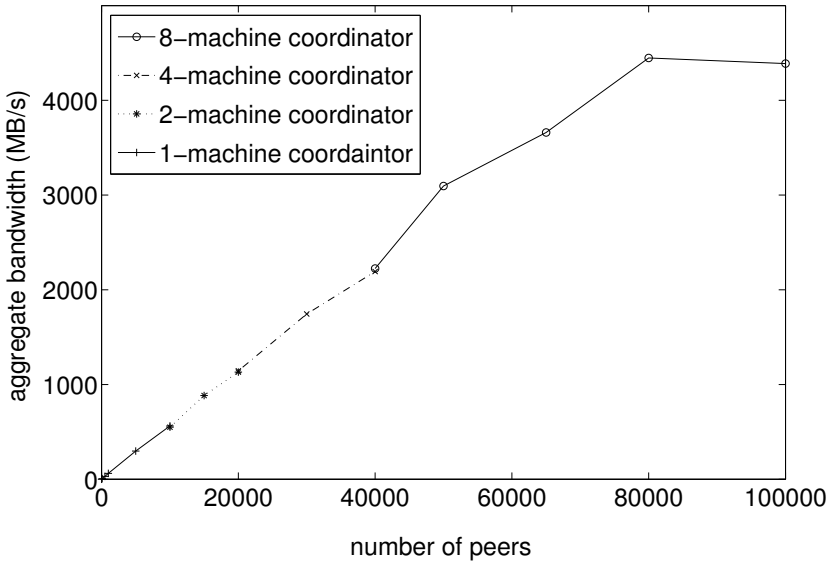

Figure 4.15: **Aggregate bandwidth of swarms managed by a hierarchical coordinator of varying sizes.** Each coordinator host runs on a PlanetLab node with an artificial bandwidth cap of 100 KBytes/s to limit scalability. The task of the token coordinator is embarrassingly parallel; the system capacity scales linearly with the size of the coordinator cluster.

4.2.2 Scalability

We next examine how the logically centralized coordinators scale. We examine the steady-state bandwidth cost of running a distributed coordinator, both when they are organized hierarchically, and when they have a flat organization.

Figure 4.15 shows the bandwidth consumption of a hierarchically organized coordinator running Antfarm as a function of the number of peers based on experiments run on PlanetLab. In the experiment, the coordinator's root server and each token server ran on its own PlanetLab node, and peers were simulated across other PlanetLab nodes. To maximize generated load, peers omit the data exchange but engage in the token protocol with the coordinator. Further, we artificially limit the bandwidth available to each physical coordinator

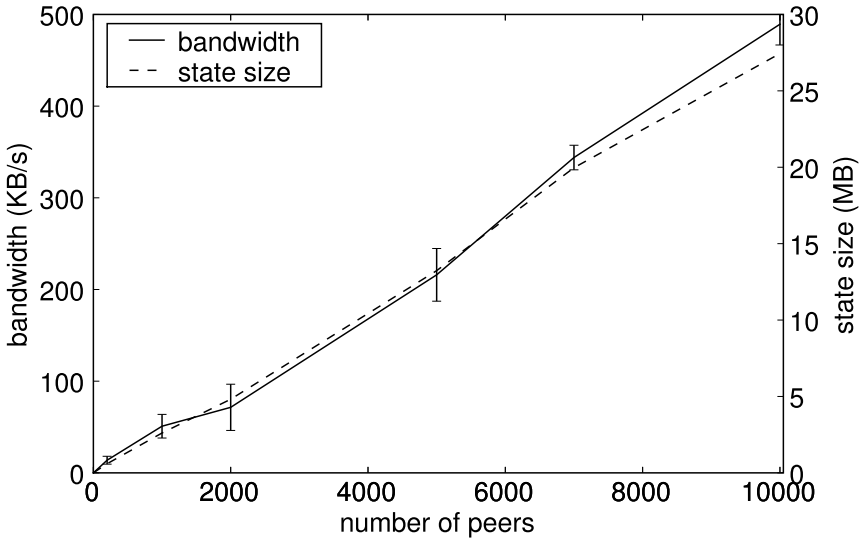

Figure 4.16: **Scalability of a flat coordinator.** Both memory consumption and bandwidth at the coordinator scale linearly with the size of the system.

host to 100 KBytes/s to gain insight into the performance of multiple coordinator nodes running with severe bandwidth constraints. The bottom curve shows the capacity of a single, artificially-bottlenecked coordinator host acting as both the root server and a token server. It is able to handle the tokens and peer lists of approximately 9000 peers before its performance reaches a plateau. Adding a second token server doubles the capacity of the system. Because the token servers engage in a massively parallel task with little communication overhead, increasing the number of token servers linearly increases the maximum supported number of peers.

The next experiment examines how our implementation of a flat 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 4.16). 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 end-to-end PlanetLab experi-

ment. 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_tokens 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 Chapter 3.4.2. Coordinator bandwidth includes all outgoing tokens, CPM values, and responses to announce requests.

Both hierarchical and flat coordinators achieve linear scalability in the size of the system, making them suitable for large deployments. The hierarchical coordinator is able to support a larger number of peers per coordinator host because the coordinator's token servers operate independently with the exception of small, periodic summaries of only a few bytes per peer to the root server. In contrast, a flat coordinator relies on a distributed, shared state layer to maintain a consistent view of the system. Consequently, the distributed state layer can pose a bottleneck when there is a high volume of incoming requests.

CHAPTER 5

RELATED WORK

Managed swarms lie at the intersection of a large body of prior work, including content distribution in general, and peer-to-peer swarming systems in particular. Research on measuring and modeling existing peer-to-peer systems has characterized peer behavior and the availability of content. In peer-assisted systems, content availability is of critical importance due to the long tail in content popularity distributions. Efforts to improve content availability have sparked research in efficient caching, analysis of the lifetimes of BitTorrent swarms, and incentivization for peers to contribute their resources to improve the performance of swarming systems. Exploring and examining incentives and currency in peer-to-peer systems has become a research area of its own, with the overarching goal of fostering cooperation among end users, who each have their own selfish goals. Finally, recent literature has focused on streaming systems that support video-on-demand applications. Such systems have implemented hybrid, peer-assisted architectures that have similarities to the work in this thesis.

