Peer-assisted Content Distribution on a Budget[☆]

Pietro Michiardi[a], Damiano Carra[b,*], Francesco Albanese[a], Azer Bestavros[c]

[a] *Networking and Security Department, Eurecom, France*
[b] *Computer Science Department, University of Verona, Italy*
[c] *Computer Science Department, Boston University, USA*

Abstract

In this paper, we propose a general framework and present a prototype implementation of peer-assisted content delivery application. Our system – called Cyclops – dynamically adjusts the bandwidth consumed by content servers (which represents the bulk of content delivery costs) to feed a set of swarming clients, based on a feedback signal that gauges the real-time health of the swarm. Our extensive evaluation of Cyclops in a variety of settings – including controlled PlanetLab and live Internet experiments involving thousands of users – shows a significant reduction in content distribution costs when compared to existing swarming solutions, with a minor impact on the content delivery times.

Keywords: Internet Content Distribution, Peer-assisted Content Distribution

[☆]This research was supported in part by NSF awards #0720604, #0735974, #0820138, and #0952145 and has been partially supported by the French ANR-VERSO project VIPEER.

[*]Corresponding author

Email addresses: pietro.michiardi@eurecom.fr (Pietro Michiardi), damiano.carra@univr.it (Damiano Carra), francesco.albanese@eurecom.fr (Francesco Albanese), best@cs.bu.edu (Azer Bestavros)

1. Introduction

Traditional Content Delivery Networks (CDNs) such as Akamai [1] were conceived as special-purpose services catering almost exclusively to large, highly-popular content providers such as iTunes and CNN. Today, however, the advent of cheap Internet storage and delivery services – for example Amazon S3 [2] and CloudFront [2] – make it possible for much smaller-scale content providers to deploy and provision their own "ad hoc" CDNs in an almost real-time fashion.

In general, the major cost contributor for CDNs is the bandwidth consumed to deliver content from content servers to the clients. Essentially, the bandwidth costs associated to content distribution are reflected into "content delivery plans" that customers of a CDN service are required to pay. With the goal of bounding such costs – and thus attract a variety of customers, including small content producers with budget contraints – it is increasingly the case that content delivery solutions are evolving from simple *client-CDN* interactions (reminiscent of the traditional client-server model) into *swarm-CDN* interactions, wherein the content servers are not merely responding to individual client requests, but rather to the collective demand of a set of clients that are looking for the same content, usually referred to as "swarm." Indeed, to reduce bandwidth requirements, an increasing number of CDN solutions (including those offered by major market players such as Akamai [1], Limelight [17], and Amazon [2]) rely on swarm-based, peer-assisted approaches that leverage the uplink capacity of end-users to reduce the CDN bandwidth consumption. This approach, which is particularly effective for highly-popular content, can be seen as seamlessly bridging client-CDN and

swarm-CDN interactions: For less-popular content, a peer-assisted CDN behaves as a traditional CDN system, whereas for highly-popular content, it behaves as a peer-to-peer system.

Existing cloud-based peer-assisted CDNs rely on swarm-based protocols such as BitTorrent [5]. While such swarm-based protocols are quite efficient for exchanging content among peers (in terms of download time, resource utilization, and fairness), they are not designed to provide the content source with the means to gauge the marginal utility of its contribution to the swarm. Specifically, in a peer-assisted CDN setting, swarm-based protocols do not enable the content server to gauge and hence manage the inherent tradeoffs between server bandwidth utilization and the efficacy of content delivery. This is precisely the capability that the work presented in this paper aims to provide.

Paper Scope and Contributions: In this paper, we propose a framework for peer-assisted CDN solutions in which the content server is able to adjust the bandwidth it contributes to the swarm (the set of clients downloading content) so as to achieve a specific *objective* based on a *feedback signal* related to the state of the swarm. Our framework is general enough to allow for many possible combinations of objectives and feedback signals. For instance, the objective may simply be to keep the swarm alive based on a feedback signal indicating the level of redundancy for particular pieces of content in the swarm. Alternately, the objective may be to ensure a desirable level of service based on a feedback signal gauging average delivery time to clients.

To establish a reference model for these as well as other combinations of

objectives and feedback signals, in Section 2, we discuss the cost-performance tradeoff for peer-assisted content delivery. Our findings suggest the existence of a quiescent (close to optimal) operating point beyond which the marginal utility from additional content server bandwidth utilization is negligible.

Based on this understanding, in Section 3, we present the design and prototype implementation of CYCLOPS, a peer-assisted content delivery service. The content server in CYCLOPS is able to modulate its bandwidth contribution to the swarm so as to remain in the vicinity of the aforementioned quiescent operating point – thus minimizing its cost without sacrificing performance. Our design relies on the feedback signal provided through an on-line monitoring tool, which we have implemented as part of CYCLOPS.

To demonstrate the effectiveness of our approach, in Sections 4 and 5 we report on a fairly extensive series of Internet experiments, in which we compare the performance of CYCLOPS to those of "open-loop" swarm-based protocols used by cloud-based content delivery services. Our experiments are carried out both in a controlled environment (by delivering content to PlanetLab clients) and in the wild (by delivering content to a real Internet user population). These experiments show that our feedback-based approach reduces drastically the volume of data served from the cloud (and hence the cost incurred by the content provider) with negligible performance degradation.

For illustrative purposes, our experimental setting involves an instance of CYCLOPS deployed as a "cloud-based" service. In particular, we used Amazon EC2 instances to create a possible deployment scenario and measure the performance and costs of content delivery. In live experiments involv-

ing more than 10,000 users exhibiting highly dynamic arrival and departure patterns, we were able to document monetary savings of up to two orders of magnitudes for our system.

2. Cost-Performance Tradeoff

When designing a content server, one of the first problem to solve is the bandwidth allocation, *i.e.,* how many resources are necessary to provide the content delivery service. In this Section we consider the tradeoff between the bandwidth utilization by a content server to the average delivery time perceived by a set of swarming users (clients).

To this aim we have developed a simple Markovian model that provides some interesting insights. Since the details of the model are not necessary for understanding the design of our system, we refer the interested reader to Appendix A for a complete description of the model. Here we report only the results with some sample inputs.

