Antfarm: Efficient Content Distribution with Managed Swarms

Ryan S. Peterson Emin Gün Sirer
{ryanp,egs}@cs.cornell.edu
Department of Computer Science, Cornell University
United Networks, L.L.C.

This paper describes Antfarm, a content distribution system based on managed swarms. A managed swarm couples peer-to-peer data exchange with a coordinator that directs bandwidth allocation at each peer. Antfarm achieves high throughput by viewing content distribution as a global optimization problem, where the goal is to minimize download latencies for participants subject to bandwidth constraints and swarm dynamics. The system is based on a wire protocol that enables the Antfarm coordinator to gather information on swarm dynamics, detect misbehaving hosts, and direct the peers' allotment of upload bandwidth among multiple swarms. Antfarm's coordinator grants autonomy and local optimization opportunities to participating nodes while guiding the swarms toward an efficient allocation of resources. Extensive simulations and a PlanetLab deployment show that the system can significantly outperform centralized distribution services as well as swarming systems such as BitTorrent.

1 INTRODUCTION

Content distribution has emerged as a critical application as demand for high fidelity multimedia content has soared. Large multimedia files require effective content distribution services. Past solutions to the content distribution problem can be categorized into two approaches, namely client-server systems and peer-to-peer swarming systems, whose fundamental limitations render them inadequate for many deployment environments.

In the client-server approach to content distribution, the content owner operates a set of servers that provide the content to every client without tapping into any client-side resources. The presence of a central authority simplifies the design of client-server systems, exemplified by YouTube and Akamai: provisioning the network simply requires purchasing sufficient bandwidth for the desired quality of service and the targeted number of clients; 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 chief drawback to the client-server approach is its cost and feasibility: the distributor must bear the entire bandwidth cost of distributing the content, and operating a high-bandwidth data center for a large client population can be prohibitively expensive [11].

Peer-to-peer swarms offer an emerging alternative, where clients interested in downloading a file provide content to other clients interested in the same file. Swarming protocols transfer part of the bandwidth cost from centralized servers to the participants and their ISPs by taking advantage of the additional upload capacity offered by downloading peers. This redistribution of costs reduces the bandwidth burden on the servers, helps improve download times for clients, and reduces ingress bandwidth demand for ISPs. Swarming protocols proposed to date, including BitTorrent [1], Avalanche [24], and Dandelion [52], have been designed to resist technical and legal attacks by avoiding management and centralization. This design choice has led to protocols that lack coordination among peers, rely solely on directly-obtained measurements to avoid trusting information relayed by peers, and depend on randomization to thwart adversaries. The highly decentralized nature of existing unmanaged swarming systems leads to a performance penalty for legitimate content distributors.

To understand why unmanaged swarming architectures fail to make efficient use of bandwidth in multi-swarm environments, imagine a content provider with two movies to distribute to two sets of users using a set of seeders[1] over which they have full control and at which both movies are replicated. Depending on the size of the swarms and the nature of the peers that make up each

[1] In this paper, seeders are trusted servers managed by the coordinator that distribute data blocks to peers. This is in contrast with BitTorrent terminology, where seeders are altruistic peers that have finished downloading a file and provide content without further downloads themselves.

swarm, the two swarms may have vastly different internal dynamics. Seeders and peers with blocks belonging to multiple swarms face a difficult choice: which swarm should they reward with their upload bandwidth? Simple heuristics, such as round-robin, are unlikely to work well because they do not take swarm dynamics into account. The default BitTorrent behavior, which awards download slots to the peers with a proven track of fast downloads, works well within a single torrent, but can lead to wholesale starvation in a multi-torrent setting.

The fundamental problem is one of global optimization: the seeders should award their bandwidth such that download times across all swarms are minimized. Current swarming protocols lack the mechanisms to compute and operate at this point. Consequently, administrators that run torrent sites manually prune old torrents and reallocate bandwidth to more popular downloads by hand. This approach is not guaranteed to achieve a good allocation of bandwidth, leads to the "heavy-tail" problem where old, unpopular torrents are difficult to find, and does not scale.

This paper describes an efficient content distribution system, called Antfarm, based on *managed swarms*. The goal of Antfarm is to distribute a large set of files to a potentially very large set of clients. Managed swarms introduce a hybrid approach to swarming systems in that they permit a *coordinator*, typically managed by the content distributor, to control the behavior of the swarms.

Antfarm is designed to maximize the system-wide benefit of the critical resource, seeder bandwidth. Each Antfarm peer provides resources to other participants, receives unforgeable *tokens* in return, and receives credit for its cooperation by presenting these tokens to the central coordinator. The Antfarm token protocol forces each participant to divulge its upload contributions to the swarm coordinator, which enables the coordinator to determine swarm dynamics and allocate bandwidth to competing swarms. This enables the coordinator to exert control while enabling peers to use microoptimizations, such as optimistic unchoking for peer discovery, tit-for-tat for peer selection, and rarest first, to improve the efficiency of swarming downloads. Overall, the Antfarm transport protocol makes the system resistant to attacks through unforgeable tokens, reveals a coarse-grain view of the network to the central coordinator, and enables the coordinator to adopt and enforce a chosen bandwidth allocation strategy.