5.1 Content Distribution

Content distribution networks are scalable systems used to alleviate server load, reduce download times, and avoid network hotspots. Akamai [42], for example, is a widely deployed infrastructure-based CDN that many content providers rely on to distribute their content. Similarly, cooperative web caching [16, 62, 52, 125, 127] removes load from origin servers. ECHOS [78] proposes distributing servers using a peer-to-peer network of set-top boxes distributed at the Internet's periphery, managed by a single entity that can op-

timize system performance, but does not address bandwidth management at the servers. Although distributed CDNs scale, the bandwidth cost of operating them resides entirely with the content provider and distributor.

CobWeb [120], CoBlitz [96], and Coral [49] are HTTP-based content distribution networks that cache and serve files from distributed infrastructure hosts. This reduces load on origin servers and enables modestly provisioned hosts to serve popular content to many clients. Such systems use distributed hash tables based on consistent hashing [68], which enables clients to locate objects efficiently by hash. Corona [112, 106] similarly uses consistent hashing to provide a publish-subscribe system for the web that minimizes requests to servers to achieve a target update latency for subscribers. Finally, Bamboo [113] provides a distributed hash table that operates reliably in the presence of churn, critical when storage nodes include end users.

Applications that sit above the infrastructure layer often have their own individual constraints. Chen et al. [25] develops a general framework that enables multiple applications that compete for bandwidth to specify their own bandwidth requirements. The system then generates a topology of hosts that satisfies the each application's bandwidth constraints.

Efficient caching of content is critical to the performance of content distribution systems. Caching content among infrastructure hosts can improve latency by locating content close to end users, and it can improve throughput and alleviate hotspots by distributing client requests over multiple hosts. Applegate et al. [6] propose an algorithm for choosing which cache servers in a deployed infrastructure should cache each of a set of media files from a large content library. The algorithm frames the problem as one of optimization where the goal is to

minimize the network bandwidth required to satisfy all client requests for content given the available disk space constraints of each host. The solution does not consider the cost of moving content among cache servers because, as the authors claim, redistributing content among caches more than once per week yields little performance benefit. In contrast, Qiu et al. [109] instead focus on where content distributors should place their cache servers within their infrastructure. Huang et al. [58] explicitly cache content at peers by allocating a fixed, 1 GByte cache on each participant's disk for storing and serving content.

In order to redistribute content among cache servers, Fastreplica [20] provides a simple, efficient algorithm for replicating a large file among a fixed number of infrastructure hosts. In the first of two phases, the origin host in possession of the file splits the file into equally sized pieces, one for each replica, and sends each replica its own piece. Then, in the second phase, each replica sends the piece that it received to each of the other replicas. The result is efficient utilization of the n^2 links connecting a set of n replicas.

To leverage caches for faster downloads, Rodriguez et al. [111] propose enabling a client downloading a large file to simultaneously pull content from each of multiple cache servers. This prevents a single client from placing high load on a single server. Similarly, Shark [8] and ChunkCast [18] reduce client-perceived download latency by enabling clients to efficiently find nearby copies of content in a structured overlay network. Given a set of caches and their cached content, Alzoubi et al. [11] routes clients' requests among the cache servers in order to minimize the load on any single cache server.

Peer-to-peer CDNs, discussed in detail below, effectively shift bandwidth costs from the content provider to clients. Hybrid systems, which combine

peer-to-peer data exchange with data from server-class hosts, have enormous potential to reduce cost for both content distributors and ISPs, and to improve performance for end users [55, 54, 67]. For instance, Slurpie [117] combines a mesh network with a traditional client-server architecture to reduce server load for bulk downloads of large, popular files. Hei et al. [57] provide insights for future system designers based on extensive measurements from live traces from the PPLive system [5].

5.2 Single-Swarm Systems

Peer-to-peer content distribution constitutes more than half of all Internet traffic [4], the majority of it coming from swarming systems where end users organized into unstructured mesh networks exchange data blocks with one another. BitTorrent [17] is the most popular swarming system, and studies consistently show that BitTorrent traffic constitutes a significant fraction of Internet traffic [95, 119].

Due to the popularity of peer-to-peer protocols, researchers have examined the nature of peer-to-peer systems in general, and swarming, BitTorrent-like systems in particular, in order to better understand and improve their performance. The vast majority of this work has focused on the interactions of peers within a single swarm, a difficult problem due to the dynamic nature of swarms and the unpredictable behavior of individual peers.

A large number of measurement studies and mathematical models have enumerated the strengths and weaknesses of the BitTorrent protocol within a single swarm. Zhang et al. [130], in the largest BitTorrent measurement study

to date, shows a large variance in user upload behavior from one BitTorrent community to another. This supports the managed swarming approach, which relies on active measurements of swarm behavior to adapt to unpredictable dynamics. However, while there is much variance among swarms, Bharambe et al. [14] verify the high efficiency that BitTorrent swarms maintain with respect to intra-swarm link utilization. This is in contrast to previous generations of peer-to-peer systems that predate BitTorrent's incentivization mechanisms, such as Gnutella, in which heterogeneity and lack of cooperation often led to poor performance [118].

BitTorrent swarms naturally organize peers over time into communities based on the bandwidth contributions of individual peers. A measurement study by Cuevas et al. [30] show that approximately 100 torrent publishers account for 75% of all downloads, based on traces of 55,000 torrents from popular content aggregators. During their downloads, Legout et al. [70] show that peers naturally cluster based on bandwidth, tending to unchoke other peers that have similar upload behavior. Fortunately, even selfish peers, which attempt to game BitTorrent by relying on other peers' optimistic unchokes, do little harm to the global performance of a BitTorrent system, based on measurements from Liogkas et al. [76].

Based on observations of BitTorrent's behavior, a large body of work has proposed optimizations to the protocol for improving its performance. Many of these enhancements are based on principled parameter selection, such as the optimal size of each data block [86], efficient peer unchoking algorithms [131], and reallocation of peers among multiple trackers that all manage torrents for the same piece of content [40]. Other work has focused on analyzing and improv-

ing BitTorrent's protocol, originally dubbed "tit-for-tat", for deciding how much upload bandwidth to allocate to each unchoked peer. Levin et al. [72] shows that "tit-for-tat" is a misnomer, and that the algorithm is more accurately described as a block auction, where peers bid their own bandwidth to a peer in an attempt to "win" reciprocated bandwidth. They suggest a new protocol called PropShare whereby peers award upload bandwidth in proportion to the bandwidth received from peers, rather than distributing bandwidth equally among all auction winners. BitMax [77] offers an alternate protocol, recommending that peers upload the maximum bandwidth possible to each unchoked peer, which they show increases performance when links are asymmetric. Peterson et al. [104] suggests taking a holistic approach to allocating bandwidth rather than solely basing each peer's bandwidth allocation decisions on local information.