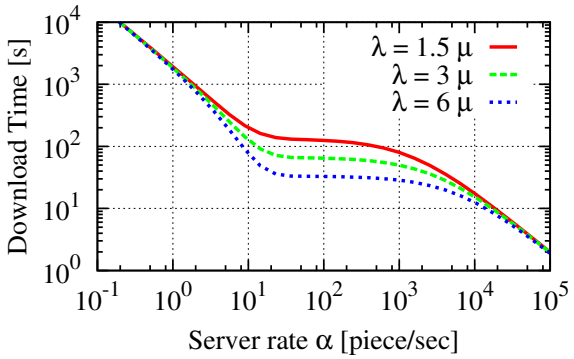

Figure 1: Download time as a function of the content server rate.

Figure 1 quantifies the tradeoff between the server bandwidth utilization (*i.e.*, the average upload rate α of the content server) and the average delivery

rate to clients involved in a swarm with upload capacity λ – the parameter μ represents the departure rate, *i.e.,* the rate at which the content disappears, in contrast to λ, which can be interpreted as the rate at which the content is replicated (for the details see Appendix A).

The figure shows three operating regions. The first region (left-side of the plot) is when α tends to zero, resulting in piece starvation, and a corresponding increase in download time. The second operating region (right-side of the plot) is when α tends to values that far exceed λ, resulting in a client-server-like mode of operation. The third and more interesting operating region is an intermediate one, within which an increase in α does not result in a corresponding decrease in download time. The "width" of this region depends on the health of the swarm, which is a function of the content popularity captured by the client departure rate μ, and the mean client upload bandwidth λ.

The behavior described by our model suggests the existence of a quiescent operating point (at the transition between the first and second operating regions depicted in Fig. 1), beyond which the marginal utility from additional bandwidth utilization is negligible. A content server operating around this quiescent point would be fully leveraging the uplink bandwidth of its clients, while minimizing its own cost: *operating below this quiescent point would jeopardize performance, and operating above this quiescent point would be cost inefficient.* Note that previous models, such as the one proposed in [18], show that the download time depends linearly from the server bandwidth, while our model highlights the existence of different operating regions.

We are now ready to describe the design and prototype implementation

of a content server that uses a feedback signal to adjust its bandwidth contribution to the swarm so as to remain in the vicinity of a nominal quiescent operating point. It is important to notice that the prototype we present in the following is *not* an explicit implementation of the model: the design is inspired by the key observations made above.

While our framework allows for many combinations of objectives and feedback signals, in the remainder of this paper we focus on the objective of maximizing the performance per unit cost, using the availability of content in the swarm as the feedback signal.

3. System Design and Implementation

We now present the design of CYCLOPS, our peer-assisted content delivery service. As depicted in Fig. 2, our CYCLOPS service consists of a *content server* and a *swarm monitor*. The swarm monitor interprets the signaling messages exchanged between swarming clients, and generates a feedback signal that enables the content server to gauge the marginal utility of its contribution to the swarm. The content server participates in the swarming protocol to satisfy client requests, but only *feeds* the swarm when its contribution is deemed necessary (based on the feedback signal). In CYCLOPS, the swarm feeding rate is set to maximize the swarm performance-per-unit-cost, using the availability of content in the swarm as the feedback signal. As established in our model in Section 2, the quiescent operating point for this objective is the minimum rate that avoids swarm starvation.

CYCLOPS is conceived to work with *any* swarm-based application/protocol that features (1) a coordinating entity that tracks all swarm participants, en-

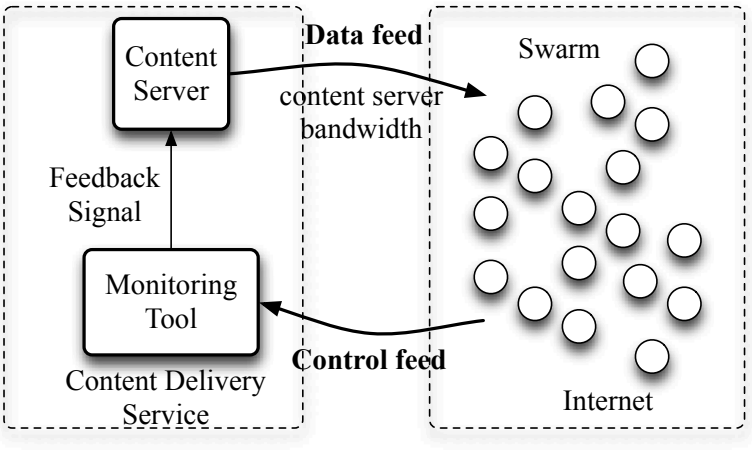

Figure 2: Overview of Cyclops Architecture: The content server and swarm monitor reside in the CDN in distinct virtual machines, with bandwidth used for data feed (to the swarm) and control feed (from the swarm).

abling them to establish peer-to-peer connections; (2) content that is divided into pieces to be distributed/exchanged independently; and (3) a control messaging scheme used by swarm participants to advertise piece availability.

For practical reasons, we present our system and conduct our experiments focusing on a single content server, used to deliver a single content (file) to a set of clients. Problems related to concurrent swarms are orthogonal to our approach, and the solutions proposed in the literature, *e.g.*, [20], can be integrated independently. Similarly, issues related to the efficiency of the distribution process, solved using approaches based on traffic locality, are complementary to our solution, and previous work on this topic, *e.g.*, [8, 9], can be incorporated seamlessly.

Cyclops was conceived and implemented as a service that can be deployed on any content delivery platform. As an illustrative example, in this

work we focused on a "cloud-based" service: we used the Amazon Elastic Compute Cloud (EC2) environment, and produced an Amazon Machine Image (AMI) that supports *both* the content server and the swarm monitor functionalities.

The source code of CYCLOPS can be found in [10].

3.1. The CYCLOPS *Swarm Monitor*

Swarm monitoring in CYCLOPS is achieved using a set of components that we called the On-line Feedback (OF) nodes. OF nodes connect to a live swarm, but neither download nor upload content: they monitor *all* clients in the swarm and collect signaling messages they exchange. Using this information, OF nodes construct snapshots in time that characterize the health/performance of the swarm. In our particular implementation, these snapshots are used to derive the instantaneous piece availability, which constitutes the feedback signal fed to the CYCLOPS content server using a complementary protocol.

