CLive: Cloud-Assisted P2P Live Streaming

Amir H. Payberah[†], Hanna Kavalionak[‡], Vimalkumar Kumaresan[‡], Alberto Montresor[‡], Seif Haridi[†]
[†]Swedish Institute of Computer Science (SICS), Sweden
[‡]University of Trento, Italy
[†]{amir,seif}@sics.se [‡]{name.family}@unitn.it

***Abstract*—Peer-to-peer (P2P) video streaming is an emerging technology that reduces the barrier to stream live events over the Internet. Unfortunately, satisfying soft real-time constraints on the delay between the generation of the stream and its actual delivery to users is still a challenging problem. Bottlenecks in the available upload bandwidth, both at the media source and inside the overlay network, may limit the quality of service (QoS) experienced by users. A potential solution for this problem is *assisting* the P2P streaming network by a cloud computing infrastructure to guarantee a minimum level of QoS. In such approach, rented cloud resources (*helpers*) are added on demand to the overlay, to increase the amount of total available bandwidth and the probability of receiving the video on time. Hence, the problem to be solved becomes minimizing the economical cost, provided that a set of constraints on QoS is satisfied. The main contribution of this paper is CLIVE, a cloud-assisted P2P live streaming system that demonstrates the feasibility of these ideas. CLIVE estimates the available capacity in the system through a gossip-based aggregation protocol and provisions the required resources from the cloud to guarantee a given level of QoS at low cost. We perform extensive simulations and evaluate CLIVE using large-scale experiments under dynamic realistic settings.**

I. INTRODUCTION

Peer-to-peer (P2P) live streaming is becoming an increasingly popular technology, with a large number of academic [1], [2], [3], [4], [5] and commercial [6], [7] products being designed and deployed.

In such systems, one of the main challenges is to provide a good quality of service (QoS) in spite of the dynamic behavior of the network. For live streaming, QoS means high *playback continuity* and short *playback delay*. There is a trade-off between these two properties: it is possible to increase the playback continuity by adopting larger stream buffers, but at the expense of delay. On the other hand, improving playback delay requires that no bottlenecks are present in either the upload bandwidth of the media source and the aggregated upload bandwidth of all peers in the *swarm*, i.e., the peers forming the P2P streaming overlay [8], [9].

Increasing the bandwidth at the media source is not always an option, and even when possible, bottlenecks in the swarm have proven to be much more disruptive [8]. An interesting approach to solve this issue is the addition of auxiliary *helpers* to accelerate the content propagation. A helper could be an *active* computational node that participates in the streaming protocol, or it could be a *passive* storage service that just provides content on demand. The helpers increase the total upload bandwidth available in the system, thus, potentially reducing the playback delay. Both types of helpers could be rented on demand from an IaaS (Infrastructure as a Service) cloud provider, e.g., Amazon AWS. Considering the capacity and the cost of helpers, the problem consists in selecting the right type of helpers (passive vs. active), and provisioning their number with respect to the dynamic behavior of the users. If too few helpers are present, it could be impossible to achieve the desired level of QoS. On the other hand, renting helpers is costly, and their number should be minimized.

This P2P-cloud hybrid approach, termed *cloud-assisted* P2P computing, has already been pursued by a number of P2P content distribution systems. For example, CLOUDANGEL [9] dynamically places active helpers in the swarm to optimize data delivery, and CLOUDCAST [10] employs a single passive helper and enforces strict limits on the number of (costly) interactions with it that originate from peers. However, adapting the cloud-assisted approach to P2P live streaming is still an open issue. Live streaming differs from the content distribution for its soft real-time constraints and a higher dynamism in the network, as the users may be zapping between several channels and start or stop to watch a video at anytime [11], [12].

The contribution of this paper is the design and evaluation of CLIVE, a novel cloud-assisted P2P live streaming system that guarantees a predefined QoS level by dynamically renting helpers from a cloud infrastructure. We model our problem as an optimization problem, where the constraints are given by the desired QoS level, while the objective function is to minimize the total economic cost incurred in renting resources from the cloud. We provide an approximate, on-line solution that is (i) adaptive to dynamic networks and (ii) decentralized.

CLIVE extends existing *mesh-pull* P2P overlay networks for video streaming [2], [5], [13], in which each peer in the swarm periodically sends its data availability to other peers, which in turn pull the required chunks of video from the neighbors that have them. The swarm is paired with a CLIVE *manager* (CM), which participates with other peers in a gossip-based aggregation protocol [14], [15] to find out the current state of the swarm. Using the collected information in the aggregation protocol, the CM computes the number of active helpers required to guarantee the desired QoS. CLIVE includes also a passive helper, whose task is to provide a last resort for peers that have not been able to obtain their video chunks through the swarm.

A delicate balance between the amount of video chunks obtained from the passive helper and the number of active helpers in the system must be found. Either approaches are associated with an economical cost, that depends on (i) the

running time for active helpers, (ii) the storage space and number of data requests for passive helpers, and (iii) the consumed bandwidth for both.

To demonstrate the feasibility of CLIVE, we performed extensive simulations and evaluate our system using large-scale experiments under dynamic realistic settings. We show that we can save up to 45% of the cost by choosing the right number of active helpers compared to only using a passive helper to guarantee the predefined QoS.

II. System model and problem definition

We consider a network consisting of a dynamic collection of *nodes* that communicate through message exchanges. Nodes could be *peers*, i.e., edge computers belonging to users watching the video stream, *helpers*, i.e., computational and storage resources rented from an IaaS cloud, and the *media source* (*source* for short), which generates the video stream and starts its dissemination towards peers.

Each peer is uniquely identified by an ID, e.g., composed by IP address and port, required to communicate with it. We use the term *swarm* to refer to the collection of all peers. The swarm forms an *overlay network*, meaning that each peer connects to a subset of nodes in the swarm (called *neighbors*). The swarm is highly dynamic: new peers may join at any time, and existing peers may voluntarily leave or crash. Byzantine behavior is not considered in this work.

There are two types of helpers: (i) an *active helper* (AH) is an autonomous virtual machine composed of one or more computing cores, volatile memory and permanent storage, e.g., Amazon EC2, and (ii) a *passive helper* (PH) is a simple storage service that can be used to store (PUT) and retrieve (GET) arbitrary pieces of data, e.g., Amazon S3. We assume that customers of the cloud service are required to pay for computing time and bandwidth in the case of AHs, and for storage space, bandwidth and the number of PUT/GET requests in the case of PHs. This model follows the Amazon's pricing model [16], [17].

We assume the source generates a constant-rate bitstream and divides it into a number of *chunks*. A chunk c is uniquely identified by the real time $t(c)$ at which is generated. The generation time is used to play chunks in the correct order, as they can be retrieved in any order, independently from previous chunks that may or may not have been downloaded yet.