Prior work has explored new avenues for improving performance, security, and fairness within swarming systems. Piatek et al. [98] augment the BitTorrent protocol to enable peers to share reputation information through one level of intermediary nodes. Cuevas et al. [31] analyze the benefits of keeping peer traffic within an ISP to reduce cross-ISP traffic, and Choffnes et al. [33] implement a solution by harvesting data from existing CDNs for locality information.

Finally, new mechanisms on top of unstructured mesh networks integrate social networks into peers' block exchange decisions. For instance, Tribler [102] and OneSwarm [60] use end users' social networks to favor or restrict data exchanges to connections between peers who have expressed trust in each other. The intention is that third parties are less likely to extract the behavior or interests of end users because system activity is compartmentalized based on explicit relationships of trust. Such mechanisms can be applied to multi-swarm settings

without affecting global performance goals or approaches to content distribution based on managed swarms.

5.3 Multi-Swarm Systems and Content Availability

This thesis focuses on achieving fast downloads in settings where servers with limited bandwidth seed multiple swarms. Existing work on multi-swarm environments largely focuses on utilizing resources to instead improve content availability. Content availability is a concern for peer-to-peer content distribution systems because end users and leave the system without warning, and sometimes take the last copy of an unpopular piece of content with them.

Past work has shown that content availability depends on several properties of swarms aside from content popularity, such as the distribution of download completion within a swarm [121]. The distribution of content among peers is a critical factor, and Kaune et al. [66] show that almost a quarter of peers in seederless swarms are able to complete their downloads from blocks exclusively from other downloading peers. Menasché et al. [88] formalize the related notion of *self-sufficient* swarms with a model that predicts when swarms are able to efficiently utilize their uplinks without aid from seeders.

Bundling is one approach that existing multi-swarm systems employ, where content distributors combine multiple pieces of content together into one archive served by a single swarm. When peers download one piece of content in a bundle, they are forced to download the other content as well. Consequently, peers serve blocks from all content in the bundle at the cost of downloading more content than they request. This improves the availability of unpopular

content and potentially provides seeding peers for the 82% of BitTorrent swarms with 10 or fewer peers that are often not able to sustain themselves [130]. Guo et al. [51] show that, empirically, peers are willing to stay in swarms for previously downloaded content, which maintains the content's availability. Menasché et al. [89] offer the first analytical model of content availability in swarming systems and present bundling as a simple and effective solution for increasing availability.

Further work has expanded on content bundling to make it more fine-grained and find the most efficient tradeoff between allowing peers to download only content in which they are interested and downloading other content to increase availability. Carlsson et al. [29] simulates a dynamic bundling technique where peers use spare download and upload bandwidth to opportunistically download individual blocks from other swarms and propagate them to downloaders. Levtov et al. [81] uses a combination of Markov decision processes and stochastic games to find a arrive at an efficient mix of downloading desired content and downloading undesired content for the global good. Finally, Yang et al. [129] propose a system that dynamically bundles content to improve both availability and performance. The system rewards peers that seed bundled content by increasing their download performance for other content in the bundle.

Availability is clearly a concern in the peer-to-peer community, and the long content popularity tail suggests that much content is extremely unpopular. A recent, large-scale measurement study of BitTorrent's content popularity by Dán et al. [41] shows that peer-to-peer content popularity does not follow a Zipf distribution. Instead, based on an analysis of proper statistical sampling over long

periods of time, the long tail of content popularity likely decreases exponentially. Consequently, the authors argue, problems related to content availability and caching are not nearly as serious and intractable as Zipf models of the content popularity tail suggest. Regardless, content popularity distributions empirically lead to challenges in availability, and research for addressing content availability for the tail has potential for improving the performance of managed swarming deployments with large content libraries.

5.4 Streaming Systems

With the rapidly growing popularity of video streaming services such as Netflix, Hulu, and YouTube, peer-to-peer content distribution research has shifted toward optimizing content streaming. New streaming systems generally organize peers into multicast trees, where content trickles from an origin server toward the leaves of the tree, or build upon BitTorrent's unstructured mesh networks to distribute streaming content in a more resilient but ad hoc manner.

5.4.1 Multicast

Multicast streaming systems organize peers into structured multicast trees, where content originates at a tree's root and propagates down the edges of the tree until it reaches the leaf peers. The seminal work by Deering proposed IP multicast to efficiently deliver content to multiple destinations [44]. Deployment difficulties with global IP multicast [38] led to application-level multicast

systems such as End System Multicast [32], Your Own Internet Distribution (YOID) [46], and others [132].

Several techniques have been proposed to increase efficiency of application-level multicast. Overcast [61] distributes content by constructing a bandwidth-optimized overlay tree among dedicated infrastructure nodes. Liu et al. [79] similarly optimize tree construction by calculating the appropriate tree depth and node degree. SplitStream [24] distributes content via a peer-to-peer overlay that disseminates content along branches of trees constructed on top of a peer-to-peer substrate. Bullet [64] and Bullet′ [63] also use a randomized overlay mesh to distribute data. ChunkySpread [123] is a hybrid approach to multicast trees that uses both structured and unstructured overlays to distribute content. Finally, Exapeer [53] proactively pushes replicas to regions of a multicast tree that do not yet have the content to ensure an even distribution of content.

5.4.2 Swarming

Multicast streaming systems inherently limit the bandwidth to each peer [85], and they require peer organization that is difficult to maintain in practice, especially in highly dynamic systems where peers near the root of a multicast tree may depart without warning. In response to multicast solutions, much work has focused instead on unstructured swarming systems similar to BitTorrent that support streaming. Chainsaw [101], for instance, is a peer-to-peer multicast system based on an unstructured overlay mesh in which peers explicitly request packets from neighbors. Many other systems in this design space discuss and address a tradeoff between swarming systems designed for bulk downloads

and swarming systems built for content streaming [133, 26, 122, 103, 90, 116]. That is, whereas peers in bulk download systems maximize download speeds by requesting rare blocks from neighbors, streaming applications prioritize blocks near the media's play position to minimize buffering delays. Zhao et al. [134] take this approach further by proposing an analytical framework for evaluating the tradeoff between rarest-first and in-order block selection. Fan et al. [45] show that this tradeoff is fundamental: they prove that rarest-first block selection policies and in-order block selection policies are in contention with each other. By implication, any swarming solution to streaming must make a tradeoff between these properties based on application requirements.