To ensure scalability (and seamless elasticity of the service), we adopted a distributed design for OF nodes, whereby new clients joining the swarm are assigned to different OF node to balance load. Accordingly, a swarm S is partitioned into N_p non-overlapping sets, where N_p is the number of OF nodes in the system. Swarm partitioning is achieved using consistent hashing [13]: each OF node is responsible for a fraction of the key-space, defined by the client ID (*e.g.*, IP address).

3.2. The CYCLOPS *Content Server*

The main objective of our approach is to minimize server bandwidth consumption without running the risk of starving the swarm. Based on the feedback signal provided by the swarm monitor, the content server feeds the swarm only when necessary, *i.e.*, when piece availability falls below a desirable threshold. To that end, in our design we adopted an ON/OFF control strategy, whereby the content server operation oscillates between two states: *serving* and *idle*.

When in the *serving state*, the content server dedicates its full uplink capacity to serve missing pieces of content. By design, the server avoids injecting duplicate pieces into the swarm. The rationale for doing so is that pieces can be quickly replicated by the swarm participants themselves. All clients connected to the content server are induced to request the set of missing pieces, which constitute the *serving set* maintained by the content server: this is possible since the server masquerades as a set of virtual clients holding a fraction of all available pieces. This serving set is partitioned into k non-overlapping subsets that are announced as "available." For instance, if the serving set consists of pieces {1,2,3,4} and $k = 2$, then k messages each announcing pieces {1,2} and {3,4}, respectively, will be sent to k users that will eventually issue download requests. Once a piece has been served, it is removed from the serving set, provided that the swarm monitor has confirmed the presence of the piece in the swarm. When the server has finished injecting all missing pieces into the swarm, it transitions to the *idle state*.

When in the *idle state*, the content server simply closes all connections

to remote clients, and refuses any incoming connection. The content server remains in the idle state until the feedback signal triggers a transition to the *serving state*.

4. Experimental Methodology

In this Section, we summarize the specifics of the CYCLOPS instance we have experimented with, along with various details regarding its illustrative deployment on a commercial cloud provider. We also describe the different types of experiments we have conducted, both in a controlled environment (involving PlanetLab clients under our control), and in the wild (involving thousands of real Internet users accessing content we advertised and made available).

BitTorrent-based Swarming: As we alluded to in Section 3, CYCLOPS can be instantiated to work with any swarm-based content distribution protocol, supporting a specific set of features. For experimental purposes, we created an instance of CYCLOPS that is compatible with the popular BitTorrent (BT) client. Note that, in all our experiments, clients execute unmodified BT code. This choice is partly motivated by the wide adoption of BT by Internet users, as well as its adoption by many content delivery services (including Amazon S3 and many others [6]) as an underlying swarming protocol. The details of the BT protocol and algorithms are not essential to understanding CYCLOPS, thus we refer interested readers to [16] for a technical description of BT. Here we only mention that the coordinating entity that maintains the list of clients in the swarm is called the *tracker*, and that the two control messages used by BT to advertise pieces available at a client

are the `have` and the `bitfield` messages: they indicate the availability at a client of a specific (single) piece, and of a set of pieces, respectively [16].

In the remainder of this paper, we use open-loop-BT to refer to a standard peer-assisted content delivery system based on BitTorrent, whereas we use Cyclops to refer to our "feedback-controlled" content delivery service.

Deployment Details: In our experiments, we used Amazon's Elastic Computing Cloud (EC2) to host, on *separate* virtual machines, the open-loop-BT content server (called the *seed*), the tracker, and Cyclops (including the content server and the swarm monitor). To mitigate the negative impacts on networking performance due to shared resources (CPU and I/O) in a virtualized environment, we used large EC2 instances, which were all located in a *single* US-based data center. Our open-loop-BT and Cyclops content servers were well-provisioned, with an upload capacity of 2.4 Mbps. Note that, in our experiments, a single OF node proved to be sufficient to monitor the entire swarm fed by Cyclops.

As for the threshold used to trigger from the *idle state* to the *serving state*, in order to let the system work using the minimum amount of bandwidth, we have set it to one piece.

Flash Crowd Experiments: To emulate a *flash crowd* arrival process – representative of a scenario in which a large number of clients exhibit a sudden interest in some specific content – we deployed a set of clients on PlanetLab machines, whereby all clients initiate their requests as a result of a centralized trigger: clients start downloading the content within 1 minute of that trigger signal. Once a user is done downloading the content it continues to serve other clients until the end of the experiment. We conducted our

experiments using two flash crowd sizes of $L = 50$ and $L = 300$ clients, respectively. In order to minimize the resource utilization of PlanetLab nodes, we used a homogeneous configuration with an application level cap of 160 Kbps for the client's uplink capacity, which is the default setting for BT. The content size was set to 50 MB.

Waves of Arrivals Experiments: We synthesized extreme swarm dynamics on PlanetLab, with the goal of studying CYCLOPS under stress: in practice, we created a scenario in which availability problems would hinder the content distribution process, requiring CYCLOPS to intervene more often than in a real swarm. The dynamics consisted of three successive bursts of client arrivals: a first burst of 100 clients arrive in a 10-minute span and leave after completing their download (within 50 minutes of arrival); a second burst of 100 clients join the swarm just before the mass exodus of the first wave of users. This process is then repeated for a third burst of arrivals. The interval between the mass exodus from one wave and the burst of arrivals from the next wave is set up in such a way that there would not be sufficient time for content pieces to propagate fully from the clients of one wave to the next (which should cause the swarm monitor's feedback signal to trigger the CYCLOPS content server to rev up its contribution to the swarm). As before, the client's uplink capacity was capped at 160 Kbps, and the content size was set to 50 MB.

Live Internet Experiments: We conducted experiments to evaluate our system under realistic CDN operating conditions, including web-driven arrival and departure processes for users drawn from a diverse set of ISPs and with diverse software settings. To do so, we distributed a non-copyrighted

movie packed in a 350MB file. We created two distinct *torrent* meta-files (one for distribution using CYCLOPS and the other for distribution using open-loop-BT), and we publicized both simultaneously on popular content search web-sites. In these experiments, both the CYCLOPS and the open-loop-BT content servers had no cap on their uplink capacity (beyond what is possible through a large EC2 instance), and needless to say, we had no control on the settings (or even the BT variants) of the clients.