The key contribution of this paper is the design of an efficient and scalable coordinator for multiple, concurrent swarms. Given the internal dynamics of a set of swarms, we show how to optimize bandwidth among the swarms such that average download latencies are minimized across all peers. If desired, the algorithm can guarantee a minimum service level to certain swarms, avoid starvation, and enforce prioritization among swarms. Minimizing the average download latency in turn enables a content distributor to achieve the best possible service from the available bandwidth.

This paper makes two additional contributions for achieving high throughput in a practical multiple-swarm download service. First, the paper presents a wire-level protocol for accurately measuring the internal dynamics of individual swarms by making peer contributions evident to the coordinator, enabling the coordinator to optimally allocate bandwidth among the competing swarms. Second, a full implementation of the protocol, accompanied by extensive simulations and a deployment on PlanetLab, quantifies the performance of Antfarm against a client-server system and BitTorrent. In our experiments, Antfarm achieves aggregate bandwidths up to a factor of five higher than BitTorrent, and the protocol scales well with increasing peers and swarms.

The rest of this paper is structured as follows. The next section describes the Antfarm system and the central optimization that underlies the approach. Section 3 outlines the protocol that Antfarm uses for data distribution. Section 4 shows that the system achieves high performance. Section 5 describes related work and highlights Antfarm's differences, and Section 6 summarizes our contributions.

2 APPROACH

Antfarm is based on a hybrid peer-to-peer architecture that utilizes resources provided by peers according to an optimal strategy for managing multiple swarms computed by a coordinator. Each coordinator can manage multiple swarms, a single peer may participate in swarms managed by multiple coordinators, and coordinators may be physically replicated to scale with the number of peers and swarms. For simplicity, we assume a single coordinator in the following discussion and address the issue of scale in Section 3.

The coordinator's central task is to achieve the shortest possible download times across multiple swarms. Finding the right allotment of bandwidth among swarms is best viewed as a constrained optimization problem. The primary constraint is the available bandwidth at the seeders. The primary input to this optimization problem is the inherent *response curve* of each swarm. The response curve represents the swarm bandwidth as a function of allocated seeder bandwidth. It depends on the number of peers in the swarm, number of seeders, spare bandwidth on upload and download links of swarm participants, and the distribution of unique blocks. Peers' local decisions also influence their swarms' response curves,

as peers can advertise a lower upload bandwidth capacity than they are capable of providing. However, the Antfarm wire protocol, discussed in Section 3, encourages peers to cooperate within their swarms, granting the coordinator more available bandwidth to optimally allocate among all swarms in the system.

Response curves embody the critical properties of each swarm and have a characteristic shape—a fact we exploit in this work. Figure 1 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 small, 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 committed by the seeder. Once the peers in a homogeneous 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.

A coordinator relies on two key properties of response curves to maximize the achieved aggregate swarm bandwidth while respecting the seeder bandwidth constraint. First, response curves are monotonic: a swarm's aggregate bandwidth will never decrease as a result of increasing the seeder bandwidth to the swarm. 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 that Antfarm exploits. In heterogeneous swarms, where peer uplinks and downlinks are nonuniform, 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 to heterogeneity, 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. Section 3 describes how Antfarm maintains an accurate view of the system and adjusts its behavior in the presence of dynamic membership.

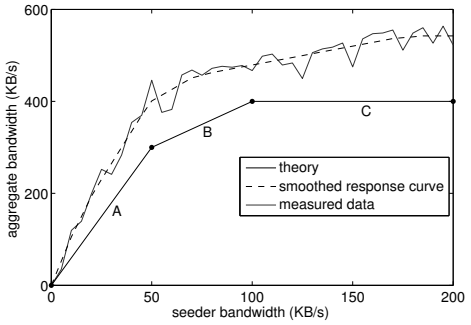

Figure 1: **Response curves of a theoretical homogeneous swarm and a measured heterogeneous swarm on PlanetLab.** Aggregate bandwidth increases rapidly as seeder bandwidth increases (A) until peer uplink capacity is exhausted (B) and reaches its maximum when downlinks are saturated (C).

The monotonicity and concavity of swarm response curves form the foundation of Antfarm's multiple-swarm optimization. Intuitively, when a seeder is supporting a swarm that has a large number of saturated peers, such as in regions B or C in Figure 1, it should reduce its bandwidth to that swarm and divert it to a swarm whose peers can readily share additional bandwidth. More generally, given a response curve for each swarm Antfarm is currently distributing, the coordinator "climbs" each of the curves, always preferring the steepest curve, until it has allocated all seeder bandwidth. The resulting *point of operation* on each curve represents the amount of bandwidth the seeders plan 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 [40]: decreasing the seeder bandwidth to one swarm in favor of another will not improve the overall performance of the system. Antfarm's allocation of seeder

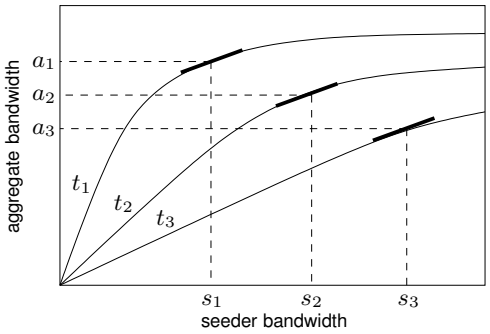

Figure 2: **Optimal bandwidth allocation for three concurrent swarms.** The Antfarm coordinator awards seeder bandwidth by hill-climbing the steepest response curves first until all available bandwidth has been allocated.

