Hierarchically Clustered P2P Video Streaming: Design, Implementation, and Evaluation

Yang Guo[a,*], Chao Liang[a], Yong Liu[b]

[a] *Alcatel-Lucent, Holmdel, NJ 07733*
[b] *Polytechnic Institute of NYU, Brooklyn, NY 11201*

Abstract

P2P based live streaming has been gaining popularity. The new generation P2P live streaming systems not only attract a large number of viewers, but also support better video quality by streaming the content at higher bit-rate. In this paper, we propose a novel P2P streaming framework, called *Hierarchically Clustered P2P Video Streaming, or HCPS*, that can support the streaming rate approaching the optimal upper bound while accommodating large viewer population. The scalability comes with the hierarchical overlay architecture by grouping peers into clusters and forming a hierarchy among them. Peers are assigned to appropriate cluster so as to balance the bandwidth resources across clusters and maximize the supportable streaming rate. Furthermore, individual peers perform *distributed queue-based scheduling algorithms* to determine how to retrieve data chunks from source and neighboring peers, and how to utilize its uplink bandwidth to serve data chunks to other peers. We show that queue-based scheduling algorithms allow to fully utilize peers' uplink bandwidths, and HCPS supports the streaming rate close to the optimum in practical network environment. The prototype of HCPS is implemented and various design issues/tradeoffs are investigated. Experiments over the PlanetLab further demonstrate the effectiveness of HCPS design.

Keywords: Peer-to-Peer, streaming, hierarchical clustering, optimal scheduling.

*Corresponding author
Email addresses: yang.guo@alcatel-lucent.com (Yang Guo), chao_c.liang@alcatel-lucent.com (Chao Liang), yongliu@poly.edu (Yong Liu)

1. Introduction

Video over Internet has been gaining popularity due to the fast penetration of high-speed Internet access, expanding group of savvy broadband users, and rich video contents available over the Internet. ISPs are aggressively rolling out new infrastructure enhanced with advanced protocols to offer customers Internet TV services. Video traffic is expected to dominate the Internet in near future. Traditionally, video content is streamed to end users either directly from video source servers, or indirectly from edge servers belong to Content Distribution Network (CDN). Peer-to-Peer video streaming has emerged as an alternative with low infrastructure cost. P2P streaming systems, such as CoolStreaming [1], PPLive [2], and SopCast [3], attract millions of users to watch live or on-demand video programs. The existing P2P based systems show remarkable capability of handling large viewer population, and being robust against peer churns and dynamic network environment.

P2P design philosophy seeks to utilize peers' upload bandwidth to reduce server's workload. The maximum video bit-rate that can be serviced over a P2P system is determined by the server's upload capacity and peers' average upload capacity [4]. The recent study [5] suggests that simple scheduling algorithm employed by many P2P systems is unable to fully utilize peers' upload capacity. Higher streaming rate equates better video quality. The capability to support high bit-rate streaming also provides more cushion to absorb the bandwidth variations in case the constant-bit-rate (CBR) video is broadcasted. A new P2P streaming framework that is capable of supporting high video bit-rate and accommodating a large number of users is highly desirable.

Most existing P2P streaming solutions maintain a loosely connected mesh to accommodate a large number of users. Individual peers share data with a small set of neighboring peers. While such random mesh is scalable, the study in [6] suggests that the supportable video bit-rate in a random mesh is throttled by the *content bottleneck*, i.e., a peer's upload bandwidth cannot be utilized if it does not have *fresh* video data for its neighbors. More recently, several intelligent scheduling algorithms capable of fully utilizing peers' upload capacities have been developed [4, 7, 8]. These scheduling algorithms can achieve the maximum streaming rate if peers are connected in a full mesh. The requirement of fully connected mesh, however, confines the system scalability. It is unrealistic to maintain hundreds or thousands of

peering connections on a peer.