Peers, helpers and the source are characterized by different bounds on the amount of available download and upload bandwidth. A node can create a bounded number of download connections and accept a bounded number of upload connections over which chunks are downloaded and uploaded. We define the number of *download slots*, $Down(p)$, and *upload slots*, $Up(p)$, of a peer p as its number of download and upload connections, respectively. Thanks to the replication strategies between different data centers currently employed in clouds [18], we assume that the PH has an unbounded number of upload slots and can serve as many requests as it receives. Preliminary experiments using PlanetLab and Amazon Cloudfront show that this assumption holds in practice, as adding as many clients as possible has not saturated the upload bandwidth.

We assume that nodes are approximately synchronized; this is a reasonable assumption, given that some cloud services, like Amazon AWS, are already synchronized and sometimes require the client machines to be synchronized as well.

The goal of CLIVE peers is to play the video with predefined *playback delay* (the time between the generation of the video and its visualization at the peer) and *playback continuity* (the percentage of chunks that are correctly streamed to users). To reach this goal, CLIVE is allowed to rent a PH and/or AHs from the cloud.

Deciding about which and how much resources to rent from the cloud can be modeled as an optimization problem, where the objective function is to minimize the economic cost and the constraints are the following:

1) the maximum playback delay should be less than or equal to T_{delay}, meaning that if a chunk c is generated at time $t(c)$ at the source, no peers will show it after time $t(c) + T_{delay}$;
2) the maximum percentage of missing chunks should be less than or equal to P_{loss}.

Note that different formulations of this problem are possible, such as fixing a limit on the amount of money to be spent and trying to maximize the playback continuity. We believe, however, that a company, willing to stream its videos, should not compromise on the users' experience, but rather exploit peers whenever possible and fall back to the cloud when peers are not sufficient.

III. System architecture

The basic elements forming CLIVE have been already introduced: the media source, a swarm of peers, a single passive helper (PH), and a number of active helpers (AH). Aim of this section is to discuss how a such diverse collection can be organized and managed. We present two architectural models, illustrated in Figures 1 and 2. The *baseline* model (Figure 1) can be described as a P2P streaming protocol, where peers revert to the PH whenever a chunk cannot be retrieved from other peers. The *enhanced* model (Figure 2) builds upon the baseline, by considering AHs and by providing a distributed mechanism to provision their number and appropriately organizing them.

In the rest of the section, we first discuss the baseline model, introducing the underlying P2P video streaming protocol and showing how it can be modified to exploit a PH. Then, we add the AHs into the picture and illustrate the diverse architectural options available when including them.

A. The baseline model

The baseline model can be seen as a P2P streaming service associated with a server – as simply as that. We introduce this model as a baseline for comparison and validation of our enhanced architectural model.

Note that the idea of augmenting a P2P video streaming application by renting cloud resources is general enough to

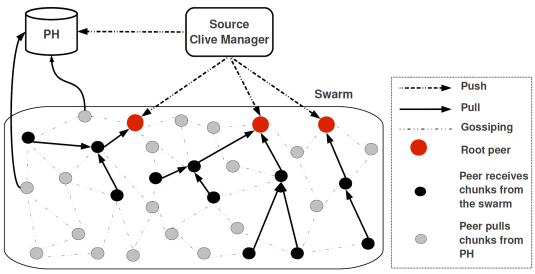


Fig. 1. The baseline model.

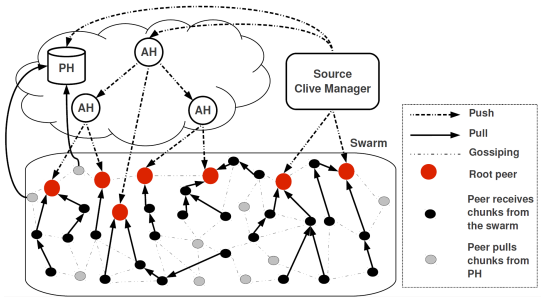


Fig. 2. The enhanced model.

be applied to several existing video streaming applications. We adopt a *mesh-pull* approach for data dissemination [19], meaning that peers are organized in an unstructured overlay and explicitly ask the missing chunks from their neighbors. Peers discover each other using a gossip-based peer-sampling service [20], [21], [22], [23]; then, the random *partial views* created by this service can be used by any of the existing algorithms to build the streaming overlay [2], [4], [24], [25]. In this model, neighboring peers exchange their data availability with each other, and the peers use this information to schedule and pull the required chunks. There are a number of studies [26], [27] on chunk selection policies, but here we use the *in-order* policy, as in COOLSTREAMING [13], where peers pull the missing chunks with the closest playback time first.

The baseline model builds upon this P2P video streaming protocol by adding a PH (Figure 1). The source, apart from pushing newly created video chunks to the swarm, temporary stores them on the PH. In order to guarantee a given level of QoS, each peer is required to have a predefined amount of chunks buffered ahead of its playback time, called *last chance window* (LCW), corresponding to a time interval of length T_{lcw}. If a given chunk has not been obtained from the swarm T_{lcw} time units before the playback time, it is retrieved directly from the PH.

B. The enhanced model

If the P2P substrate does not suffice, the baseline model represents the easiest solution, but as our experiments will show, this solution could be too expensive, as an excessive number of chunks could end up being retrieved directly from the PH. However, even if the aggregate bandwidth of the swarm may be theoretically sufficient to serve all chunks to all peers, the soft real-time constraints on the playback delay may prevent to exploit entirely such bandwidth. No peer must lag behind beyond a specified threshold, meaning that after a given time, chunks will not be disseminated any more. We need to increase the amount of peers that receive chunks in time, and this could be done by increasing the amount of peers that are served as early as possible. The enhanced model pursues this goal by adding a number of AHs to the swarm (Figure 2).

AHs receive chunks from the source or from other AHs, and push them to other AHs and/or to peers in the swarm. To discover such peers, AHs join the peer sampling protocol [23] and obtain a partial view of the whole system. We use a modified version of CYCLON [23], such that peers exchange their number of upload slots along with their ID. AH chooses a subset of *root peers* (Figure 2) from their partial view and establish a connection to them, pushing chunks as soon as they become available. Root peers of an AH are not changed over time, unless they fail or leave the system, or AH finds a peer with more upload slots than the existing root peers. Clearly, a peer could accept to be a root peer only for one AH, to avoid to receive multiple copies of the same chunk. The net effect is an increase in the number of peers that receive the video stream early in time. The root peers also participate in the P2P streaming protocol, serving a number of peers directly or indirectly. PH still exists in the enhanced model to provide chunks upon demand, but it will be used less frequently compared to the baseline model.

Architecturally speaking, an important issue is how to organize multiple AHs and how to feed chunks to them. There are two possible models:

- *Flat*: the AHs receive all their chunks directly from the source and then push them to peers in the swarm, acting just as bandwidth multipliers for the source.
- *Hierarchical*: the AHs are organized in a tree with one AH at the root; the source pushes chunks to the root, which pushes them through the tree.

The advantage of the flat model is that few intermediary nodes cause a limited delay between the source and the peers. However, the source bandwidth could end up being entirely consumed to feed the AHs; and more importantly, any communication to the cloud is billed, including the multiple ones from the source to the AHs. We, thus, decided to adopt the hierarchical model, also considering that communication inside the cloud is (i) extremely fast, given the use of gigabit connections, and (ii) free of charge [28].