Other swarming systems are more explicit in their goal to reduce load on origin servers. The BitTorrent-Assisted Streaming System (BASS) [37] and Toast [35] add swarming-optimized peer-to-peer interactions to a client-server model. Toast, in particular, provides an implementation of a streaming swarming system that offloads 70–90% of traffic from servers to peers.

Rakesh et al. [65] introduce a stochastic fluid model for peer-to-peer streaming systems that captures the shortcomings of swarming mesh systems, such as churn. Their model, which assumes that all content originates at a single server, gives a closed-form result that states that a sufficiently large swarm whose traffic load exceeds a critical value will successfully distribute streaming content. Siddhartha et al. [9] propose a swarming protocol with small neighborhoods of topographically close peers for exchanging blocks, and uses heuristics to handle swarms of heterogeneous link capacities. Ration [126] dynamically adjusts streaming servers' upload bandwidths on a per-ISP basis to reduce inter-ISP

traffic and minimize server bandwidth requirements to sustain video streaming within the same ISP.

Finally, many streaming and multicast architectures use network coding to increase content delivery reliability [50, 7, 87, 15]. Network coding enables peers to encode content from multiple data blocks together in random linear combinations and then distribute these mathematically combined blocks. Consequently, peers downloading content no longer have to be concerned with which blocks they download, as any downloaded block is likely to contain new information that the peer can use to help decode the original content file.

5.5 Incentive Compatibility

In distributed systems that depend on end users to contribute resources, there is often contention between the global goals of the system and the selfish goals of individual end users. In managed swarms and peer-to-peer download systems, for instance, an individual user's goal is to download the content as fast as possible, possibly at the expense of other users' ability to download the content. Managed swarms address this contention by using the coordinator's global view of the system to compute how peers should behave for the global benefit of all peers, then using a variety of incentive mechanisms to ensure that peers follow their prescribed allocations. The coordinator uses a token-based virtual currency to incentivize peers to contribute bandwidth, and it uses its role as an authoritative tracker to punish peers that deviate from the protocol. In addition, managed swarms embrace the altruism of peers that choose to remain in swarms beyond their download completion time. Much work has examined in-

centives and currency systems in detail, especially in the context of peer-to-peer swarming.

Early model and analysis by Qiu and Srikant [108] of BitTorrent's incentive mechanism showed that the system converges to a Nash equilibrium where all peers upload at their capacity. However, more recent work, including BitTyrant [97], BitThief [80], and Sirivianos et al. [114], has demonstrated that average download times currently depend on significant altruism from high capacity peers that, when withheld, reduces performance for users.

In peer-assisted content distribution systems, altruism drives much of the research in incentive compatibility. Levin et al. [74] provide a high-level game theoretic and economic description of incentives that considers three mechanisms that affect peer behavior: money, punishment, and altruism. Managed swarms utilize these three mechanisms based on the coordinator's global view of the system and its ability to manage peers' membership and distribute tokens. Carlsson et al. [27] incentivize peers to seed content after they finish their downloads by prioritizing block requests from such peers in other swarms.

Dandelion [115] and BAR gossip [82] avoid relying on altruism to distribute data. They use a cryptographic fair exchange mechanism that requires a client to upload content to other clients in exchange for virtual credit, which can be redeemed for future service. Microcurrencies [19, 128, 100, 84] similarly rely on cryptographically protected tokens for fair resource exchange, and optionally provide additional features such as spender anonymity. iOwe [73] is a virtual currency backed by network resources; the currency is a promise to do future work to any other peer to implement a "pay-it-forward" scheme. BitStore [110] also uses a token-based currency backed by real money, which peers receive

when they offer resources for public use to store and upload content blocks. BitTorrent-like systems are susceptible to Sybil attacks [43], where a single user increases download speeds by joining the same swarm from several hosts.

Decentralized resource allocation in peer-to-peer systems requires incentives for participants to contribute resources. Ngan et al. [91] suggest cooperative audits to ensure that participants contribute storage commensurate with their usage. Samsara [34] considers storage allocation in a peer-to-peer storage system and introduces cryptographically signed storage claims to ensure that any user of remote storage devotes a like amount of storage locally. Both techniques center around audits of resources that are spatial in nature.

Karma [124] and SHARP [47] resource allocation can apply to renewable resources such as bandwidth. Karma employs a global credit bank, with which clients maintain accounts. The value of a client's account increases when it contributes and decreases when it consumes. A client can only consume resources if its account contains sufficient credit. SHARP operates at the granularity of autonomous systems or sites. To join the system a SHARP site must negotiate resource contracts with one or more existing group members. These contracts, in effect, specify the system's expectations of the site and the site's promise of available resources to the system. Accountable claims make it possible to monitor each participant's compliance with its contracts, simplifying audits and making collusion more difficult in SHARP relative to other decentralized peer-to-peer systems.

Carlsson et al. [28] extend peer incentives to the streaming media domain. One-Hop Reputations [98] and Contracts [99] use propagation trees of depth one to increase peer accountability in bulk download and live streaming sys-

tems, respectively. The protocols introduce peer incentivization strategies that outperform BitTorrent's bilateral tit-for-tat approach.

Freedman et al. [48] 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.

CHAPTER 6

CONCLUSIONS

6.1 Summary

This thesis introduces a new approach to content distribution based on managed swarms. Managed swarms, where the behavior of the nodes in the network are guided by a virtual entity to achieve a globally desirable outcome, enable content distributoirs to use network resources efficiently. Deployments of managed swarms can make effective use of resources available at server-class hosts as well as bandwidth contributed by end users. The managed swarming approach described in this thesis employs a logically centralized coordinator that guides hosts, whether or not they are owned and operated by the content distributor, to achieve global performance objectives.

A coordinator forms the core of a managed swarming deployment. The coordinator takes active measurements of host behavior, which it uses to model the behavior of swarms. With its global view of swarm dynamics, the coordinator is able to compute an efficient allocation of network resources to maximize system-wide performance according to an operator-defined objective. Then, because end users often have their own motivations and incentives to deviate from globally beneficial behavior, the coordinator monitors individual peers to ensure that they follow their prescribed allocations.