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 seeders have excess bandwidth to award, yet the derivatives of multiple response curves are identical, indicating that multiple swarms offer the same global benefit (Figure 2). In such cases of equivalent global benefit, Antfarm uses a tie-breaker algorithm to maximize perceived improvement by peers. Suppose that two swarms t_1 and t_2 have response curves with equivalent slopes at seeder bandwidths s_1 and s_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 t_1, which already enjoy a larger aggregate throughput, is small compared to the relative improvement that members of t_2 would perceive. Consequently, Antfarm breaks ties by awarding 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 minimizing download time, while the participants receive maximal marginal benefit whenever there is freedom in making a bandwidth allocation that is in line with the primary goal.

3 IMPLEMENTATION

The Antfarm implementation is centered around a token-based wire protocol that implicitly reveals peer dynamics to the coordinator. This section provides an overview of the Antfarm implementation, outlines the wire protocol and the use of tokens, and describes how tokens are used in the larger context of bandwidth allocation. We illustrate the common case first and treat the corner cases stemming from token misuse, peer misbehavior, and overall scalability in Sections 3.4 and 3.5.

3.1 Overview

An Antfarm deployment consists of two types of servers provided by the content provider. *Coordinators* manage the system by issuing tokens, computing response curves, and determining bandwidth allocations. *Seeders* expend their bandwidth to distribute blocks of files to peers. For small deployments, a single server machine can act as both coordinator and seeder, while large deployments will comprise multiple physical hosts.

Antfarm seeders are members of all swarms and distribute data blocks without downloading any themselves. They may be under the direct administrative control of the coordinator, or they may be deployed by ISPs to reduce their ingress bandwidth demand; in either case, they may be geographically distributed to improve bandwidth to peers. Seeders do not demand tokens from peers in exchange for blocks because they do not place resource demands on the system.

Peers interact with coordinators, seeders, and each other to download files. Each peer in Antfarm is identified by a certificate acquired from the coordinator during an initial, one-time registration. Once a connection with a peer has been established and the peer has been authenticated with the coordinator, wire messages identify peers using a public IP address and port pair that is shorthand for the verified certificate. Antfarm assumes that peers are either rational, where the protocol will incentivize them to contribute resources to the global pool, or malicious, where they may behave in a Byzantine manner; the protocol is resilient to such malicious hosts (see Section 3.5).

The Antfarm wire protocol is designed around peer-to-peer data exchange in return for *tokens*. A token is a cheap, unforgeable capability that the bearer may exchange for a data block in a given swarm. Logically, a token is composed of a unique, randomly generated ID string, an expiration time after which the token is invalid, a reference to the intended spender of the token, and a reference to the file for which the token should be spent. The coordinator records these four fields when it mints a new token for a particular peer. A token can only be spent by the peer to which it was issued in exchange for blocks of the designated file; tokens are not interchangeable between swarms.

Downloadable files in Antfarm are described by a ".ant" *swarm description*, analogous to a ".torrent" file, which contains the name of the file, the address and port

of the coordinator managing the swarm, data block size, and a hash of each data block.

3.2 A Peer's Perspective

An Antfarm peer joins a swarm by opening a connection to the swarm's coordinator and authenticating itself using its peer certificate. Once a connection has been established, all correspondence with the coordinator and peers occurs with the exchange of protocol messages summarized in Table 1. When a new peer joins a swarm, the coordinator sends the peer a subset of the peers in the swarm and an initial allowance of tokens unless the new peer has a history of malicious behavior. The peer can similarly join additional swarms, acquiring peer lists and initial tokens for each.

The basic data transmission protocol in Antfarm has three phases consisting of peer and block selection, data-for-token exchange, and bandwidth allocation.

Peers may determine their own criteria for selecting peers and blocks. This enables Antfarm peers to perform optimizations based on local information, reducing the burden on the centralized coordinator. The default behavior in Antfarm 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 the running average bandwidth achieved through that peer's history of interactions. Blocks are chosen using a rarest-first algorithm; peers 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 [30], the Antfarm 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 through an asynchronous notification. This is an advisory notification that causes the recipient to increase the priority of the named, underutilized peers. This is a no-cost optimization for Antfarm; 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 could be improved by the use of network proximity measures [19, 33, 41], though our current implementation does not yet include this optimization.

Once a peer (receiver) has chosen another peer (sender) and determined a suitable block for download, it sends a data-exchange request. If the sender has unchoked the receiver, it sends the requested block to the re-

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 1: **Antfarm 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.

ceiver. 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. This communication takes place every minute in the current implementation. 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.

Peers following the above protocol will face a stream of competing requests for data blocks. Peers use a leaky-bucket algorithm to restrict upload bandwidth according to the coordinator-prescribed allocation. Altruistic peers that finish downloading a file may remain in the swarm and continue to upload content, functioning similarly to seeders.

3.3 The Coordinator's Perspective

The coordinator collects statistics on peer network behavior, computes response curves and bandwidth allocations for each peer and seeder, and steers the swarm toward an efficient operating point. It affects these 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. This information is sufficient to maintain two key parameters for each peer p: the set of swarms T_p that p is a member of and a rolling average of its upload bandwidth u_p. In addition, the coordinator keeps track of the set of all seeders S and two pieces of state for each swarm: a set P_t of peers in swarm t and a response scatterplot for each swarm, represented as a collection of data points with associated time-decaying weights. Data points decay according to $1/t$ and are removed after 30 minutes.