In this paper, we propose *Hierarchically Clustered P2P Streaming system (HCPS)* that supports the streaming rate approaching the optimum yet scales to host a large number of peers. P2P overlay topology and distributed chunk scheduling algorithms running at individual peers collectively determine the performance of a P2P streaming system. Accordingly, we address the design challenges following a two-step approach. First, we propose a hierarchically clustered P2P overlay that scales to host a large number of users/peers. In addition, the overlay is designed as such that its *maximum supportable streaming rate*, defined as the maximum streaming rate allowed by a P2P overlay, approaches the optimum upper bound. Second, we develop *distributed queue-based chunk scheduling algorithms* that actually achieves the maximum supportable streaming rate allowed by the HCPS overlay in spite of the large number of users/peers. The main contributions of this paper can be summarized as follows:

- We propose a novel P2P streaming framework that is scalable and supports the streaming rate approaching the optimum upper bound. The optimality is proved analytically and evaluated through experiments.
- The full-fledged prototype is implemented. Various design considerations are explored to handle dynamics in realistic network environments, including peer churns, peer bandwidth variations, and inside network congestion.
- The performance of the prototype system is examined through experiments over PlanetLab [9]. Both the optimality and the adaptiveness of the proposed chunk scheduling method are demonstrated.

The remaining paper is organized as follows. Related work is presented in Section 2. In Section 3, overlay construction of hierarchically clustered P2P video streaming (HCPS) is presented. In Section 4, distributed queue-based chunk scheduling algorithms are described. Section 5 discusses the design issues in implementing HCPS prototype. Section 6 presents the experiment results of HCPS over PlanetLab. Finally, conclusions and discussions are included in Section 7.

2. Related work

P2P technology has become an effective paradigm for building distributed networked applications. P2P based file sharing [10], voice-over-IP [11], and video streaming services [12, 1, 2, 3] all achieve admirable success, attracting a large number of users and changing the way *digital goods* are delivered over the Internet. According to the overlay structure, P2P systems can be broadly classified into two categories: tree-based systems and mesh-based systems. The tree-based systems, such as ESM [12], have well-organized overlay structures and typically distribute video by actively pushing data from a peer to its children peers. In contrast, a mesh-based system is not confined to a static topology. A peer dynamically connects to a subset of random peers in the system based on the content and bandwidth availability on peers. Video chunk is pulled by a peer from its neighbor who has already obtained the chunk. The study in [13] shows that the mesh-based scheme is superior over the tree-based scheme thanks to the dynamic mapping of content to the delivery paths.

Despite the success of mesh-based P2P streaming systems, the *quality of experience* perceived at end users requires further improvement in terms of video quality, startup delay, and playback smoothness. Measurement study on PPLive [14, 15] showed that most programs have bit rates around 400 kbps and the start-up delays for a channel range from a few seconds to a few minutes. Performance comparison study [5] further sheds lights on the potential root cause of limited streaming rate currently supportable over the Internet. It turns out a naieve mesh-based P2P scheme is not able to efficiently utilize the available bandwidth resources available in the P2P system.

Several efforts have been made to improve the resource utilization. [6] proposes a two phase swarming scheme where the fresh content is diffused to the entire system in the first phase, and peers exchange available content in the second phase. Network coding has been applied to P2P streaming and a reality check is done in [16]. [17] further proposes a P2P live streaming protocol that takes full advantage of random network coding to improve the overall performance. Neither above approaches, however, can be proved to optimally utilize the bandwidth resources. [4] develops a centralized solution that fully utilizes peer uploading bandwidth and achieves the streaming rate upper bound. [7] designs an optimal randomized distributed algorithm. [8] further expands the result in [7] and designs a set of pushed-based schemes. [18] analyzes pull-based streaming protocol and shows that the streaming

rate can be near optimal if the long delay and large signaling overhead are tolerable. A hybrid push-pull based scheme is proposed mitigating some of the issues. The aforementioned studies, nevertheless, always require the assumption of the fully connected mesh in their optimality proofs. HCPS overcomes the challenge by introducing hierarchically clustered P2P overlay and distributed queue-based scheduling algorithms, which are provably optimal with small startup delay. Full-fledge prototype further allows us to address realistic implementation issues, and evaluate the system over the real network.