In this thesis, we explore in depth the global performance goal of maximizing the aggregate bandwidth of all downloaders. This in turn maximizes both the average download rate across peers in the system. To this end, we have de-

fined two problems that formalize the goal of maximizing bandwidth that apply to different deployment scenarios. The single-seeder multi-swarm content distribution problem is appropriate for small content owners whose libraries fit in a single logically centralized server. A more universal variant, called the general multi-swarm content distribution problem, relaxes that constraint, allowing deployments with content libraries that can be split across any number of origin servers and cache servers.

To address the content distribution problems, we have developed two algorithms for allocating each host's upload bandwidth among swarms competing for its bandwidth. Antfarm addresses the single-seeder variant, and it computes an optimal allocation of bandwidth from the centralized origin server to ensure that a small content distributor receives the maximum possible performance possible from limited resources. To address the general variant of the problem, we have developed V-Formation, in which the coordinator measures each host's individual impact on the swarms to which the host belongs. Then, based on each swarm's response to bandwidth from the host, the coordinator allocates the host's bandwidth accordingly so that each host uploads content to the swarm that receives the most benefit from its bandwidth.

We have implemented both algorithms and a distributed, scalable coordinator to build an efficient managed swarming system. We have demonstrated that our deployment can scale to large deployments and achieves qualitatively higher performance than existing content distribution systems and approaches, including traditional client-server systems, popular peer-to-peer protocols, and numerous heuristics that hosts might use to allocate bandwidth in deployments lacking the infrastructure to make principled bandwidth allocation decisions.

6.2 Future Work

The general multi-swarm content distribution problem, addressed in this thesis, formalizes the goal of maximizing aggregate bandwidth given the swarm memberships of all peers. The algorithms that we have developed to address this problem ensure that hosts' bandwidth is used efficiently to distribute the content that they possess. However, the coordinator has no control over which content hosts download and consequently are able to share with other peers. Granting the coordinator the flexibility to change swarm memberships would give the coordinator the power to place content within the network and to allocate resources more freely among swarms to increase system-wide performance.

Two challenges arise when the coordinator is tasked with dynamically assigning peers to swarms. First, the extra degree of freedom makes the the coordinator's task of optimizing bandwidth among swarms more computationally difficult. Second, scenarios in which the coordinator modifies the swarm memberships of end users require new incentives for end users to contribute their upload bandwidth for distributing content in which the users might not be interested. We have begun preliminary work to address these challenges, which we discuss in turn.

6.2.1 Content Placement Within Distribution Infrastructure

Positioning content strategically among cache servers is an important and well-known method for improving the perceived efficiency of content distribution networks. End users benefit when content in which they are interested is located

nearby and is sufficiently replicated to ensure availability. Managed swarming systems are in a unique position to address this problem because all cache servers are under the control of a logically centralized coordinator that can dictate how they allocate their resources. Thus, a managed swarming deployment has the opportunity to optimize the placement of content, rather than rely on the content owner to manually set and maintain swarm memberships for each cache server.

The managed swarming framework provided in this thesis provides the necessary tools to address the placement problem. A coordinator can determine the placement of content by directing cache servers to join and leave swarms on demand. We formalize the optimization problem of choosing which swarms each server joins as the *content placement multi-swarm content distribution problem*. Namely, given a set of swarms S and a set of peers P, this variant of the problem designates a subset of peers as cache servers under the content owner's control $C \subseteq P$, whose swarm memberships are chosen by the coordinator. The complement set of peers $E = P \setminus C$ is the set of end users, who choose their own swarm memberships, as in the general multi-swarm content distribution problem. Then, given a set of end-user memberships $M_E \subseteq E \times S$, the content placement multi-swarm content distribution problem is to determine to which swarms each cache server should belong $M_C \subseteq C \times S$ in order to maximize the aggregate bandwidth of end users $a = \sum_{p \in E} D_p$, where D_p is the download bandwidth of peer p. For a particular set of cache-swarm memberships M_C, the aggregate bandwidth a is defined based on the optimal allocation of each peer's bandwidth to the swarms to which it belongs, identically to the general multi-swarm content distribution problem with $M = M_E \cup M_C$. This problem is difficult because it treats the general multi-swarm content distribution problem

as a subproblem by wrapping it in an additional maximization problem to find the most efficient set of swarm memberships for each cache server.

To address the content placement multi-swarm content distribution problem, one approach is to generalize the technique of probing swarms with a small, constant number of block uploads to measure the benefit of caching blocks from particular piece of content. Specifically, just as a peer can upload a block and track its propagation to estimate the value of a block upload to a swarm, a cache server can cache a block for a period of time and measure the number of times it uploads that block and weight the result by the average propagation of those block uploads to estimate the value of caching a piece of content.

A solution to this problem would enable content distributors to efficiently use their available infrastructure without the need to manually place content in the network. Managed swarms offer the machinery to evaluate a given configuration of content, and they would therefore prove useful in estimating the value of shifting content among cache servers to improve performance.

6.2.2 Increasing Control Over End Users' Resources

Enabling the coordinator to control cache servers' swarm memberships enables it to position content in the network. By extension, the coordinator could further optimize network resources if it had similar control over swarm memberships of end users. Unlike cache servers, end users' resources are not under the control of the content distributor; end users have their own motivations and behaviors. Specifically, end users join swarms for content that they want to download

or distribute, and contributing resources back to their swarms is a secondary concern. In managed swarms, the coordinator incentivizes peers to contribute bandwidth to their swarms by offering them tokens that they can use to obtain additional data blocks from peers. Finding an incentivization model for peers to contribute bandwidth to other swarms, for content in which they are not necessarily interested, would be beneficial for steering end users' bandwidth to swarms that most need it.

We have considered two possible approaches for incentivizing peers to join swarms for content that they do not want. The first approach is to award peers tokens for uploading blocks from swarms to which they do not belong. In such a scheme, the coordinator would calculate an *exchange rate* for each swarm by weighing the global benefit of a block uploaded in one swarm relative to other swarms. The exchange rate would determine the number of tokens that a peer would receive for each block that it uploads to that swarm. If the exchange rate for a particular swarm is high enough, it would incentivize peers that do not belong to the swarm to join it, download data blocks for its content, and upload those blocks in order to earn tokens that the peer can use to download content in which it is interested.