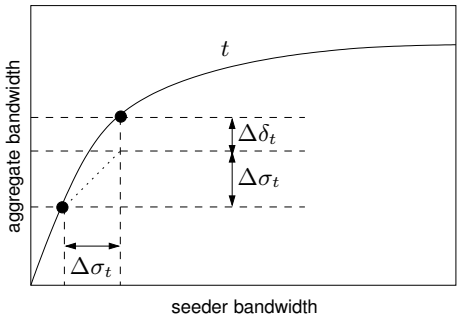

Figure 3: **Bandwidth allocation.** The black dots denote the allocation of bandwidth for swarm t before and after one iteration of allocation. For each $\Delta\sigma$ tasked to a seeder by the hill-climbing algorithm, a randomly selected peer with spare upload capacity is tasked with allocating a corresponding $\Delta\delta$. The dotted line has a slope of 1, accounting for the seeders' contribution to the swarm's aggregate bandwidth.

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 by fitting a piecewise linear function that respects the monotonicity and concavity constraints, contains a segment for each measurement point, and minimizes error using least-squares.

The coordinator computes the amount of bandwidth each seeder and peer should dedicate to each swarm based on the computed response curves, represented as two matrices σ and δ. For each swarm t, $\sigma_{s,t}$ captures the amount of bandwidth seeder s will dedicate to t, and $\delta_{p,t}$ captures the amount of bandwidth peer p is expected to dedicate to t. This determines the critical allocation of seeder upload bandwidth $\sigma_t = \sum_{s \in S} \sigma_{s,t}$ to swarm t in order to achieve a swarm aggregate bandwidth $(\sigma_t + \delta_t)$, where $\delta_t = \sum_{p \in P_t} \delta_{p,t}$ is the bandwidth component resulting from peer-to-peer uploads. The coordinator computes this allocation periodically, every 5 minutes in our current implementation, and also when the area under the curve has changed by more than 10%. In computing σ and δ the coordinator operates under two hard constraints. First, $\delta_p = \sum_{i \in T_p} \delta_{p,t}$ can never exceed p's upload capacity u_p. Second, the node must have the file to seed; a peer will never be tasked to upload blocks of a file it is not interested in downloading. The coordinator determines σ and δ iteratively. Initially, $\sigma_{s,t} = \delta_{p,t} = 0$ for all peers p, seeders s, and swarms t. The coordinator determines the allocation of bandwidth through a

greedy hill-climbing algorithm using the computed response curves and its knowledge of the seeders' upload capacities, illustrated in Figure 3. It allocates bandwidth in discrete units to the swarms whose response curves have the highest gradient, breaking ties in favor of the swarm with the lower value of $(\sigma + \delta)$, as described in Section 2. For each increase in seeder bandwidth $\Delta\sigma_t$ to swarm t, the algorithm chooses a peer at random from P_t with spare upload bandwidth and tasks it with uploading an additional $\Delta\delta_t$ to t, as prescribed by t's response curve. The coordinator continues the process until all seeder bandwidth has been allocated. The final peer allocation δ satisfies the two critical constraints described above and ensures that peer transfers within each swarm achieve the previously measured aggregate bandwidth based on the seeders' 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.

Antfarm's token protocol incentivizes peers to report statistics to the coordinator in a timely manner. A token's expiration time (5 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 enough 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 [32], 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 suggesting which peers are underutilized. The latter partly influences unchoking decisions as described previously. The coordinator determines this list for each peer by selecting a small subset of the top uploaders to that swarm, chosen randomly from a probability distribution determined by upload bandwidth.

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 KB/s (kilobytes per second). Such variances provide additional datapoints for the response scatterplot, enabling the system to overcome false local minima due to transient effects.

The coordinator does not enforce peers' compliance with the coordinator's directives in allocating their upload bandwidth. A peer is free to shift bandwidth away from one swarm in favor of another at its discretion. In such a scenario, the coordinator will simply observe a shift in the swarms' dynamics, which will be reflected in the response curves. In the next epoch, the coordinator will perform a new bandwidth allocation that takes the peer's behavior into account.

3.4 Scalability

The Antfarm coordinator is optimized to ensure that the logical centralization does not pose a CPU or bandwidth scalability bottleneck.

Shuttling tokens to and from the coordinator for each data block transaction is the main source of coordinator bandwidth expenditure. To reduce the burden, Antfarm does not rely on public-key cryptography to issue or exchange tokens. The Antfarm protocol minimizes the size of tokens on the wire, transmitting only relevant fields when tokens change hands. Only a token's ID, file reference, and expiration time are sent on the wire when the coordinator sends fresh tokens, and only the ID 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 ID and the identifier of the peer that spent the token. Using 4-byte token IDs, each token exchange requires less than 24 bytes of total bandwidth and less than 16 bytes of bandwidth at the coordinator for each data block of around 32-128 KB.