One important question in the enhanced model is: *how many AHs to add*? Finding the right balance is difficult; too many AHs may reduce the PH load, but cost too much, given that they are billed hourly and not only per bandwidth. Too few AHs also increases the PH load, and as we show in the experiments, increases the cost. The correct balance dynamically depends on the current number of peers in the swarm, and their upload bandwidth.

The decision on the number of AHs to include in the system is taken by the CLIVE *manager* (CM), a unit that is responsible for monitoring the state of the system and organizing the AHs.

By participating in a decentralized aggregation protocol [14], the CM obtains information about the number of peers in the system and the distribution of upload slots among them. Based on this information, it adds new AHs or remove existing ones, trying to minimize the economic cost. The CM role can be played either directly by the source, or by one AH. A detailed description of the CM is provided in the next section.

IV. The Clive manager

Based on the swarm size and the available upload bandwidth in the swarm, CM computes the number of AHs that have to be active to minimize the economic cost. Then, depending on the current number of AHs, new AHs may be booted or existing AHs may be shutdown.

The theoretical number of AHs that minimize the cost is not so straightforward to compute, because no peer has a global view of the system and its dynamics, e.g., which peers are connected and how many upload slots each peer has. Instead, we describe a heuristic solution, where each peer runs a small collection of gossip-based protocols, with the goal of obtaining approximate aggregate information about the system. CM joins these gossip protocols as well, and collects the aggregated results. It exploits the collected information to estimate a lower bound on the number of peers that can receive a chunk either directly or indirectly from an AH or the source, but not from PH. The CM, then, uses this information to detect whether the current number of AHs is adequate to the current size of the swarm, or if correcting actions are needed by adding/removing AHs.

The participating swarm peers and CM in the gossip-based aggregation protocol collect the following information:

- the current size of the swarm;
- the probability density function of the upload slots available at peers in the swarm.

The rest of this section provides the details about the protocols used to collect the required information and the model used to compute the number of peers reachable from an AH or the source.

The swarm size estimation. The size of the current swarm, N_{swarm}, is computed, with high precision, through an existing aggregation protocol [14]. This information is made available to all peers in the system, including CM that participates in the aggregate computation.

Upload slots estimation. Knowing the number of upload slots of all peers is infeasible, due to the large scale of the system and its dynamism. However, we can obtain a reasonable approximation of the probability density function of the number of upload slots available at all peers.

Assume ω is the actual upload slot distribution among all peers. We adopt ADAM2 [29] to compute $P_\omega : \mathbb{N} \rightarrow \mathbb{R}$, an estimate probability density function of ω. ADAM2 is a gossip-based algorithm that provides an estimation of the cumulative distribution function of a given attribute across all peers. $P_\omega(i)$, then, represents the proportion of peers that have i upload slots w.r.t. the total number of peers, so that $\sum_i P_\omega(i) = 1$. For ADAM2 to work, we assume that each peer is able to estimate its own number of upload slots, and the extreme values of such distribution are known to all.

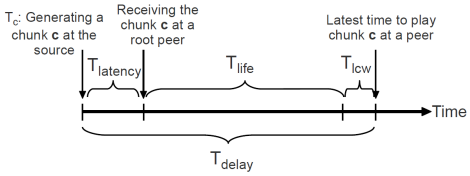


Fig. 3. Live streaming time model.

Chunk lifetime. The number of peers that can receive a chunk from either the swarm, the source or one of the AHs is bounded by the time available to the dissemination process. This time depends on a collection of system and application parameters:

- T_{delay}: No more than T_{delay} time units must pass between the generation of a chunk at the source and its playback at any of the peers.
- $T_{latency}$: The maximum time needed for a newly generated chunk to reach the *root peers*, i.e., the peers directly receive the chunks from AHs or the source, is equal to $T_{latency}$. While this value may depend on whether a particular root peer is connected to the source or to an AH, we consider it as an upper bound and we assume that the latency added by AHs is negligible.
- T_{lcw}: If a chunk is not available at a peer T_{lcw} time units before its playback time, it will be retrieved from the PH.

Therefore, a chunk c generated at time $t(c)$ at the source must be played at peers no later than $t(c) + T_{delay}$, otherwise the QoS contract will be violated. Moreover, the chunk c becomes available at a root peer at time $t(c) + T_{latency}$, and it should be available in the local buffer of any peer in the swarm by time $t(c) + T_{delay} - T_{lcw}$, otherwise the chunk will be downloaded from the PH (Figure 3). This means that the lifetime T_{life} of a chunk from the root peer on is equal to:

$$T_{life} = (T_{delay} - T_{latency}) - T_{lcw} \quad (1)$$

Modeling the dissemination process. Whenever a root peer r receives a chunk c for the first time, it starts disseminating it in the swarm. Biskupski et al. in [30] show that a chunk disseminated by a pull mechanism through a mesh overlay follows a tree-based diffusion pattern. We define the *diffusion tree* $DT(r, c)$ rooted at a root peer r of a chunk c as the set of peers defined as follows: (i) r belongs to $DT(r, c)$, and (ii) q belongs to $DT(r, c)$ if it has received c from a peer $p \in DT(r, c)$.

Learning the exact diffusion tree for all chunks is difficult, because this would imply a global knowledge of the overlay network and its dynamics, and each chunk may follow a different tree. Fortunately, such precise knowledge is not needed. What we would like to know is an estimate of the number of peers that can be theoretically reached through the

Algorithm 1: Lower bound for the diffusion tree size.

```
procedure size(DENSITY P_ω, int depth)
    int min ← +∞;
    repeat k times
        min ← min(min, recSize(P_ω, depth));
    return min;

procedure recSize(DENSITY P_ω, int depth)
    int n ← 1;
    int slots ← random(P_ω);
    for i ← 1 to slots do
        n ← n + recSize(P_ω, depth − 1);
    return n;
```

source or the current population of AHs.

The chunk generation execution is divided into rounds of length T_{round}. Chunk uploaded at round i becomes available for upload to other peers at round $i+1$. The maximum depth, $depth$, of any diffusion tree of a chunk over its T_{life} is computed as: $depth = \lfloor T_{life}/T_{round} \rfloor$. We assume that T_{round} is bigger than the average latency among the peers in the swarm. Given $depth$ and the probability density function P_ω, we define the procedure $\texttt{size}(P_\omega, depth)$ that executes locally at CM and provides an estimate of the number of peers of a single diffusion tree (Algorithm 1). This algorithm emulates a large number of diffusion trees, based on the probability density function P_ω, and returns the smallest value obtained in this way. Emulation of a diffusion tree is obtained by the recursive procedure $\texttt{recSize}(P_\omega, depth)$. In this procedure, variable n is initialized to 1, meaning that this peer belongs to the tree. If the depth of the tree is larger than 0, another round of dissemination can be completed. The number of upload slots is drawn randomly by function $\texttt{random}()$ from the probability density function P_ω. Variable n is then increased by adding the number of peers that can be reached by recursive call to $\texttt{recSize}()$, where the depth is decremented by 1 at each step before the next recursion.