A second approach for steering end users swarm memberships is to extend content bundling to incentivize peers to download and serve particular content. Bundling is a technique whereby multiple pieces of content are combined into a single swarm, often a popular piece of content with content that would benefit from additional bandwidth. Then, the many peers downloading the popular content are forced to download the less popular content as well, improving bandwidth and availability for peers interested in the long tail. Man-

aged swarms are in a unique position to improve content bundling. Current bundling work depends on ad hoc heuristics for deciding which content to bundle together. The coordinator in managed swarming systems has the data and algorithms to accurately predict each swarm's expected response to additional bandwidth. It could use this information to optimize content bundling and better serving the content popularity tail.

6.3 Impact

Managed swarms have had an impact on the content distribution landscape. We have commercialized our implementation of managed swarms, first based on Antfarm, then based on V-Formation, to better match an open content distribution service based on user-contributed content. Our content delivery service, called FlixQ [3], enables users to share videos and other bulk content publicly or with specific groups of users. FlixQ has been live for more than two years as of this writing, and has several hundred users sharing thousands of videos.

The FlixQ service enables students, faculty, and staff of particular colleges and universities to share content within the scope of their schools. Cornell University's computer science uses this feature to make its departmental colloquium talks available to the Cornell community. Several of the department's servers proactively join swarms for new colloquia and cache them for availability.

Managed swarming technology has spurred new research in content distribution systems based on realistic models of swarm behavior for predicting their response to future bandwidth contributions. For instance, Capotă et al. [21]

have expanded on the Antfarm work to address additional global performance goals and provide support for streaming video. Similarly, Niu et al. [90] apply machine learning techniques to managed swarms to predict future changes in swarm behavior and adjust bandwidth allocations accordingly. Kash et al. [69] have examined systems that use virtual currencies, including our deployments of managed swarms, to optimize the use of currencies and incentivize efficient peer behavior.

Managed swarms offer an intuitive model for describing the complex behavior of swarms. This thesis explores how managed swarms enable content distributors to achieve high performance, and focuses on maximizing system-wide link utilization for fast bulk downloads. There are many unexplored avenues for improving performance and reducing the cost of distribution. We believe that managed swarming approach provides a general framework for deploying efficient, scalable systems that are well suited to a wide range of problems that future research will address.

BIBLIOGRAPHY

[1] BitTornado. http://www.bittornado.com/download.html.

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

[3] Flixq.com – Videosharing for the Masses. http://flixq.com.

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

[5] PPTV. http://www.pptv.com.

[6] D. Applegate, A. Archer, and V. Gopalakrishnan. Optimal Content Placement for a Large-scale VoD System. *Conference on Emerging Networking Experiments and Technologies*, Philadelphia, PA, November 2010.

[7] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4), July 2000.

[8] S. Annapureddy, M. J. Freedman, and D. Mazières. Shark: Scaling File Servers via Cooperative Caching. *Symposium on Networked System Design and Implementation*, Boston, MA, May 2005.

[9] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Is High-quality VoD Feasible Using P2P Swarming? *International World Wide Web Conference*, Banff, Canada, May 2007.

[10] T. Arango and D. Carr. Netflix's Move onto the Web Stirs Rivalries. *The New York Times*, November 2010.

[11] H. A. Alzboubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. d. Merwe. Anycast CDNs Revisited. *International World Wide Web Conference*, Beijing, China, April 2008.

[12] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. *IEEE International Conference on Distributed Computing Systems*, Lisboa, Portugal, July 2006.

[13] 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.

[14] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Networks Performance Mechanisms. *IEEE International Conference on Computer Communications*, Barcelona, Spain, April 2006.

[15] 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.

[16] 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.

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

[18] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz. An Anycast Service for Large Content Distribution. *International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[19] 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.

[20] L. Cherkasova and J. Lee. Fastreplica: Efficient Large File Distribution within Content Delivery Networks. *USENIX Symposium on Internet Technologies and Systems*, Seattle, WA, March 2003.

[21] M. Capotă, N. Andrade, T. Vinkó, F. Santos, J. Pouwelse, and D. Epema. Inter-swarm Resource Allocation in BitTorrent Communities. *IEEE International Conference on Peer-to-Peer Computing*, Kyoto, Japan, August 2011.

[22] M. Calore. Zudeo Announces Deal with BBC. *Wired Blog Network*, December 19 2006.

[23] M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. *Symposium on Operating System Design and Implementation*, Boston, MA, December 2002.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh. Splitstream: High-bandwidth Multicast in Cooperative Environments. *Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[25] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou. Utility Maximization in Peer-to-Peer Systems. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Annapolis, MD, June 2008.

[26] N. Carlsson and D. L. Eager. Peer-assisted On-demand Streaming of Stored Media Using BitTorrent-like Protocols. *IFIP/TC6 Networking*, Atlanta, GA, May 2007.

[27] N. Carlsson and D. L. Eager. Modeling Priority-based Incentive Policies for Peer-assisted Content Delivery Systems. *IFIP/TC6 Networking*, Singapore, May 2008.

[28] N. Carlsson, D. L. Eager, and A. Mahanti. Peer-assisted On-demand Video Streaming with Selfish Peers. *IFIP/TC6 Networking*, Aachen, Germany, May 2009.

[29] N. Carlsson, D. L. Eager, and A. Mahanti. Using Torrent Inflation to Efficiently Serve the Long Tail in Peer-assisted Content Delivery Systems. *IFIP/TC6 Networking*, Chennai, India, May 2010.

[30] R. Cuevas, M. Kryczka, A. Cuevas, S. Kaune, C. Guerrero, and R. Rejaie. Is Content Publishing in BitTorrent Altruistic or Profit-driven? *Conference on Emerging Networking Experiments and Technologies*, Philadelphia, PA, November 2010.

[31] R. Cuevas, N. Laoutaris, X. Yang, G. Siganos, and P. Rodriguez. Deep Diving Into BitTorrent Locality. *IEEE International Conference on Computer Communications*, Shanghai, China, April 2011.

[32] Y.-h. Chu, S. G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8), October 2002.

[33] 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.

[34] L. P. Cox and B. D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. *Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[35] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai. Improving VoD Server Efficiency with BitTorrent. *ACM Multimedia*, Augsburg, Germany, September 2007.

[36] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2009–2014. June 2010.

[37] C. Dana, D. Li, D. Harrison, and C.-N. Chuah. Bass: BitTorrent Assisted Streaming System for Video-on-demand. *IEEE Workshop on Multimedia Signal Processing*, 2005.