Antfarm uses highly compact versions of token identifiers to reduce bandwidth. A 4-byte ID 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 ID, 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 Section 3.5. Thus, with 4-byte token IDs, 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 Antfarm coordinator expends its bandwidth to send tokens to peers, receive spent tokens back from peers, and periodically send swarm allocations and lists of top contributors to peers and seeders. To alleviate the bandwidth demands placed on the coordinator, the Antfarm protocol enables the coordinator to be distributed hierarchically. A lead coordinator machine handles computing response curves and determining swarm bandwidth allocations. The remaining coordinators, called *token coordinators*, issue tokens, collect tokens back from peers, and periodically send each peer's upload and download rates to the lead coordinator each time the lead coordinator computes bandwidth allocations. The lead coordinator 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, so it can verify the token with the token coordinator that issued it. Thus, each token exchanged between peers involves at most two token coordinators.

Token coordination is an embarrassingly parallel task. The high ratio between token size and data block length ensures that the coordinator bandwidth is leveraged several thousand-fold. Section 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 machines.

The coordinator performs two periodic CPU-bound tasks: it computes response curves from scatterplots and allocates seeders' and peers' bandwidth. These tasks are computed centrally in order to derive bandwidth allocations based on the most recent measurements. Our current implementation on a 2.2 GHz CPU with 3 GB 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 lead coordinator can easily be replicated to mask network and host failures.

3.5 Security

A formal treatment of the security properties of the underlying Antfarm wire protocol is beyond the scope of this paper. Past work on similar, though heavier-weight, protocols [52] 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 the bandwidth allocation algorithm.

Antfarm makes 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 the Antfarm wire protocol, 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.

Antfarm requires peers to contribute bandwidth to their swarms, engage in legitimate token-for-block 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 simply comparing the token ID against its own registry of active tokens. 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 a fresh replacement token. The coordinator detects all counterfeit tokens, but when it detects an invalid token, it is unable to differentiate the peer sending the token from its ledger from the peer that originally spent the token as the culprit. Therefore, it notifies both peers of the forgery so the honest peer can blacklist the culprit.

To hold peers more accountable for their actions when the coordinator is unable to precisely identify malicious peers, Antfarm peers 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, the current Antfarm implementation avoids this to reduce burden on the coordinator.

Using cryptographically signed tokens can provide stronger guarantees than Antfarm currently does 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. Antfarm does not implement a cryptographic scheme because the added overhead is not accompanied by a clear increase in performance.

It is possible for Antfarm 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 [12]. 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 Summary

The Antfarm protocol strikes a balance between micromanaging peers and granting them freedom over block transfers. Tokens that must be returned to the coordinator enable the coordinator to collect accurate statistics on swarm dynamics and peer behavior. Systems such as BitTorrent, which grant peers full autonomy, do so at the expense of control and efficiency. At the other extreme, a centralized solution that precomputes the entire download schedule for all participants would limit peers' ability to quickly determine which peers have blocks they require and retrieve them without intervention. Antfarm provides a hybrid approach that leaves peers free to determine their own local behavior while extracting sufficient information from the network to compute the globally optimal allocation of available bandwidth among swarms.

4 EVALUATION

We have implemented the full protocol described in this paper, as well as a simulator of the Antfarm and BitTorrent protocols. The Antfarm deployment runs on Windows, Linux, and Mac OS X. Both the implementation and the simulator contain optimizations present in version 5.0.9 of BitTorrent, including optimistic unchoke, regular unchoke, and local-rarest-block-first. For the experiments in this section, Antfarm's system parameters (block size=64KB, optimistic unchoke interval=30s, regular unchoke interval=10s) are identical to those found in this version of BitTorrent. We pick upload and download bandwidths representative of cable-connected end nodes.

This section evaluates the performance of the Antfarm protocol in comparison to BitTorrent and traditional client-server approaches. Through simulations, we illustrate scenarios under which BitTorrent misuses its seeder capacity and show how Antfarm can achieve qualitatively higher performance by allocating seeder bandwidth to swarms that provide the highest return. A PlanetLab deployment confirms Antfarm's allocation strategy under realistic network conditions. Lastly, this section shows that Antfarm's coordinator can scale to support large deployments using modest resources.

4.1 Simulations

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 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

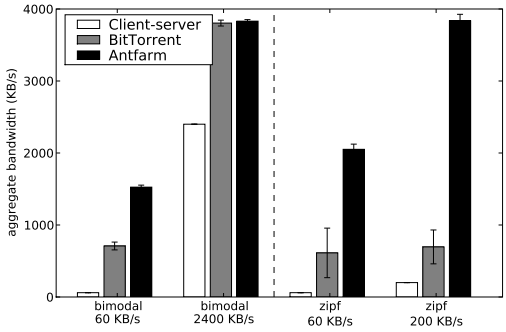

Figure 4: **Aggregate bandwidth for a client-server system, BitTorrent, and Antfarm.** When seeder bandwidth is plentiful, even a client-server model can deliver high throughput. When seeder bandwidth is limited, Antfarm outperforms BitTorrent by allocating bandwidth to swarms that receive the most benefit. Error bars indicate 95% confidence intervals.