We note that, due to their very nature, live Internet experiments cannot be repeated: as such, they should be regarded as illustrative deployment scenarios involving real Internet users that exhibit un-controlled and un-conditioned behavior.

Performance Metrics: In all of our experiments, we considered two main performance metrics. From the content server perspective, we measured the aggregate volume of data uploaded during an experiment, *i.e.*, the server bandwidth utilization. Since content servers are under our control, we can measure their bandwidth utilization using local log files. From the client side, we measured the content delivery times. For PlanetLab experiments, we did that by collecting application-level logs from the clients. For live experiments, where we do not have access to client logs, we measured the content delivery times using our swarm monitor, which aggregates information provided by OF nodes. The accuracy of this approach was validated using the PlanetLab experiments: we compared the download times computed using individual log files (of PlanetLab clients) to those obtained from OF nodes, and verified the match between the empirical cumulative distribution functions of download times for the two methodologies. Furthermore, to assert

the statistical significance of our results, our PlanetLab experiments were performed five times for each configuration.

5. Experimental Results

5.1. Flash Crowd Experiments

End-users' performance in downloading content is expressed in terms of individual download times. Figure 3 reports the most important percentiles (25th, 50th and 75th) of the empirical cumulative distribution function (ECDF) of download times.

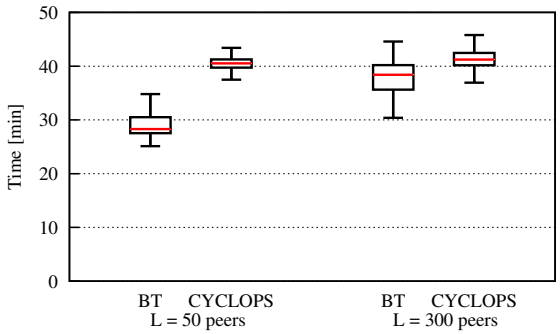

Figure 3: **Flash Crowd**: content download times (file size: 50MB).

As a general trend, we observe that the median download time of open-loop-BT swarms is lower than that of CYCLOPS swarms, with the gap reduced in larger swarms. The reason lies in the fact that an open-loop-BT seed keeps feeding the swarm during the whole experiment, resulting in a larger fraction of users receiving data from the content server itself (which is faster than the user), and hence the shorter content delivery time. Furthermore, we note that aside from visible but relatively small variations, the download time for CYCLOPS clients was less sensitive to the swarm size.

Table 1: **Flash Crowd**: average server load (file size: 50MB)

	BT	CYCLOPS
$L = 50$	12.2	1
$L = 300$	15.36	1

The above explanation is further confirmed by the results in Table 1, which reports the average bandwidth utilization expressed in volume of data served by both the CYCLOPS and the open-loop-BT content servers, normalized by content size. An open-loop-BT seed injects the swarm with 10–15 times the size of the original content, whereas CYCLOPS feeds the swarm only when necessary, which given the static nature of this experiment is once. These results corroborate the intuition discussed in Section 2. A content server that can gauge the marginal utility of its contribution to a swarm can settle in the vicinity of an operating point in which an additional expense of server bandwidth resources has a marginal effect on the swarm performance.

5.2. Waves of Arrivals

Figure 4 shows the key percentiles of the empirical cumulative distribution function (ECDF) for the delivery times experienced by clients in the successive waves of arrivals. In this case, the difference between the delivery times achieved by CYCLOPS and the open-loop-BT content servers is small: the median value of the distribution indicates an advantage of roughly 15% in favor of the latter.

Table 2 shows the average volume of data served by both schemes, as

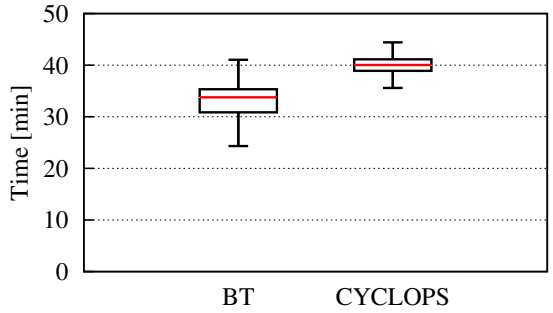

Figure 4: **Waves of Arrivals**: content download times (file size: 50MB).

well as information on traffic overhead (namely, volume of control messages involving bandwidth resources). For CYCLOPS, we show the aggregate overhead incurred by the content server and the swarm monitor. For completeness, we report the feedback traffic exchanged between the content server and OF node, noting that these messages are exchanged within the confines of the cloud and hence do not entail additional costs.

The data in Table 2 corroborates our conclusion that CYCLOPS achieves low server bandwidth utilization, even when the system is artificially stressed by complex client dynamics.

Next we examine the evolution in time of the feedback signal (namely, system-wide piece availability) generated by the CYCLOPS swarm monitor and the content server state transitions it triggers. Let M be the number of pieces into which a file is divided, and let $I(i,t)$, $i = 1,\ldots,M$ be the indicator function for piece i at time t, *i.e.*, $I(i,t) = 1$ if there is at least one copy of piece i at time t, otherwise $I(i,t) = 0$. The availability feedback signal $A(t)$ at time t is computed as:

$$A(t) = \frac{\sum_i I(i,t)}{M} \tag{1}$$

Table 2: **Waves of Arrivals**: server load & overhead (file size: 50MB)

	BT	Cyclops
Normalized server load	39.86	1.5
Outgoing overhead	55 KB	52 KB
Incoming overhead	2560 KB	716 KB
Feedback overhead	–	145 KB

Figure 5 shows the time-series for the swarm size, the availability feedback signal, and the content server state transitions induced by this signal. It shows that as soon as the feedback signal indicates piece starvation (*i.e.*, availability is less than 1), the content server switches to the serving state and feeds the swarm. Piece availability is zero when the swarm bootstraps, and drops whenever clients holding the unique copy of a particular piece depart from the system. The content server switches from the idle state to the serving state only when necessary to restore piece availability to 1. Note that in this experiment we have purposefully created an extreme case of swarm dynamics: in a real swarm, user behavior is not as synchronous.