At this point, the expected number of the total peers that can receive a chunk directly or indirectly from AHs and the source, but not from PH, N_{exp}, is given by the total number of root peers times the estimated diffusion tree size, $N_{tree} = \texttt{size}(P_\omega, depth)$. The number of root peers is calculated by the sum of the upload slots at the source $Up(s)$ and AHs $Up(h)$, minus the number of slots used to push chunks to the AHs themselves, as well as to the PH, which is equal to the number of AHs plus one. Formally,

$$N_{exp} = \left(Up(s) + \sum_{h \in \mathcal{AH}} Up(h) - (|\mathcal{AH}| + 1) \right) \cdot N_{tree} \quad (2)$$

where $\mathcal{AH}$ is the set of all AHs.

AHs management model. We define the cost C_{ah} of an AH in one round (T_{round}) as the following:

$$C_{ah} = C_{vm} + m \cdot C_{chunk} \quad (3)$$

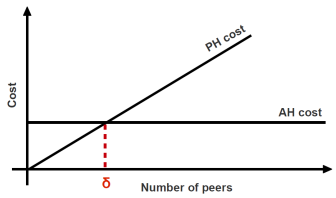


Fig. 4. Calculating the number of peers that is economically reasonable to serve with PH utilization instead to run an additional AH.

where C_{vm} is the cost of running one AH (virtual machine) in a round, C_{chunk} is the cost of transferring one chunk from an AH to a peer, and m in the number of chunks that one AH uploads per round. Since we utilize all the available upload slots of an AH, we can assume that $m = Up(h)$. Similarly, the cost C_{ph} of pulling chunks from PH per round is:

$$C_{ph} = C_{storage} + r \cdot (C_{chunk} + C_{req}) \quad (4)$$

where $C_{storage}$ is the storage cost, C_{req} is the cost of retrieving (GET) one chunk from PH and r is the number of chunks retrieved from PH per round. C_{chunk} of PH is the same as in AH. Moreover, since we store only a few minutes of the live stream in the storage, $C_{storage}$ is negligible.

Figure 4 shows how C_{ah} and C_{ph} (depicted in Formulas 3 and 4) changes in one round (T_{round}), when the number of peers increases. We observe that C_{ph} increases linearly with the number of peers (number of requests), while C_{ah} is constant and independent of the number of peers in the swarm. Therefore, if we find the intersection of the cost functions, i.e., the point δ in Figure 4, we will know when is economically reasonable to add a new AH, instead of putting more load on PH.

$$\delta \approx \frac{C_{vm} + m \cdot C_{chunk}}{C_{chunk} + C_{req}} \quad (5)$$

CM considers the following thresholds and regulation behavior:

- $N_{swarm} > N_{exp} + \delta$: This means that the number of peers in the swarm is larger than the maximum size that can be served with a given configuration, thus, more AHs should be added to the system.
- $N_{swarm} < N_{exp} + \delta - Up(h) \cdot N_{tree}$: Current configuration is able to serve more peers than the current network size, thus, extra AHs can be removed. $Up(h) \cdot N_{tree}$ shows the number of peers served by one AH.
- $N_{exp} + \delta - Up(h) \cdot N_{tree} \leq N_{swarm} \leq N_{exp} + \delta$: In this interval the system has adequate resource and no change in the configuration is required.

CM periodically checks the above conditions, and takes the necessary actions, if any. In order to prevent temporary fluctuation, it adds/removes only single AH in each step.

V. EXPERIMENTS

In this section, we evaluate the performance of CLIVE using KOMPICS [31], a framework for building P2P protocols that

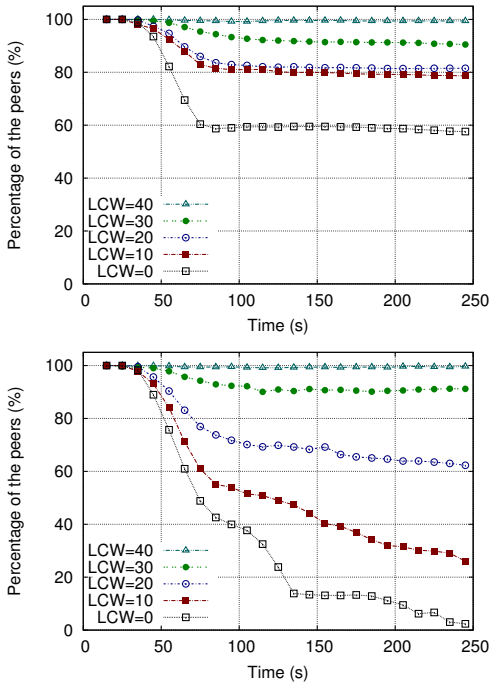


Fig. 5. The percentage of the peers receiving 99% playback continuity with different values of T_{lcw} (measured in number of chunks). Top: join scenario, bottom: churn scenario (1% churn rate).

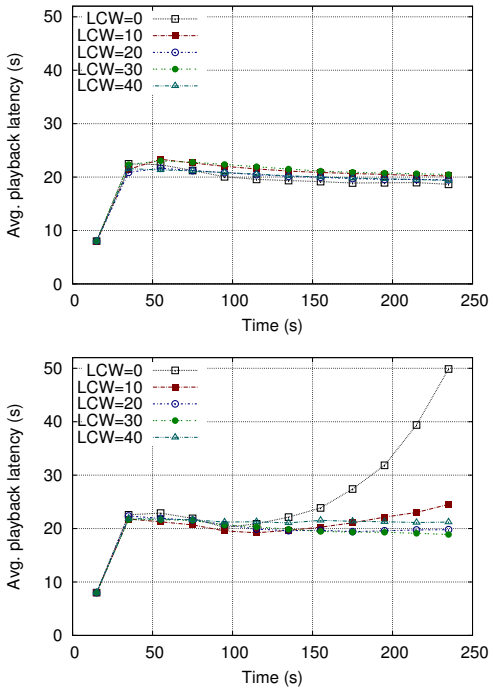


Fig. 6. Average playback latency across peers with different values of T_{lcw} (measured in number of chunks). Top: join scenario, bottom: churn scenario (1% churn rate).

TABLE I
SLOT DISTRIBUTION IN FREERIDER OVERLAY.

Number of slots	Percentage of peers
0	49.3%
1	18.7%
2	8.4%
3-19	5.2%
20	6.8%
Unknown	11.6%

provides a discrete event simulator for testing the protocols using different bandwidth, latency and churn scenarios.