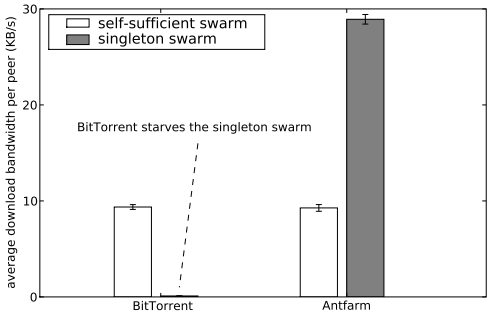

Figure 5: **Bandwidth of a singleton swarm and a large, self-sufficient swarm.** Even though a self-sufficient swarm can saturate its peers' bandwidth without seeder bandwidth, BitTorrent awards bandwidth to peers in the swarm. In contrast, Antfarm awards seeder bandwidth to the singleton swarm because it receives high marginal benefit.

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, Antfarm steers the seeder's capacity to swarms where blocks can be further shared among peers.

Antfarm's dynamic bandwidth allocation adapts well to changes in swarm dynamics. A well-known phenomenon is that 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 at the exclusion of available peer bandwidth, leading to a shortage of seeder bandwidth for other, needier swarms.

Figure 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 scenario, the seeder has a capacity of 100 KB/s, and each peer downloads a 10 MB file with 30 KB/s download capacity and 10 KB/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 approximately 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 6 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 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 allocate 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 KB/s download capacities and 20 KB/s upload capacities, and the self-sufficient swarms have peer interarrival times of 3, 6, 12, 24, 50, and 100 seconds. In the left-hand graph, BitTorrent ded-

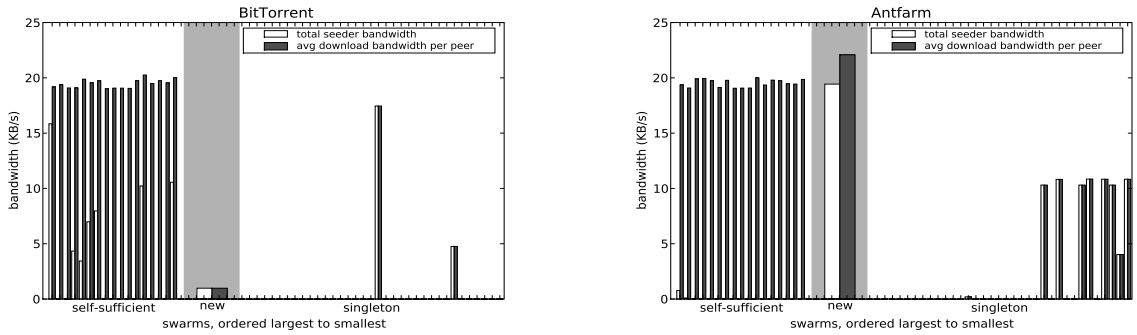

Figure 6: **BitTorrent versus Antfarm serving the middle of the popularity distribution.** The shaded region indicates a new swarm of 5 peers. Swarms to its left are self-sufficient; swarms to its right are singletons. BitTorrent (left) starves the new swarm, favoring to dedicate bandwidth to the many peers in self-sufficient swarms. Antfarm (right) 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.

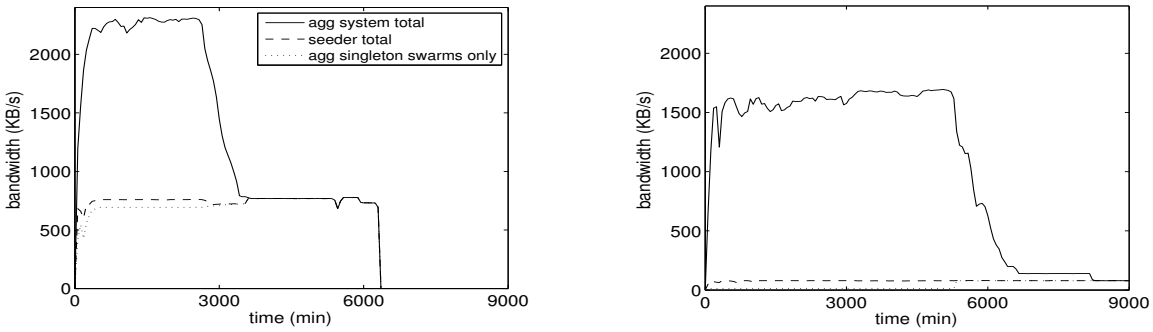

Figure 7: **Time versus bandwidth for Antfarm.** The figures show seeder and aggregate bandwidths of the bimodal experiment with seeder bandwidths of 800 KB/s (left) and 80 KB/s (right). Antfarm follows drastically different bandwidth allocation strategies (dashed and dotted lines) to achieve high throughput (solid lines).

icates 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 right-hand 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.

Figure 7 shows Antfarm's bandwidth allocation over time to provide insight into Antfarm's strategy. The left-hand 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 right-hand 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.

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 bandwith.

4.2 PlanetLab Deployment

We tested Antfarm's performance through a PlanetLab [5] deployment. To demonstrate Antfarm's response curves in practice, Figure 8 shows a measured response curve of a swarm comprised of 25 PlanetLab nodes, each

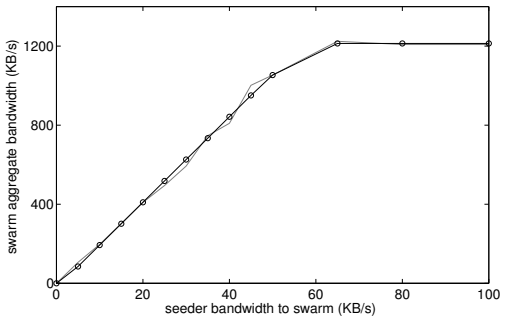

Figure 8: **A response curve for a swarm consisting of 25 PlanetLab nodes, each with an upload capacity of 50 KB/s.** Each data point is based on the average swarm aggregate bandwidth over 10 minutes. Real-world response curves confirm simulations.