5.3. Live Internet Experiments

In the set of experiments we present in this Section, we do not control the client arrival and departure processes, but rather we let these processes reflect the popularity of the content we disseminate. Furthermore, clients participating in our swarms exhibit realistic uplink and downlink capacities, unlike our PlanetLab experiments in which all clients have the same uplink

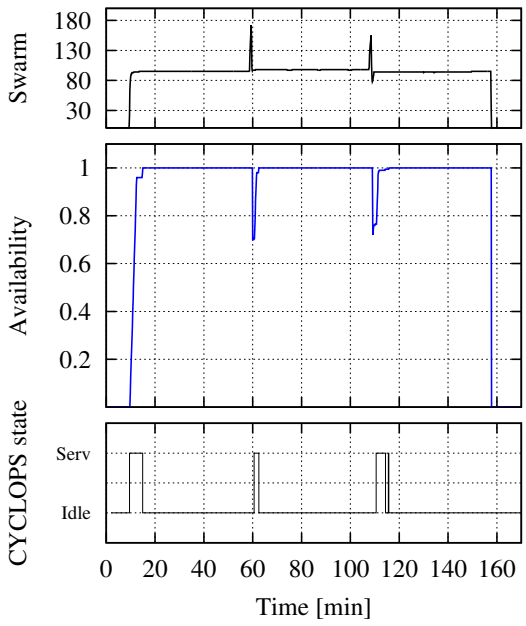

Figure 5: **Waves of arrivals**: availability over time.

capacity.

For Cyclops, out of a total of 7633 users we tracked, 3509 obtained the full content. All other users departed before finishing the download process. For the open-loop-BT content server, 2486 out of a total of 5044 users completed the content download. Despite the diversity in the user base for each experiment, we stress the relevance of a live Internet content distribution: our goal is to show that Cyclops copes well with an heterogeneous environment, with un-controlled clients and that it offers measurable benefits in an illustrative cloud-based scenario.

Figure 6 depicts the instantaneous number of users for both swarms. In our experiments, after the transients of the first few hours have subsided, the user arrival and departure rates within each swarm equalized, with approxi-

mately 35-40 users joining each swarm per minute.

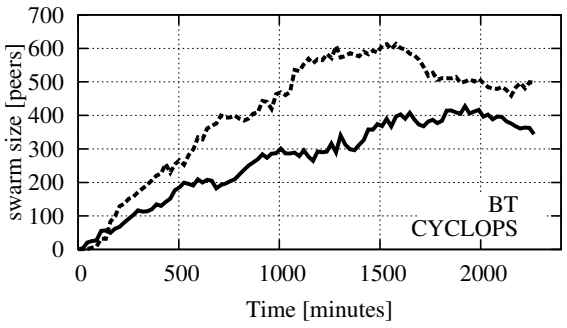

Figure 6: **Live Experiment:** Swarm size over time.

Figure 7 shows the content delivery times (with the most important percentiles) achieved by all users that were able to complete the download. These results indicate that, in our experiments, the median delivery time achieved by both content servers is very similar. For the CYCLOPS content server, the ECDF indicates longer tails: this is mainly due to a larger swarm size, which included clients with poor Internet connectivity. From the end-users' perspective, the difference in the download performance when they are served by CYCLOPS or by open-loop-BT is negligible.

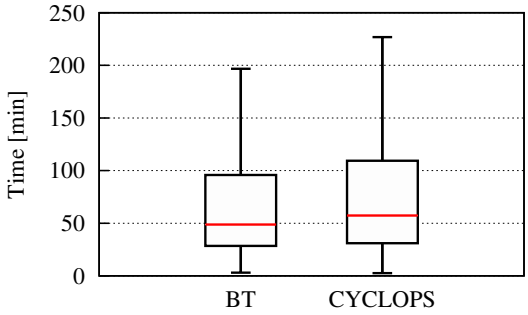

Figure 7: **Live Experiment:** content download times (file size: 357.5 MB).

The content server bandwidth utilization, the associated volume of data and related costs supported by content servers underscore the superiority of Cyclops. Table 3 indicates that the Cyclops content server served a total of 731.6 MB of content data, while the open-loop-BT seed injected a whopping 133.03 GB of content data! Table 3 also reports the overhead traffic, as defined in the previous Section.

These results support our conclusion that the framework discussed in Section 2 and the particular instance we presented in this work are viable candidates for real Internet content distribution systems. Note that both experiments lasted 38 hours, and that the swarm sizes allowed us to assume equivalent uplink capacity distributions for users in each torrent.

Since we deployed our content servers on Amazon EC2 instances, we were able to quantify the economic value of our proposed scheme: For the experiment we carried out, the total cost (including overheads) for distributing the same content when using a legacy BT seed is roughly 180 times higher that of a Cyclops content server.

6. Additional Considerations

We now discuss several points that complement the work presented in this paper.

Dealing with alternative objectives and feedback signals: The framework proposed in this work is general enough to allow many possible combinations of objectives and feedback signals. For example, an alternative objective may be to ensure some minimal level of service based on a feedback regarding the *average* content delivery time. The swarm monitor described

Table 3: **Live Experiment:** service statistics (file size: 357.5 MB)

	BT	CYCLOPS
Total number of users observed in the swarm	5044	7633
Normalized server load	381.04	2.05
Outgoing overhead	6.5 MB	0.2 MB
Incoming overhead	160.8 MB	24.6 MB
Cost of delivery	$ 23.73	$ 0.13

in Section 3 can readily measure the average content delivery times, using the same swarm signaling traffic we discussed earlier. Indeed, clients advertise whenever they receive a new content piece, information that can be simply used to compute the average download rate of the swarm. Based on this information, the content server can choose the appropriate level of bandwidth (*i.e.*, the cost it incurs) to complement the serving capacity λ of the swarm, with the constraint of remaining in the vicinity of the quiescent operating point discussed in Section 2. With reference to Fig. 1, this approach corresponds to a content server selecting to contribute bandwidth resources that move across the various operating regions obtained for different values of λ.

Dealing with alternative ways to collect feedback signals: The swarm monitor described in Section 3 is achieved using a set of OF nodes

that connect to all users. We show in Section 5 that the cost of this solution, in terms of overheads, is not significant. Nevertheless, maintaining many connections may pose some challenges. An alternative solution is to use periodic sampling of the swarm state: The OF nodes, instead of connecting to all the users in the swarm, periodically obtain a subset of users from the tracker and connect temporarily to this subset to collect the information about pieces owned by the users. Using sampling statistics, it is possible to *infer* system-wide piece availability, subject to preset levels of confidence. Clearly, the larger the sampling set, the more precise the availability information: In practice, approximating data availability may yield higher server load, since pieces may not be detected even if they are in the swarm.