[38] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the ip Multicast Service and Architecture. *IEEE Network*, 14(1), January 2000.

[39] F. Dabek, R. Cox, M. F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. *SIGCOMM Conference*, Portland, OR, August 2004.

[40] G. Dán and N. Carlsson. Dynamic Swarm Management for Improved BitTorrent Performance. *International Workshop on Peer-to-Peer Systems*, Boston, MA, April 2009.

[41] G. Dán and N. Carlsson. Power-law Revisited: A Large Scale Measurement Study of P2P Content Popularity. *International Workshop on Peer-to-Peer Systems*, San Jose, CA, April 2010.

[42] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally Distributed Content Delivery. *IEEE Internet Computing*, October 2002.

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

[44] S. E. Deering. Multicast Routing in Internetworks and Extended LANs. *SIGCOMM Conference*, Stanford, CA, August 1988.

[45] B. Fan, D. G. Andersen, M. Kaminsky, and K. Papagiannaki. Balancing Throughput, Robustness, and In-order Delivery in P2P VoD. *Conference on Emerging Networking Experiments and Technologies*, Philadelphia, PA, November 2010.

[46] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, and B. Lindell. Yoid: Your Own Internet Distribution. `http://www.isi.edu/div7/yoid`, March 2001.

[47] Y. Fu, J. S. Chase, B. N. Chun, S. Schwab, and A. Vahdat. Sharp: An Architecture for Secure Resource Peering. *Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[48] 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.

[49] 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.

[50] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. *IEEE International Conference on Computer Communications*, Miami, FL, March 2005.

[51] L. Guo, S. Chen, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. *Internet Measurement Conference*, Berkeley, CA, October 2005.

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

[53] A. Hayakawa, M. Asahara, K. Kono, and T. Kojima. Efficient Update Propagation By Speculating Replica Locations on Peer-to-Peer Networks. *IEEE International Conference on Parallel and Distributed Systems*, Shenzhen, China, December 2009.

[54] C. Huang, J. Li, A. Wang, and K. W. Ross. Understanding Hybrid CDN-P2P: Why Limelight Needs Its Own Red Swoosh. *Network and Operating System Support for Digital Audio and Video*, Braunschweig, Germany, May 2008.

[55] C. Huang, J. Li, and K. W. Ross. Can Internet Video-on-demand Be Profitable? *SIGCOMM Conference*, Kyoto, Japan, August 2007.

[56] D. Hales and S. Patarin. How to Cheat BitTorrent and Why Nobody Does. *University of Bologna Technical Report UBLCS-2005-12*, 2005.

[57] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A Measurement Study of a Large-scale P2P IPTV System. *IEEE Transactions on Multimedia*, 9(8), December 2007.

[58] Y. Huang, T. Z. J. Fu, D.-M. Chiu, J. C. S. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P VoD System. *SIGCOMM Conference*, Seattle, WA, August 2008.

[59] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. *Passive and Active Network Measurement*, Antibes Juan-les-Pins, France, April 2004.

[60] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving P2P Data Sharing with Oneswarm. *SIGCOMM Conference*, New Delhi, India, August 2010.

[61] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. *Symposium on Operating System Design and Implementation*, San Diego, CA, October 2000.

[62] 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.

[63] D. Kostic, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining High-bandwidth Under Dynamic Network Conditions. *USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[64] D. Kostic, A. Rodriguez, J. R. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. *Symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003.

[65] R. Kumar, Y. Liu, and K. W. Ross. Stochastic Fluid Theory for P2P Streaming Systems. *IEEE International Conference on Computer Communications*, Anchorage, AK, May 2007.

[66] S. Kaune, R. C. Rumin, G. Tyson, A. Mauthe, C. Guerrero, and R. Steinmetz. Unraveling BitTorrent's File Unavailability: Measurements, Analysis and Solution Exploration. *IEEE International Conference on Peer-to-Peer Computing*, Delft, Netherlands, August 2010.

[67] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet Service Providers Fear Peer-assisted Content Distribution? *Internet Measurement Conference*, Berkeley, CA, October 2005.

[68] 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.

[69] I. A. Kash, E. J. Friedman, and J. Y. Halpern. Optimizing Scrip Systems: Efficiency, Crashes, Hoarders, and Altruists. *ACM Conference on Electronic Commerce*, San Diego, CA, June 2007.

[70] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and Sharing Incentives in BitTorrent Systems. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, San Diego, CA, June 2007.

[71] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms Are Enough. *Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[72] 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.

[73] D. Levin, A. Schulman, K. LaCurts, N. Spring, and B. Bhattacharjee. Making Currency Inexpensive with iOwe. *Workshop on the Economics of Networks, Systems, and Computation*, San Jose, CA, June 2011.

[74] D. Levin, N. Spring, and B. Bhattacharjee. Systems-compatible Incentives. *International Conference on Game Theory for Networks*, Istanbul, Turkey, May 2000.

[75] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. *Symposium on Networked System Design and Implementation*, Cambridge, MA, April 2007.

[76] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploring the Robustness of BitTorrent Peer-to-Peer Content Distribution Systems. *Concurrency and Computation: Practice and Experience*, 20(2), February 2008.

[77] N. Laoutaris, D. Carra, and P. Michiardi. Uplink Allocation Beyond Choke/unchoke: Or How to Divide and Conquer Best. *Conference on Emerging Networking Experiments and Technologies*, Madrid, Spain, December 2008.

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

[79] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang. Performance Bounds for Peer-assisted Live Streaming. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Annapolis, MD, June 2008.

[80] 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.

[81] N. Lev-tov, N. Carlsson, Z. Li, C. Williamson, and S. Zhang. Dynamic File-selection Policies for Bundling in BitTorrent-like Systems. *IEEE International Workshop on Quality of Service*, Beijing, China, June 2010.

[82] 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.

[83] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. Modeling and Analysis of Bandwidth-inhomogeneous Swarms in BitTorrent. *IEEE International Conference on Peer-to-Peer Computing*, Seattle, WA, September 2009.

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

[85] N. Magharei and A. H. Rasti. Prime: Peer-to-Peer Receiver-driven Mesh-based Streaming. *IEEE International Conference on Computer Communications*, Anchorage, AK, May 2007.