A. Experimental setting

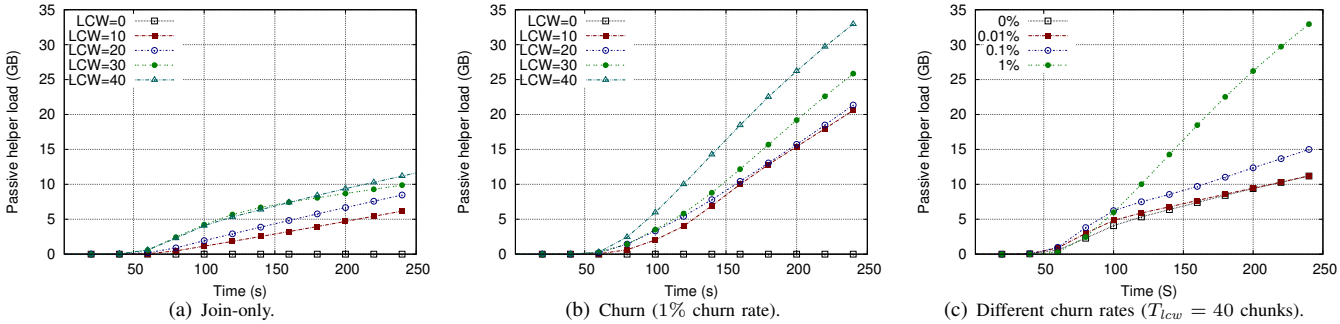
In our experimental setup, we set the streaming rate to 500kbps, which is divided into chunks of 20kb; each chunk, thus, corresponds to 0.04s of video stream. Peers start playing the media after buffering it for 15 seconds, and T_{delay} equals 25 seconds. We set the bandwidth of an upload slot and download slot to 100kbps. Without loss of generality, we assume all peers have enough download bandwidth to receive the stream with the correct rate. In these experiments, all peers have 8 download slots, and we consider three classes of upload slot distributions: (i) *homogeneous*, where all peers have 8 upload slots, (ii) *heterogeneous*, where the number of upload slots in peers is picked uniformly at random from 4 to 13, and (iii) *real trace* (Table I) based on a study of large scale streaming systems [11]. As it is shown in Table I, around 50% of the peers in this model do not contribute in the data distribution. The media source is a single node that pushes chunks to 10 other peers. We assume PH has infinite upload bandwidth, and each AH can push chunks to 20 other peers. Latencies between peers are modeled using a latency map based on the King data-set [32].

In our experiments, we used two failure scenarios: *join-only* and *churn*. In the join-only scenario, 1000 peers join the system following a Poisson distribution with an average inter-arrival time of 10 milliseconds, and after joining the system they will remain till the end of the simulation. In the churn scenario, approximately 0.01%, 0.1% and 1% of the peers leave the system per second and rejoin immediately as newly initialized peers [33]. However, unless stated otherwise, we did the experiments with 1% churn rate to show how the system performs in presence of high dynamism.

B. The effect of T_{lcw} on system performance

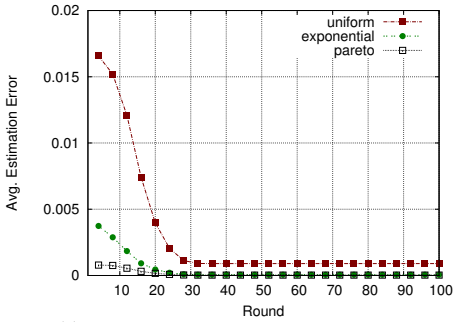
In the first experiment, we evaluate the system behavior with different values for T_{lcw}, measured in number of chunks. In this experiment, we measure *playback continuity* and *playback latency*, which combined together reflect the QoS experienced by the overlay peers. Playback continuity shows the percentage of chunks received on time by peers, and playback latency represents the difference, in seconds, between the playback point of a peer and the source.

For a cleaner observation of the effect of T_{lcw}, we use the homogeneous slot distribution in this experiment. Figure 5 shows the fraction of peers that received 99% of the chunks before their timeout with different T_{lcw} in the join-only and churn scenarios (1% churn rate). We changed T_{lcw} between 0 to 40 chunks, where zero means peers never use PH, and 40

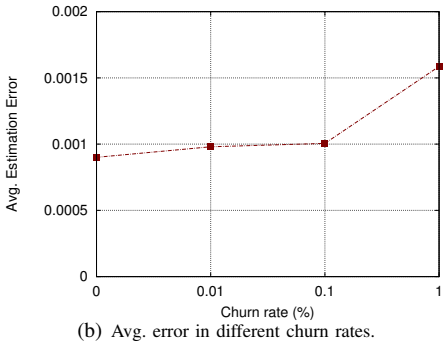


(a) Join-only. (b) Churn (1% churn rate). (c) Different churn rates ($T_{lcw} = 40$ chunks).

Fig. 7. The cumulative PH load with different values of T_{lcw} and churn rates.



(a) Avg. error in different slot distributions.



(b) Avg. error in different churn rates.

Fig. 8. Avg. estimation error.

means that a peer retrieves up to chunk $c+40$ from PH, if the peer is currently playing chunk c. As we see, the bigger T_{lcw} is, the more peers receive chunks in time. Although for any value of $T_{lcw} > 0$ peers try to retrieve the missing chunks from PH, the network latency may not allow to obtain the missing chunk in time. As Figure 5 shows, all the peers retrieve 99% of the chunks on time when $T_{lcw} = 40$. Given that each chunk corresponds to 0.04 seconds, $T_{lcw} = 40$ implies 1.6 seconds.

The average playback latency of peers is shown in Figure 6. In the join-only scenario, playback latency does not depend on T_{lcw}, while in the churn scenario we can see a sharp increase when T_{lcw} is small.

C. PH load in different settings

Here, we measured PH load or the amount of fetched chunks from PH with different T_{lcw} values and churn rates. Figures 7(a) and 7(b) show the cumulative load of PH in the join-only and churn scenarios (1% churn rate), respectively. As we see in these figures, by increasing T_{lcw}, more requests are sent to PH, thus, increasing its load. Figure 7(c) depicts the cumulative PH load over time for four different churn rates and T_{lcw} equals 40 chunks. As the figure shows, there is no big change in PH load under low churn scenarios (0.01% and 0.1%), which are deemed realistic in deployed P2P systems [33]. However, it sharply increases in the presence of higher churn rates (1%), because peers lose their neighbors more often, thus, they cannot pull chunks from the swarm in time, and consequently they have to fetch them from PH.