Dealing with multiple content servers: In this paper, we conducted experiments in which a single content server is deployed. There are many obvious reasons to consider a more general scenario involving multiple content servers. For example, a CDN operator may wish to use Cyclops on edge servers positioned in multiple locations so as to serve clients efficiently: in this scenario, end-users might be directed to their geographically closest Cyclops content server. Traffic locality to mitigate the impact on ISPs economics, calls for a technique to create distinct swarms. This can be achieved with techniques proposed in the literature without requiring any modification to the design of Cyclops. Alternatively, multiple Cyclops servers could be combined to contribute to the same swarm. In this case, such content servers would have to coordinate what content pieces they serve and when to avoid inefficiencies. Our current implementation does not have provisions for avoiding the overlap between the *serving sets* compiled by different content

servers. That said, standard distributed algorithms could be easily used to manage such situations for production-scale systems.

Dealing with Cloud services cost models: In this work we have illustrated the benefits, in terms of bandwidth consumption, of our approach for an instance of a "cloud-based" content delivery service. Despite CYCLOPS is an appropriate approach to general peer-assisted content delivery networks, here we discuss more on cloud-based services. Since we focused on a simple objective, *i.e.,* to keep a swarm alive, CYCLOPS exhibits an *intermittent behavior*: the system leaves the idle state only when the swarm risks starvation. Under this operational mode, it is natural to question whether this behavior matches current cost models that apply to resources rented in the Cloud. For example, the granularity for paying an Amazon's Elastic Computing Cloud (EC2) instance is one hour. As such, although bandwidth resources are not used nor payed for when CYCLOPS is in the idle state (probing traffic aside), the virtual machine is payed independently of the bandwidth consumption.

With respect to the above discussion, we stress that the experimental setting we used in this work is not intended to be deployed in production. Ideally, our system is suitable for a deployment with the Amazon Simple Storage Service (S3) and its CDN extension (called CloudFront). Alternatively, our approach is suitable for more elaborate scenarios in which multiple contents are distributed via multiple instances of CYCLOPS that coexist in the same virtual machine. In such case, the unit cost of the virtual machine can be amortized by having CYCLOPS content servers coordinate: when one CYCLOPS instance is idle, another instance can be in the serving state, if necessary. Note that such coordination is not trivial: the intermittent behavior

of CYCLOPS is a result of swarm dynamics, which cannot be controlled. It is outside the scope of this work to extend the CYCLOPS architecture to cope with such additional complexity.

For the cost of distributing the content, it is worth noticing that the proposed evaluation does not take into account possible volume discounts, which may decrease the difference between a pure CDN and our solution. In any case, we believe that the impact of this aspect should be minimal.

Dealing with adversarial workloads: Denial of Service (DoS) attacks as well as other improper behavior of end-users aiming to exploit swarm resources is a concern that has to be considered when embracing a peer-assisted CDN solution such as ours. Although this is an important problem to address, here we focus on deliberate attacks by a client (or a set of colluding clients) targeting the specifics of our CYCLOPS framework. Other types of attacks typical of P2P systems, such as Sybil or Eclipse attacks, can be solved using the techniques already presented in the literature[23]. We recognize two possible adversarial exploits, where the aim is to pollute the feedback signal computed by the CYCLOPS swarm monitor.

In the first, an adversary may seek to consume as much server bandwidth as possible. This can be done by inducing the content server to detect piece starvation (when none truly exists), thus causing the server to wastefully inject content. Since CYCLOPS swarm monitor tracks *all* clients in a swarm, such an attack would require a colluding set of malicious users of a size approximately equal to the whole swarm size, which can be safely assumed impractical.

In the second, a set of colluding users may engage in a DoS-like attack to

hinder content distribution, by inducing the content server to conclude that the swarm is healthy (when the contrary is true). This causes starvation of legitimate clients. This can be solved by letting the swarm monitor to compute the average download rate of the swarm. Based on this information, in case of content starvation, the swarm monitor may trigger an alarm, indicating, for instance, the less replicated pieces.

An alternative issue related to our scheme is the single point of failure represented by the server: if the server becomes unavailable, the content may disappear. In this case, any solution based on server replication (e.g., in a hot-standby configuration, or with multiple online servers as discussed before) would be sufficient to solve this problem.

7. Related work

Peer-Assistance: Peer assisted content distribution have been the subject of many recent studies. Of these, the work of Huang, Wang, and Ross [12] could be seen as similar in nature to the work presented in this paper. In that work, the authors advocate the use of peer-assisted content distribution by evaluating the potential gain from peer-assisted video distribution using real-world traces of two large CDN companies, Akamai and Limelight (the underlying architecture of both of which they characterized). Their approach uses the model in [11] to obtain bounds on the server load and download times, should swarming among end-users be allowed. They also quantify the potential reduction in ISP peering traffic, resulting from traffic localization. In the same vein, our work is based on an analytical model that gives key insights as to the benefits of peer-assisted content distribution

(although, our focus is on bulk as opposed to video transfers). Beyond a "proof of concept" using a tractable mathematical formulation, we go one step further by presenting practical feedback-control content injection policies that aim to satisfy performance objectives while minimizing provider's costs. Our implementation is evaluated in realistic contexts, and our results go beyond a purely theoretic estimation of the benefits of peer-assisted content distribution.

Liu *et al.* [19] propose a peer-assisted file distribution system (FS2You) which has a similar architecture to our system. Besides the fact that FS2You and Cyclops have been designed approximately at the same time, our system is based on BitTorrent protocol, which is readily available to many clients, while FS2You requires the installation of a proprietary client.

Frugal Seeding: To the best of our knowledge, the only work that has a similar objective to ours – in terms of reducing the load/cost on a content source, albeit in a very different setting – is Sanderson and Zappala's work [22]. In that work, once the seed has determined a subset of pieces that should be injected in a swarm, it will satisfy any number of requests for those pieces. As a consequence, their technique does not offer the same level of control on the seed workload as the policies we study in this work. Indeed, we observe that for experiments carried out in similar settings, our content servers inject orders of magnitude less traffic than what was documented in [22]. Additionally, our system does not require any parameter to be empirically set.