[86] P. Marciniak, N. Liogkas, A. Legout, and a. E. Kohler. Small Is Not Always Beautiful. *International Workshop on Peer-to-Peer Systems*, Tampa Bay, FL, February 2008.

[87] P. Maymounkov and D. Mazières. Rateless Codes and Big Downloads. *International Workshop on Peer-to-Peer Systems*, Springer Lecture Notes in Computer Science 2735, Berkeley, CA, February 2003.

[88] D. S. Menasché, A. A. de Aragão Rocha, E. de Souza e Silva, R. M. M. Leão, D. F. Towsley, and A. Venkataramani. Estimating Self-sustainability in Peer-to-Peer Swarming Systems. *Performance Evaluation*, 67(11), November 2010.

[89] D. S. Menasché, A. A. A. Rocha, B. Li, D. Towsley, and A. Venkataramani. Content Availability and Bundling in Swarming Systems. *Conference on Emerging Networking Experiments and Technologies*, Rome, Italy, December 2009.

[90] D. Niu, B. Li, and S. Zhao. Self-diagnostic Peer-assisted Video Streaming Through a Learning Framework. *ACM Multimedia*, Florence, Italy, October 2010.

[91] 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.

[92] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions. *Mathematical Programming*, 14(1), December 1978.

[93] S. I. B. Networks. Global Internet Phenomena Report. October 2011.

[94] T. S. E. Ng and H. Zhang. Towards Global Network Positioning. *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, November 2001.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] 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.

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

[101] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees From Overlay Multicast. *International Workshop on Peer-to-Peer Systems*, Springer Lecture Notes in Computer Science 3640, Ithaca, NY, February 2005.

[102] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker1, J. Yang, A. Iosup, D.H.J. Epema, M. Reinders, M. van Steen, and H.J. Sips. Tribler: A Social-based Peer-to-Peer System. *International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[103] K. N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of BitTorrent-like Protocols for On-demand Stored Media Streaming. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Annapolis, MD, June 2008.

[104] R. S. Peterson and E. G. Sirer. Going Beyond Tit-for-tat: Designing Peer-to-Peer Protocols for the Common Good. *Workshop on Future Directions in Distributed Computing*, Bertinoro, Italy, June 2007.

[105] 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.

[106] R. S. Peterson, V. Ramasubramanian, and E. G. Sirer. A Practical Approach to Peer-to-Peer Publish-subscribe. *;login:* 31(4), August 2006.

[107] R. S. Peterson, B. Wong, and E. G. Sirer. A Content Propagation Metric for Efficient Content Distribution. *SIGCOMM Conference*, Toronto, Canada, August 2011.

[108] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks. *SIGCOMM Conference*, Portland, OR, August 2004.

[109] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the Placement of Web Server Replicas. *IEEE International Conference on Computer Communications*, Anchorage, AK, April 2001.

[110] A. Ramachandran, A. D. Sarma, and N. Feamster. Bitstore: An Incentive-compatible Solution for Blocked Downloads in BitTorrent. *Workshop on the Economics of Networks, Systems, and Computation*, San Diego, CA, June 2007.

[111] P. Rodriguez and E. W. Biersack. Dynamic Parallel-access to Replicated Content in the Internet. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[112] V. Ramasubramanian, R. S. Peterson, and E. G. Sirer. Corona: A High Performance Publish-subscribe System for the World Wide Web. *Symposium on Networked System Design and Implementation*, San Jose, CA, May 2006.

[113] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a Dht (awarded Best Paper!). *USENIX Annual Technical Conference*, Boston, MA, June 2004.

[114] 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.

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

[116] P. Shah and J.-f. Pâris. Peer-to-Peer Multimedia Streaming Using BitTorrent. *IEEE International Performance Computing and Communications Conference*, New Orleans, LA, April 2007.

[117] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A Cooperative Bulk Data Transfer Protocol. *IEEE International Conference on Computer Communications*, Hong Kong, March 2004.

[118] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. *Multimedia Computing and Networking*, San Jose, CA, January 2002.

[119] 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.

[120] Y. J. Song, V. Ramasubramanian, and E. G. Sirer. Optimal Resource Utilization in Content Distribution Networks. *Cornell University Technical Report TR-2005-2004*, November 2005.

[121] Y. Tian, D. Wu, and K.-W. Ng. Modeling, Analysis and Improvement for BitTorrent-like File Sharing Networks. *IEEE International Conference on Computer Communications*, Barcelona, Spain, April 2006.

[122] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for Supporting Streaming Applications. *IEEE Global Internet Symposium*, Barcelona, Spain, April 2006.

[123] V. Venkataraman, P. Francis, and J. Calandrino. Chunkyspread: Multi-tree Unstructured Peer-to-Peer. *International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, February 2006.

[124] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A Secure Economic Framework for P2P Resource Sharing. *Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, May 2003.

[125] 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.

[126] C. Wu, B. Li, and S. Zhao. Multi-channel Live P2P Streaming: Refocusing on Servers. *IEEE International Conference on Computer Communications*, Phoenix, AZ, April 2008.

[127] 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.

[128] P. Wayner. *Digital Cash: Commerce on the Net*. Morgan Kaufmann, April 1996.

[129] Y. Yang, A. L. H. Chow, and L. Golubchik. Multi-torrent: A Performance Study. *Modeling, Analysis, and Simulation on Computer and Telecommunication Systems*, Baltimore, MD, September 2008.

[130] C. Zhang, P. Dunghel, D. Wu, and K. W. Ross. Unraveling the BitTorrent Ecosystem. *IEEE Transactions on Parallel and Distributed Systems*, 22(7), July 2011.

[131] H. Zhang, Z. Shao, M. Chen, and K. Ramchandran. Optimal Neighbor Selection in BitTorrent-like Peer-to-Peer Networks. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, San Jose, CA, June 2011.

[132] Y. Zhu, W. Shu, and M.-Y. Wu. Approaches to Establishing Multicast Overlays. *IEEE International Conference on Services Computing*, Orlando, FL, July 2005.

[133] Y. Zhou. A Simple Model for Analyzing P2P Streaming Protocols. *IEEE International Conference on Network Protocols*, Beijing, China, October 2007.

[134] B. Q. Zhao, J. C. S. Lui, and D.-M. Chiu. Exploring the Optimal Chunk Selection Policy for Data-driven P2P Streaming Systems. *IEEE International Conference on Peer-to-Peer Computing*, Seattle, WA, September 2009.