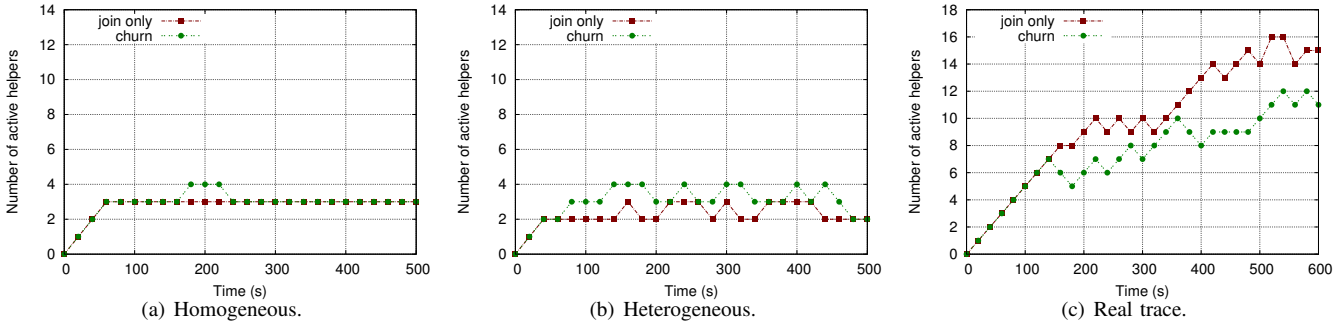
D. Upload slot distribution estimation

In the next experiment, we evaluate the estimation of upload slots distribution in the system. We adopt the Kolmogorov-Smirnov (KS) distance [34], to define the upper bound on the approximation error of any peer in the system. The KS distance is given by the maximum difference between the actual slot distribution, ω, and the estimated slot distribution, $E(\omega)$. We compute $E(\omega)$ based on P_ω for different number of slots. Since the maximum error is determined by a single point (slot) difference between ω and $E(\omega)$, it is sensitive to noise. Hence, we measure the average error at each peer as the average error contributed by all points (slots) in ω and $E(\omega)$. The total average error is then computed as the average of these local average errors.

We consider three slot distributions in this experiment: (i) the uniform distribution, (ii) the exponential distribution ($\lambda = 1.5$), and (iii) the Pareto distribution ($k = 5, x_m = 1$). Figure 8(a) shows the average error in three slot distributions, and Figure 8(b) shows how the accuracy of the estimation changes in different churn rates.

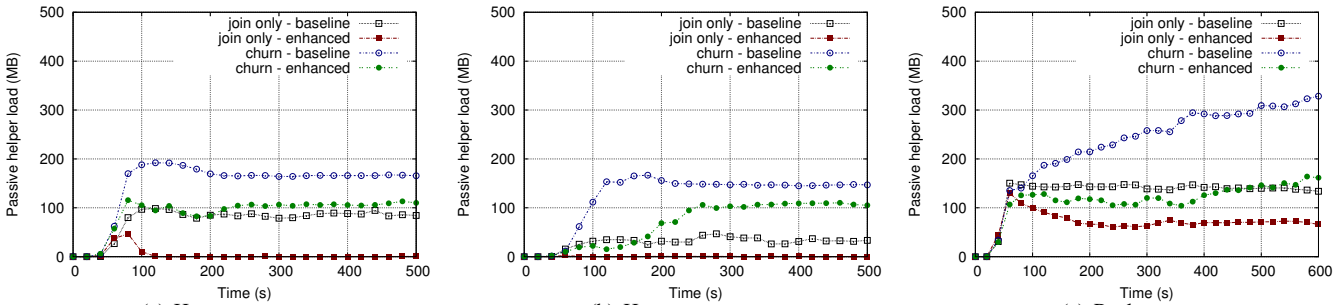
E. Economic cost

In the last experiment, we measure the effect of adding/removing AHs on the total cost. Note, in these experiments we set T_{lcw} to 40 chunks, therefore, regardless of the number of AHs, all the peers receive 99% of the chunks before their playback time. In fact, AHs only affect the total cost of the service. In Section IV, we showed how CM estimates the required number of AHs. Figure 9 depicts how the number



(a) Homogeneous. (b) Heterogeneous. (c) Real trace.

Fig. 9. Number of AHs in different settings and scenarios.



(a) Homogeneous. (b) Heterogeneous. (c) Real trace.

Fig. 10. PH load in different scenarios with dynamic changes of the number of AHs.

of AHs changes over time. In the join-only scenario and the homogeneous slot distribution (Figure 9(a)), the CM estimates the exact value of the peers that receive the chunks on time using the existing resources in the system, and consequently the exact number of required AHs. Hence, as it is shown, the number of AHs will be fixed during the simulation time. However, in the heterogeneous and real trace slot distributions (Figures 9(b) and 9(c)), CM estimation changes over time, and based on this, it adds and removes AHs. In the churn scenario (1% churn rate), CM estimation also changes over the time, thus, the number of AHs fluctuates.

Relatively, we see how PH load changes in different scenarios in the baseline and enhanced models (Figure 10). Figure 9(a) shows that three AHs are added to the system in the join-only scenario and the homogeneous slot distribution. On the other hand, we see in Figure 10(a), in the join-only scenario, with the help of these three AHs (enhanced model), the load of PH goes down nearly to zero. It implies that three AHs in the system are enough to minimize PH load, while preserving the promised level of QoS. Hence, adding more than three AHs in this setting does not have any benefit and only increases the total cost. Moreover, we can see in the join-only scenario, if there is no AH in the system (baseline model), PH load is much higher than the enhanced model, e.g., around $90mb$, $40mb$, and $130mb$ per second in the homogeneous, heterogeneous, and real trace, respectively. The same difference appears in the churn scenario.

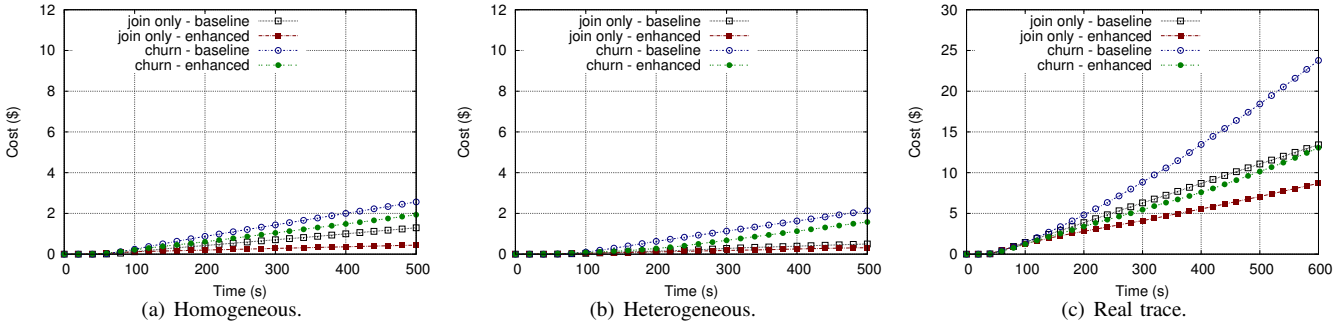
Figure 11 shows the cumulative total cost over the time in different scenarios and slot distributions. In this measurement, we use Amazon S3 as PH and Amazon EC2 as AHs. According to the price list of Amazon [16], [17], the data transfer price of S3 is 0.12$ per GB, for up to 10 TB in a month. The cost of GET requests are 0.01$ per 10000 requests. Similarly, the cost of data transfer in EC2 is 0.12$ per GB, for up to 10 TB in a month, but since the AHs actively push chunks, there is no GET requests cost. The cost of a large instance of EC2 is 0.34$ per hour. Considering the chunk size of $20kb$ ($0.02mb$) in our settings, we can measure the cost of PH in Amazon S3 per round (second) according to the Formula 4:

$$\begin{aligned} C_{ph} &\approx r \cdot (C_{chunk} + C_{req}) \\ &\approx \frac{r \times 0.02 \times 0.12}{1000} + \frac{r \times 0.01}{10000} \end{aligned} \quad (6)$$

where r is the the number of received requests by PH in one round (second). The cost of storage is negligible. Given that each AH pushes chunks to 20 peers with the rate of $500kbps$ ($0.5mbps$), then the cost of running one AHs in Amazon EC2 per second according to Formula 3 is:

$$\begin{aligned} C_{ah} &= C_{vm} + m \cdot C_{chunk} \\ &= \frac{0.34}{3600} + \frac{20 \times 0.5 \times 0.12}{1000} \end{aligned} \quad (7)$$