Chen *et al.* [7] study the "SuperSeeding" mode introduced by an alternative BT client to help peers with slow Internet connections perform initial

content seeding. The objectives of "SuperSeeding" are different from ours. Moreover, a number of problems due to multiple peers using "SuperSeeding" have been reported. The work in [3] proposes a "Smartseed" policy, which advocates serving just one copy of each piece. Besides the fact that Smartseed does not take into account dynamic scenarios, it requires the modification of clients, while our system involves changes only to the server with no modification to the client.

Models and Bounds: The literature is rich with analytical models that dissect many aspects of P2P content distribution. In [14] and [24], the authors derive lower bounds for the minimum content distribution time of a swarm-based P2P application: we build upon those works, but focus instead on the relation between the content server upload rate and the download rate achieved by peers. The work in [21] belongs to the family of fluid models of BitTorrent-like applications: however, in this model it is the number of peers (as opposed to traffic) in the system that is taken as fluid. The authors in [21] develop a differential equation for the fluid model, from which they determine the performance of the dynamic system. We also model content replication in a dynamic setting, but instead consider the number of piece replicas as the dynamic variable modeled using a Markov process.

Bandwidth Allocation in P2P Systems: While the study of alternative mechanisms that improve the bandwidth allocation in P2P systems is orthogonal to our work, results from such studies could clearly have positive implications on content server utilization. In [20], the authors design a content distribution system with the objective of maximizing the download rate of all participants in a managed swarm. The system design in [20] is

based on a wire protocol that induces peer participation (using virtual currency) to achieve a global system optimization. In our work, we focus on a different objective: we try and address the question of whether it is possible to optimize the bandwidth utilization by content servers, without negatively impacting the performance perceived by clients. We note that the model we use in this work can also explain, though in more general terms, the key intuition behind the Antfarm work [20].

The problem of devising efficient uplink allocation algorithms for swarm-based P2P bulk data transfers is addressed in [15]. Instead of using empirically set parameters, as done in BT, to determine the amount of uplink capacity dedicated to each remote connection, they cast uplink allocation as a fractional knapsack problem, and design a simple heuristic utility function to decide the amount of bandwidth a peer should dedicate to each remote connection. The focus of their work is on a cooperative P2P setting, in which peers are assumed to fully abide to the prescribed algorithms.

8. Conclusion

In this paper, we have demonstrated that peer-assisted content distribution could be leveraged to *supplant* as opposed to *supplement* the content provider's resources for purposes of efficient and scalable content distribution, *without* negatively impacting the performance perceived by clients. Our approach is based on a feedback-controlled swarm feeding mechanism, which we have modeled analytically and evaluated empirically using Cyclops – a full-fledged service that we have implemented and deployed both in controlled and un-controlled environments.

Our extensive experimental results – including the *live* distribution of content to thousands of real Internet users – show that CYCLOPS achieves enormous cost savings for the content provider (as high as two orders of magnitude when compared to non-feedback-controlled BitTorrent-based services) without noticeably impacting the performance perceived by end-users. We were able to show that the mechanisms we developed as part of this work have a clear impact on content distribution economics, including significant reduction of costs for content providers, and much more efficient resource utilization for content hosts and distributors.

Our on-going work is focused on exploring alternative objectives and alternative feedback signaling processes in CYCLOPS, as well as extensions that take into account multiple (possibly competing) content servers involved in the distribution of content from multiple sources.

Appendix A.

In this Section we develop a model that relates bandwidth utilization by a content server in the CDN to the average delivery time perceived by a set of swarming users (clients).

Appendix A.1. Model

We consider a dynamic environment, where clients join a swarm, download the content, and eventually leave the system. The number of clients in the swarm is not known *a priori*, but it can be characterized by arrival and departure rates. These rates may fluctuate drastically and such fluctuations are typical for "hot" viral Internet content, which gets published, gains significant popularity fairly quickly, but eventually dies off over time. In this

work, we assume that for the content download timescale (say minutes) they remain constant, allowing the system to reach a steady state in which the arrival and departure rates equalize, and consequently the average number of clients in the swarm is constant.

Let N be the steady-state average number of clients in the swarm, and let the content be divided into M independent pieces. If $M \gg 1$ then a client holds $M/2$ pieces on average. For analytical tractability, we do not model network bottlenecks or losses.

Consider a *birth-death Markov chain* whose state s_k represents k, the number of replicas of a single (arbitrary) piece of content. Note that one can envision an identical, independently evolving Markov chain for each one of the M pieces that make up the content. For a generic state s_k, there are two possible transitions: (1) either the piece is replicated, resulting in a *piece birth*, and thus a transition from state s_k to state s_{k+1}, or (2) a client holding a replica of the piece leaves the swarm and is replaced by a new client that does not have the piece, resulting in a *piece death*, and thus a transition from state s_k to state s_{k-1}.

Let α_k indicate the *average* rate at which the content server injects a piece in the swarm at state s_k. Let λ denote the piece replenishment rate resulting from client contributions: λ is computed by dividing the aggregate upload capacity of all N clients by the total number of pieces M. Both α_k and λ are expressed in pieces per second.

For sake of simplicity, we assume a *random* piece replication strategy: in contrast to more sophisticated replication strategies[4], random piece selection simplifies analysis and provides conservative performance bounds. Thus,

the probability of choosing to replicate the particular piece (modeled by the Markov chain) out of the $M/2$ pieces available at the client, is $2/M$. The probability that no client will choose to replicate that piece is $(1-2/M)^k$, since k is the number of clients holding the piece in state s_k. This yields a probability of $1-(1-2/M)^k$ for going from state s_k to state s_{k+1}.

To compute the transition rate from state s_k to state s_{k+1} we must also account for the rate α_k at which the content server *independently* injects the piece into the swarm. This yields a transition rate of $\lambda\cdot(1-(1-2/M)^k)+\alpha_k$. Notice that state s_0 is a special state in which only the content server can inject the piece. Thus, the transition rate from state s_0 to state s_1 is equal to the server upload rate α_0.