The general idea of clustering has been applied to different networking problmes [19, 20, 21]. The clustering in HCPS bears some similarities with that in NICE [22]. NICE utilizes a hierarchical clustering architecture to build a multicast tree over which low bit-rate videos are streamed, while HCPS employs the mesh-based P2P streaming over the hierarchy to support the high bit-rate video streaming. Different design goals lead to different challenges and different solutions. NICE attempts to solve the tree building control overhead problem, while HCPS intends to build a clustering hierarchy that supports the P2P streaming rate approaching the optimum.

3. Hierarchically Clustered P2P Streaming: Overlay Construction

HCPS groups peers into clusters which are then organized into a tree-like hierarchy, with clusters as vertices. Within each cluster, peers are fully connected and one peer is elected as the *cluster head*. The cluster head acts as the video proxy for the cluster and downloads video by joining its parent cluster as a normal/non-head peer. Effectively, the cluster heads serve as the *links* that connect clusters to their parent clusters in the hierarchy. The number of connections on each normal peer is bounded by the size of its cluster. Cluster heads additionally maintain connections in the upper level cluster. The number of connections for cluster heads could be doubled.

Video content is streamed down from the upper-level clusters to the lower-level clusters. Specifically, the source server feeds video to each cluster at the first level in the hierarchy. Peers in the same cluster efficiently utilize their upload capacity and download/upload video from/to one another by employing a distributed chunk scheduling algorithm to be presented in Section 4. Since cluster heads at level two join clusters at level one, they relay the video to clusters at level two. Peers at level two employ the same chunk scheduling algorithm to collaboratively download video from their cluster

head. The procedure repeats iteratively and video is streamed to all peers at all levels of the hierarchy. Figure 1 illustrates a simple example of two-level

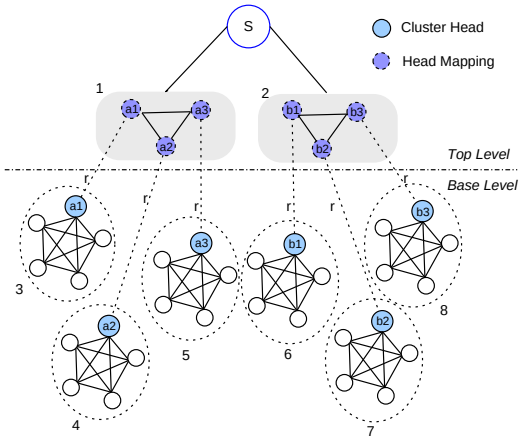


Figure 1: Hierarchically Clustered P2P Streaming System: two-level hierarchy with 30 peers.

HCPS hierarchy. At the base level, peers are grouped into six clusters, each with five peers. The peers are fully connected within a cluster. Each cluster has one cluster head. At the top level, cluster heads form two clusters, each with three heads. The video server (source) distributes the content to all cluster heads. At the base level, each cluster head acts as the video proxy in its cluster and distributes the downloaded video to other peers in the same cluster.

While decreasing the number of connections of peers, HCPS has good scalability. Suppose that the cluster size is bounded by N^{max}, and the source can support up to N^s top layer clusters. The two-level HCPS system, as shown in Fig. 1, can accommodate up to $N^s(N^{max})^2$ peers. Assume $N^s = 10$ and $N^{max} = 20$, HCPS supports up to around 4,000 peers. The maximum number of connections a peer needs to maintain is 40 for cluster head and 20 for normal peer, which is quite manageable. More peers can be accommodated by adding more levels into the hierarchy. If each of the $N^s N^{max}$ clusters at the base level has one peer with spare bandwidth higher than or equal to the playback rate, each such peer can be used as a video proxy to connect more levels to the two-level hierarchy. By adding one additional level to Figure 1, each peer proxy can support $(N^{max})^2$ additional peers. The system can sustain at least $N^s(N^{max})^3$ more peers, i.e., 80K peers, only with one more level.