Figure 11 shows the cumulative total cost for different slot distribution settings. It is clear from these figures that adding AHs to the system reduces the total cost, while keeping the QoS as promised. For example, in the high churn scenario (1% churn rate) and the real trace slot distribution the total



(a) Homogeneous. (b) Heterogeneous. (c) Real trace.

Fig. 11. The cumulative total cost for different setting and scenarios.

cost of system after 600 seconds is 24$ in the absence of AHs (baseline model), while it is close to 13$ if AHs are added (enhanced model), which saves around 45% of the cost.

VI. Related work

A. Content distribution

Although P2P algorithms are emerging as promising solutions for large scale content distribution, they are still subject to a number of challenges [19]. The bottleneck in the aggregated upload bandwidth in the overlay is one of the main problems [8]. One possible approach to increase the upload capacity is to use helpers [35], [36]. The helper role can be played by *idle* [37] or *restricted* [38] users. Idle users are peers with spare upload capacity not interested in any particular data, while restricted users are users with limited rights in the network service.

Another approach suggests to exploit dedicated servers as helpers to accelerate content distribution [9], where the servers cache and forward content to other peers. Montresor and Abeni [10] introduced an alternative way to use dedicated servers. They proposed to merge P2P and cloud storage to support information diffusion, such that the cloud storage participates in the communication as a passive node.

In addition to these solutions, Wu et al. proposed a queuing model in [39] to predict the dynamic demands of the users of a P2P video on demand (VoD) system and provide elastic amounts of computing and bandwidth resources on the fly with a minimum cost. Similarly, Jin et al. present a cloud assisted system architecture for P2P media streaming among mobile peers to minimize energy consumption [40].

Unlike all the described approaches, our work proposes to use the cloud computing and storage resources as a collection of active and passive helpers. The combination of both types of helpers together with an effective resource management, distinguishes our approach from previous work.

B. Self-monitoring and self-configuration systems

Self-monitoring and *self-configuration* mechanisms are essential to manage large, complex and dynamic systems in an effective way. Self-monitoring detects the current states of system components, while self-configuration is aimed to adapt system configuration according to the received information.

Self-monitoring allows the system to have a view on its current use and state. One of the popular approaches for monitoring P2P overlays is decentralized aggregation [14]. For example, ADAM2 [29] presents a gossip-based aggregation protocol to estimate the distribution of attributes across peers. Similarly, Van Renesse and Haridasan [41] propose a distribution estimation mechanism, which can be used when peers are not aware of the extreme value of the distribution.

Self-configuration is the process that autonomously configures components and protocols according to specified target goals, e.g., reliability and availability. To self-tune according to on-going state, the system can use an external component that controls the system either via control loops [42] or in a decentralized way [43], [44], [45]. A relevant example of a self-configuration mechanism is TMAN [43], an overlay topology management that uses a ranking function exploited locally by each peer to choose its neighbors.

Another system, proposed by Kavalionak and Montresor [44], considers a replicated service on top of a mixed P2P and cloud system. This protocol is able to self-regulate the amount of cloud storage resources utilization according to available P2P resources. However, the main goal of the proposed approach is to support a given level of reliability, while in our work we are interested in an effective data dissemination that allows to self-configure the amount of active and passive cloud resources utilization.

VII. Conclusions

The main contribution of this paper is CLIVE, a P2P live streaming system that integrates cloud resources (helpers), whenever the peer resources are not enough to guarantee a predefined QoS with a low cost. Two types of helpers are used in CLIVE, (i) active helper, which is an autonomous virtual machine, e.g., Amazon EC2, that participates in the streaming protocol, and (ii) passive helper, which is a storage service, e.g., Amazon S3, that provides content on demand. CLIVE estimates the available capacity in the system through a gossip-based aggregation protocol and provisions the required resources (passive/active helpers) from the cloud provider w.r.t the dynamic behavior of the users.

We are currently implementing a prototype CLIVE system based on Amazon's services like EC2, S3 and Cloudfront.

In particular, Cloudfront extends S3 by replicating read-only content to a number of *edge locations*, in order to put clients closer to the data and reduce the communication latency. Altogether, these edge locations (currently there are 34 of them around the world) can be seen as a unique PH, from where chunks can be pulled. From the point of view of the source, the interface remain the same: content is originally pushed to S3 and from there is replicated across geographically-dispersed data centers.

REFERENCES

[1] A. H. Payberah, J. Dowling, F. Rahimian, and S. Haridi, "gradienTv: Market-based P2P Live Media Streaming on the Gradient Overlay," in *Proc. of the 10th int. conf. on Distr. App. and Interoperable Syst.* Springer Berlin / Heidelberg, 2010, pp. 212–225.

[2] A. H. Payberah, J. Dowling, and S. Haridi, "Glive: The gradient overlay as a market maker for mesh-based p2p live streaming," in *Proc. of the 10th IEEE Int. Symp. on Parallel and Distr. Comp. (ISPDC)*, 2011.

[3] A. H. Payberah, J. Dowling, F. Rahimian, and S. Haridi, "Sepidar: Incentivized market-based p2p live-streaming on the gradient overlay network," in *Proc. of the Int. Symp. on Multimedia*, Taiwan, 2010.

[4] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "QoE in Pull Based P2P-TV Systems: Overlay Topology Design Tradeoffs," in *Proc. of the IEEE 10th Int. Conf. on Peer-to-Peer Comp.*, 2010.

[5] D. Frey, R. Guerraoui, A. Kermarrec, and M. Monod, "Boosting Gossip for Live Streaming," in *Proc. of the IEEE 10th Int. Conf. on Peer-to-Peer Comp.* IEEE, 2010, pp. 1–10.

[6] Y. Lu, B. Fallica, F. Kuipers, R. Kooij, and P. V. Mieghem, "Assessing the quality of experience of sopcast," *Journal of Internet Protocol Technology*, vol. 4, no. 1, pp. 11–23, 2009.

[7] S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno, "Analysis of pplive through active and passive measurements," in *Proc. of the 2009 IEEE Int. Symp. on Parallel&Distr. Processing*. IEEE, 2009, pp. 1–7.