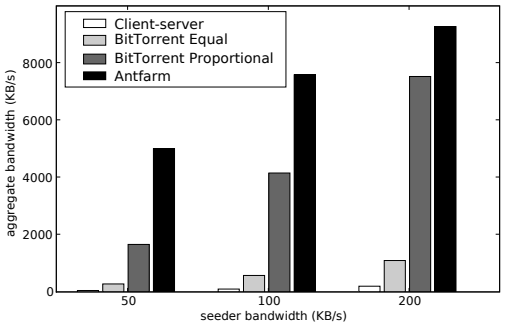

Figure 9: **PlanetLab experiments showing aggregate bandwidth in Antfarm versus BitTorrent and client-server.** 300 PlanetLab nodes are distributed among swarms ranging in size from 1 to 100. Antfarm achieves high average performance by making efficient use of limited bandwidth.

with an upload capacity of 50 KB/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.

Figure 9 compares the aggregate bandwidth achieved by Antfarm, BitTorrent, and traditional client-server downloads across 300 PlanetLab nodes, each with an upload capacity of 50 KB/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 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 its 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.

4.3 Scalability

In this section, we examine how the Antfarm coordinator scales. We examine the steady-state bandwidth cost of running a coordinator in a setting where peers download a file made up of 64 KB blocks with upload and download capacities of 64 KB/s.

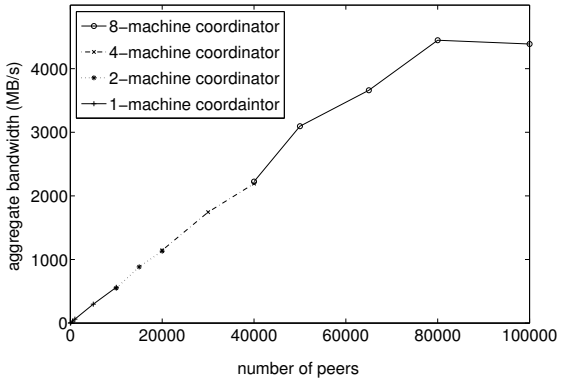

Figure 10: **Aggregate bandwidth of swarms managed by varying sizes of coordinator clusters.** Each coordinator machine runs on a PlanetLab node with an artificial bandwidth cap of 100 KB/s to limit scalability. The task of the token coordinator is embarrassingly parallel; the system capacity scales linearly with the size of the coordinator cluster.

Figure 10 shows the bandwidth consumption at the coordinator as a function of the number of peers based on experiments run on PlanetLab. In the experiment, the lead coordinator and each token coordinator ran on its own PlanetLab node, and peers were simulated across other PlanetLab nodes, engaging in the Antfarm protocol without sending actual data. The results show that even for large numbers of peers, the bandwidth consumption at the coordinator is modest. A coordinator running on a single PlanetLab host suffices for deployments of 80,000 peers or more. To demonstrate the scalability of the hierarchically distributed coordinator, we test a coordinator distributed across multiple PlanetLab hosts in a system with an aggregate bandwidth approaching

5 GB/s. 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 node to 100 KB/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 node, which is able to handle the tokens and peer lists of approximately 9000 peers before its performance reaches a plateau. Adding a second such coordinator node doubles the capacity of the system. Because the token coordinators engage in a massively parallel task with little communication overhead, increasing the number of coordinators linearly increases the maximum supported number of peers.

5 RELATED WORK

There has been much past work on content distribution, which can be grouped roughly into work on content distribution networks, token-based systems, and multicast and streaming systems.

CDNs: Content distribution networks are scalable systems used to alleviate server load, reduce download times, and avoid network hotspots. Akamai [31], for example, is a widely deployed infrastructure-based CDN that many content providers rely on to distribute their content. Similarly, cooperative web caching [7,25,27,57, 58] removes load from origin servers. ECHOS [34] 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 optimize 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.

Peer-to-peer CDNs effectively shift bandwidth costs from the content provider to clients. BitTorrent [8] is one of the most popular client-based peer-to-peer CDNs, and studies consistently show that BitTorrent traffic constitutes a significant fraction of Internet traffic [43, 53]. Piatek et al. [46] augment the BitTorrent protocol to enable peers to share reputation information through one level of intermediary nodes; it does not address the issue of multiple swarms. CoBlitz [44] is an HTTP-based content distribution network that splits a file into chunks, which are cached at distributed nodes. Choffnes et al. [15] reduce cross-ISP traffic in peer-to-peer systems by harvesting data from existing CDNs for locality information. Shark [3] and ChunkCast [9] reduce client-perceived download latency via a structured overlay, and Coral [23] and Bamboo [50] assist clients in finding nearby copies of data. Antfarm similarly shifts cost to clients; however, it retains control of network behavior by carefully allocating bandwidth to each swarm.

Further, many systems such as the Data Oriented Transfer (DOT) architecture [42, 54] use peer-to-peer swarming to speed up downloads.