Let μ denote the client departure rate (measured in clients per second). The probability of a death out of state s_k is the probability that any one of the k clients holding the piece leaves the swarm. The probability that a given departure is by one of these k users is k/N. Thus, the transition rate from state s_k to state s_{k-1} is given by $\mu k/N$.

In summary, the transition rates from state s_k to state $s_{k'}$, denoted by $s_{k,k'}$, can be expressed as follows:

$$s_{k,k'}=\begin{cases}\alpha_0 & \text{if } k=0 \text{ and } k'=1\\ \lambda\cdot(1-(1-2/M)^k)+\alpha_k & \text{if } k'=k+1,\ 0<k<N\\ \mu k/N & \text{if } k'=k-1,\ 0<k\leq N\\ 0 & \text{otherwise}\end{cases} \tag{A.1}$$

Note that, since the Markov chain is finite ($N+1$ states), the steady state solution exists.

We now compute the probability π_0 to be in state s_0. For simplicity, we consider the case in which the content server uploads a piece at an average rate $\alpha_k = \alpha$, $\forall k$, irrespectively of its state; by solving the Markov chain we get:

$$\pi_0 = \left[1 + \frac{\alpha}{\mu} N \left(1 + \Phi\right)\right]^{-1} \tag{A.2}$$

where

$$\Phi = \sum_{k=2}^{N} \left(\frac{N}{\mu}\right)^{k-1} \frac{1}{k!} \prod_{i=1}^{k-1} \left[\lambda \left(1 - \left(1 - \frac{2}{M}\right)^{i}\right) + \alpha\right]$$

We now proceed to finding the relationship between the average server rate α and the mean download time. Each client obtains $1/N$ of the swarm's upload capacity, which is $M(\lambda + \alpha)$. Since the content is composed of M pieces, the mean download time can be computed as $T = M/(M(\lambda+\alpha)/N) = N/(\lambda+\alpha)$. This is true as long as the probability of being in state s_0 is small enough. If this probability increases, then we have an additional term for the mean time spent in state s_0: this can be computed by multiplying the probability of state s_0 (π_0) by the time spent in state s_0 ($1/\alpha$). Hence, the mean download time is bounded by:

$$T \leq \frac{N}{\lambda + \alpha} + \frac{\pi_0}{\alpha}, \tag{A.3}$$

To illustrate the utility of this model, consider a swarm of $N = 100$ clients downloading content consisting of $M = 2000$ pieces, with a client departure rate of $\mu = 0.5$ clients per second, and a mean client upload rate of $\lambda = \{1.5\mu, 3\mu, 6\mu\}$ pieces per second. Figure 1 in Section 2 shows the average download time as a function of the server upload rate, as predicated by Equation A.3.

For the particular settings used in Figure 1, the intermediate region is given by $\alpha \in [10, 1000]$ piece/sec.

References

[1] Akamai, `http://www.akamai.com`, 2011.

[2] Amazon AWS, `http://aws.amazon.com`, 2011.

[3] A.R. Bharambe, C. Herley, V.N. Padmanabhan, Analyzing and improving a bittorrent networks performance mechanisms, in: Proc. of IEEE INFOCOM.

[4] F. Bin, D.M. Chiu, J.C. Lui, Stochastic analysis and file availability enhancement for bt-like file sharing systems, in: Proc. of IEEE IWQoS.

[5] BitTorrent, `http://www.bittorrent.com`, 2011.

[6] BitTorrent Protocol, `http://en.wikipedia.org/wiki/BitTorrent_(protocol)`, 2011.

[7] Z. Chen, Y. Chen, C. Lin, V. Nivargi, P. Cao, Experimental analysis of super-seeding in bittorrent, in: Proc. of IEEE ICC.

[8] D.R. Choffnes, F.E. Bustamante, Taming the torrent: A practical approach to reducing cross-isp traffic in p2p systems, in: Proc. of ACM SIGCOMM.

[9] R. Cuevas, N. Laoutaris, X. Yang, G. Siganos, P. Rodriguez, Deep Diving into BitTorrent Locality, Technical Report, `arxiv.org/abs/0907.3874`, Telefonica Research, 2009.

[10] Cyclops, http://www.eurecom.fr/~michiard/downloads/cyclops.tar.gz, 2011.

[11] C. Huang, J. Li, K. Ross, Can internet vod be profitable?, in: Proc. of ACM SIGCOMM.

[12] C. Huang, J. Li, A. Wang, K. Ross, Understanding hybrid cdn-p2p: Why limelight needs its own red swoosh, in: Proc. of ACM NOSSDAV.

[13] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, D. Lewin, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web, in: Proc. of ACM STOC.

[14] R. Kumar, K. Ross, Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems, in: Proc. of IEEE HOTWEB.

[15] N. Laoutaris, D. Carra, P. Michardi, Uplink allocation beyond choke/unchoke or how to divide and conquer best, in: Proc. of ACM CONEXT.

[16] A. Legout, G. Urvoy-Keller, P. Michiardi, Rarest first and choke are enough, in: Proc. of ACM IMC.

[17] Limelight, http://www.limelightnetworks.com, 2011.

[18] F. Liu, Y. Sun, B. Li, B. Li, Quota: Rationing server resources in peer-assisted online hosting systems, in: Proc. of IEEE ICNP.

[19] F. Liu, Y. Sun, B. Li, B. Li, X. Zhang, FS2You: Peer-assisted semi-persistent online hosting at a large scale, IEEE Transactions on Parallel and Distributed Systems 21 (2010) 1442–1457.

[20] R.S. Peterson, E.G. Sirer, Antfarm: Efficient content distribution with managed swarms, in: Proc. of USENIX NSDI.

[21] D. Qiu, R. Srikant, Modeling and performance analysis of bittorrent-like peer-to-peer networks, in: Proc. of ACM SIGCOMM.

[22] B. Sanderson, D. Zappala, Reducing source load in bittorrent, in: Proc. of IEEE ICCCN.

[23] M. Steiner, E.W. Biersack, T. En-Najjary, Exploiting kad: Possible uses and misuses, Computer Communication Review 37 (2007).

[24] R. Sweha, A. Bestavros, J. Byers, Angels – In-Network Support for Minimum Distribution Time in P2P Overlays, Technical Report BUCS-TR-2009-003, Boston University, 2009.