[8] R. Kumar and K. W. Ross, "Peer-assisted file distribution: The minimum distribution time," *Hot Topics in Web Systems and Technologies*, 2006.

[9] R. Sweha, V. Ishakian, and A. Bestavros, "Angels in the cloud: A peer-assisted bulk-synchronous content distribution service," *Proc. of IEEE Int. Conf. on Cloud Computing*, pp. 97–104, 2011.

[10] A. Montresor and L. Abeni, "Cloudy weather for P2P, with a chance of gossip," in *Proc. of the 11th Conf. on Peer-to-Peer Computing*, 2011.

[11] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. of the conf. on App., technologies, architectures, and protocols for comp. comm.* ACM, 2004, pp. 107–120.

[12] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proc. of the 4th ACM SIGCOMM conf. on Internet measurement*. ACM, 2004, pp. 41–54.

[13] X. Zhang, J. Liu, B. Li, and T. shing Peter Yum, "Coolstreaming/donet: A data-driven overlay network for Peer-to-Peer live media streaming," in *Proc. of the 24th Annual Joint Conf. of the IEEE Comp. and Comm. Societies*. IEEE, 2005.

[14] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, 2005.

[15] A. Montresor and A. Ghodsi, "Towards robust peer counting," in *Proc. of the 9th Int. Conf. on Peer-to-Peer Computing*, Seattle, WA, 2009.

[16] (2012, Feb.) Amazon elastic compute cloud (Amazon EC2). [Online]. Available: http://aws.amazon.com/ec2/

[17] (2012, Feb.) Amazon simple storage service (Amazon S3). [Online]. Available: http://aws.amazon.com/s3/

[18] S. Goel and R. Buyya, "Data replication strategies in wide area distributed systems," Idea Group Inc., Hershey, PA, USA, Tech. Rep., 2006.

[19] W. P. K. Yiu, X. Jin, and S. H. G. Chan, "Challenges and approaches in large-scale P2P media streaming," *IEEE MultiMedia*, vol. 14, no. 2, pp. 50–59, 2007.

[20] J. Dowling and A. H. Payberah, "Shuffling with a croupier: Nat-aware peer-sampling," in *Proc. of the 32nd Int. Conf. on Distr. Comp. Syst.*, 2012.

[21] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *Proc. of the 24th Int. Conf. on Distr. Comp. Syst. (ICDCS'04)*. Washington, DC, USA: IEEE, 2004, pp. 102–109.

[22] A. H. Payberah, J. Dowling, and S. Haridi, "Gozar: Nat-friendly peer sampling with one-hop distributed nat traversal," in *Proc. of the 11th IFIP Int. Conf. on Distr. App. and Interoperable Systems*, 2011.

[23] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[24] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *Proc. of the 27th Conf. on Comp. Comm.* IEEE, 2008.

[25] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. E. Mohr, and E. E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. of the 4th int. conf. on Peer-to-Peer Syst.*, 2005, pp. 127–140.

[26] N. Carlsson and D. L. Eager, "Peer-assisted on-demand streaming of stored media using bittorrent-like protocols," in *Proc. of the 6th int. IFIP-TC6 conf. on Ad Hoc and sensor networks, wireless networks, next generation internet*. Berlin, Heidelberg: Springer-Verlag, 2007.

[27] B. Q. Zhao, J. C.-S. Lui, and D.-M. Chiu, "Exploring the optimal chunk selection policy for data-driven p2p streaming systems," in *Proc. of the 9th Int. Conf. on Peer-to-Peer Computing*, 2009, pp. 271–280.

[28] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, 2010.

[29] J. Sacha, J. Napper, C. Stratan, and G. Pierre, "Adam2: Reliable distribution estimation in decentralised environments," in *Proc. of the 30th IEEE Int. Conf. on Distr. Comp. Syst. (ICDCS'10)*, 2010.

[30] B. Biskupski, M. Schiely, P. Felber, and R. Meier, "Tree-based analysis of mesh overlays for peer-to-peer streaming," in *Proc. of the 8th int. conf. on distr. app. and interoperable systems*. Springer, 2008.

[31] C. Arad, J. Dowling, and S. Haridi, "Developing, simulating, and deploying peer-to-peer systems using the Kompics component model," in *Proc. of the 4th Int. Conf. on COMm. System softWAre and middlewaRE (COMSWARE'09)*. Dublin, Ireland: ACM, 2009, pp. 1–9.

[32] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proc. of the SIGCOMM Internet Measurement Workshop*, 2002.

[33] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. of the 6th Conf. on Internet Measurement (IMC'06)*. Rio de Janeiro, Brazil: ACM, 2006, pp. 189–202.

[34] G. Schay, *Introduction to probability with statistical applications*. Birkhäuser, 2007.

[35] J. Wang and K. Ramchandran, "Enhancing Peer-to-Peer live multicast quality using helpers," in *Proc. of Int. Conf. on Image Processing*, 2008.

[36] H. Zhang, J. Wang, M. Chen, and K. Ramchandran, "Scaling Peer-to-Peer Video-on-Demand Systems Using Helpers," in *Proc. of Int. Conf. on Image Processing*. IEEE, 2009.

[37] J. Wang, C. Yeo, V. Prabhakaran, and K. Ramchandran, "On the role of helpers in Peer-to-Peer file download systems: Design, analysis and simulation," *Proc. of the 6th Int. Workshop on Peer-To-Peer Syst.*, 2007.

[38] R. Kumar and K. W. Ross, "Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems," *Workshop on Hot Topics in Web Systems and Technologies*, 2006.

[39] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. M. Lau, "Cloudmedia: When cloud on demand meets video on demand," in *Proc. of Int. Conf. on Distr. Comp. Syst.*, 2011, pp. 268–277.

[40] X. Jin and Y.-K. Kwok, "Cloud assisted P2P media streaming for bandwidth constrained mobile subscribers," in *Proc. of the 16th Int. Conf. on Parallel and Distr. Syst. (IPDPS'10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 800–805.

[41] M. Haridasan and R. van Renesse, "Gossip-based distribution estimation in peer-to-peer networks," in *Proc. of the 7th Int. Conf. on Peer-to-Peer Systems*. Berkeley, CA, USA: USENIX Association, 2008.

[42] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, pp. 41–50, 2003.

[43] M. Jelasity, A. Montresor, and O. Babaoglu, "T-Man: Gossip-based fast overlay topology construction," *Computer Networks*, vol. 53, no. 13, pp. 2321–2339, 2009.

[44] H. Kavalionak and A. Montresor, "P2P and cloud: A marriage of convenience for replica management," in *Proc. of the 7th IFIP Int. Workshop on Self-Organizing Systems (IWSOS'12)*, ser. Lecture Notes in Computer Science. Delft, The Netherlands: Springer, 2012, pp. 60–71.

[45] O. Babaoglu and M. Jelasity, "Self-* properties through gossiping." *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 366, no. 1881, pp. 3747–3757, 2008.