Token-based Incentives: Early model and analysis by Qiu and Srikant [49] 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, BitTyrant [45], BitThief [35], and Sirivianos et al. [51], has demonstrated that average download times currently depend on significant altruism from high capacity peers that, when withheld, reduces performance for all users. Further, BitTorrent's tit-for-tat mechanism only operates within an individual swarm; it does not provide information on how to allocate resources, such as seeder bandwidth, among swarms.

Dandelion [52] and BAR gossip [36] avoid the problem of 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 [10, 37, 47, 59] similarly rely on cryptographically protected tokens for fair resource exchange, and optionally provide additional features such as spender anonymity. Antfarm's token system is domain-specific and significantly lighter-weight than these approaches.

Decentralized resource allocation in peer-to-peer systems requires incentives for participants to contribute resources. Ngan et al. [39] suggest cooperative audits to ensure that participants contribute storage commensurate with their usage. Samsara [16] 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 [56] and SHARP [22] 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.

Streaming and Multicast: Multicast and streaming are alternative designs for distributing content. For instance, the seminal work by Deering proposed IP multicast to efficiently deliver content to multiple destinations [20]. Deployment difficulties with global IP multicast [18] led to application-level multicast systems such as End System Multicast [14], Your Own Internet Distribution (YOID) [21], and others [60].

Several techniques have been proposed to distribute data efficiently using application-level multicast. Overcast [26] distributes content by constructing a bandwidth-optimized overlay tree among dedicated infrastructure nodes. SplitStream [13] 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 [29] and Bullet′ [28] also use a randomized overlay mesh to distribute data. Chainsaw [48] is a peer-to-peer multicast based on an unstructured overlay mesh in which peers explicitly request packets from neighbors. This mechanism ensures that peers are able to receive all packets and avoid receiving duplicate packets. ChunkySpread [55] is a hybrid that uses both structured and unstructured overlays to distribute content. Antfarm differs from streaming multicast systems in that it aims to maximize aggregate system bandwidth for multiple concurrent batch downloads.

Another set of work proposes augmenting BitTorrent-like protocols to accommodate streaming video in a peer-to-peer setting. BASS [17] exemplifies this approach by adding peer-to-peer interactions to a client-server model where peers stream video from the server while trading blocks with other peers to alleviate load on the server in the future. Antfarm also incorporates a peer-to-peer protocol to alleviate load, but manages the interactions via the coordinator to achieve high throughput for multiple swarms. Siddhartha et al. [4] propose a BitTorrent-like protocol with small neighborhoods of topographically close peers for exchanging blocks, using heuristics to handle swarms of heterogeneous link capacities.

Finally, many streaming and multicast architectures use coding to increase content delivery reliability [2, 6, 24, 38]. Integrating coding techniques into Antfarm could further improve performance.

6 CONCLUSIONS

In this paper we introduced Antfarm, a peer-to-peer content distribution system for the batch dissemination of large files. Antfarm explores a novel space in the design of swarming protocols; whereas past systems avoid all vestiges of centralization for both technical and legal reasons and suffered from lack of coordination across swarms, Antfarm examines how modest planning by a centralized coordinator can help a set of competing swarms achieve high performance.

The key to Antfarm's performance is its restatement of the download management task as an optimization problem. The hill-climbing algorithm we propose effectively leverages available bandwidth, accommodates desired minimum bandwidth limits, avoids starvation, and enforces desired swarm priorities. The wire-level protocol enables performance information to be extracted from the network, enabling a practical deployment that reacts to changing network and swarm conditions. Even though the approach embodies a logically centralized coordinator, the computational requirements of the coordinator are modest, the bandwidth requirement is feasibly small, and the coordinator carries out an embarrassingly parallel task that is easy to replicate across datacenters. PlanetLab deployments and simulations indicate that the system is practical, scalable, and capable of achieving significantly higher bandwidth than previous approaches.

Acknowledgments: We would like to thank Hakim Weatherspoon for his help refining the protocol, Aaron Lenfestey for implementing the least-squares algorithm that generates response curves, Ymir Vigfusson for his help formalizing swarm metrics, and Steve Gribble for his input in early discussions of this work. This work was supported in part by NSF-CAREER 0546568.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] M. Calore. Zudeo Announces Deal With Bbc. *Wired Blog Network*, December 19 2006.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] S. E. Deering. Multicast Routing In Internetworks And Extended Lans. *SIGCOMM Conference*, Stanford, CA, August 1988.

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

[22] 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.

[23] 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.

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

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

[26] 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.

[27] 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*, 1999.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

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

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

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

[40] 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.

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

[42] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting Similarity For Multi-source Downloads Using File Handprints. *Symposium on Networked System Design and Implementation*, Cambridge, MA, April 2007.

[43] 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.

[44] 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.

[45] 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.

[46] 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*, 2008.

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

[48] 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.

[49] D. Qiu and R. Srikant. Modeling And Performance Analysis Of Bittorrent-like Peer-to-peer Networks. *SIGCOMM Conference*, Portland, OR, August 2004.

[50] 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.

[51] 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*, 2007.

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

[53] 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.

[54] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An Architecture For Internet Data Transfer. *Symposium on Networked System Design and Implementation*, San Jose, CA, May 2006.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

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

[60] Y. Zhu, W. Shu, and M.-Y. Wu. Approaches To Establishing Multicast Overlays. *IEEE International Conference on Services Computing*, 2, July 2005.