Next we look into the streaming rate that can be broadcasted over a HCPS overlay.

3.1. Maximum supportable streaming rate of HCPS overlay

For a given HCPS overlay, $\mathcal{H}$, we define its *maximum supportable streaming rate*, $r^{\mathcal{H}}$, to be the largest video bit-rate that can be broadcasted to all peers over this overlay. We further define *optimum upper bound*, r^{max}, to be the largest possible streaming rate that can be broadcasted to the same set of peers *without overlay structure constraint*. Hence $r^{\mathcal{H}} \leq r^{max}$. As shown in [4], the optimum upper bound, r^{max}, is governed by the following fomula:

$$r^{max} = \min\{u_s, \frac{u_s + \sum_{i=1}^{N} u_i}{N}\}, \tag{1}$$

where u_s refers to the server's upload bandwidth, the bit rate at which a video can be served out of the server, and u_i refers to the peer i's upload bandwidth. In the following, we first describe how to compute *maximum supportable streaming rate* for a fixed HCPS overlay topology. We then investigate the peer clustering strategies that would lead to efficient HCPS overlay topology with *maximum supportable streaming rate* approaching the optimum upper bound. In this study we follow the common assumption that the downlink bandwidth is not the bottleneck and thus do not set limit on them.

Assume N peers are grouped into C clusters in a HCPS mesh. Set V_c denotes a subset of peers in cluster c, $c \in [1, C]$. A peer can participate in a HCPS cluster either as a normal peer or as a cluster head. For instance, in Fig. 1, peer $a1$ joins two clusters. It is the cluster head in cluster 3 and a normal peer in cluster 1. Denote by u_{ic} the amount of upload capacity of peer i contributed to cluster c as a normal peer, and by h_{ic} the amount of upload capacity of peer i contributed to cluster c as a cluster head. Hence $u_{(a1)3} = 0$, $h_{(a1)3} \geq 0$ since peer $a1$ is the head in cluster 3; and $u_{(a1)1} \geq 0$, $h_{(a1)1} = 0$ since peer $a1$ is a normal peer in cluster 1. $u_{(a1)c} = h_{(a1)c} = 0$ for all other clusters since peer $a1$ is not member of them. Further denote by u_c^s the amount of source upload capacity used for cluster c. Similarly, $u_c^s = 0$ if the source does not participate in cluster c. If r_c^{max} represents the maximum streaming rate for cluster c as governed by Equation (1) (note that peers are fully connected within a cluster thus the upper bound can be achieved), the maximum supportable streaming rate for a given cluster-based HCPS overlay $\mathcal{H}$, $r^{\mathcal{H}}$, can be formulated as the following optimization problem.

$$r^{\mathcal{H}} = \max_{\{u_{ic}, h_{ic}, u_c^s\}} \left\{ \min \left[r_c^{max} | c = 1, 2, \ldots, C \right] \right\} \tag{2}$$

subject to:

$$r_c^{max} = \min \left\{ \frac{\sum_{i=1}^{N} (u_{ic} + h_{ic})}{V_c}, \sum_{i=1}^{N} h_{ic} + u_c^s \right\} \tag{3}$$

$$\sum_{c=1}^{C} (u_{ic} + h_{ic}) \leq u_i \tag{4}$$

$$\sum_{c=1}^{C} u_c^s \leq u_s \tag{5}$$

where Eqn. (3) is true for all $c \in [1, C]$ and Eqn. (4) is true all for $i \in [1, N]$. The value of u_{ic}, h_{ic}, and u_c^s is determined by the HCPS topology.

The maximum supportable streaming rate for a given mesh topology is the streaming rate that can be supported by all clusters. Since the cluster head participates in both upper layer and lower layer clusters and the source's upload capacity is used by several top layer clusters, the supportable streaming rate for HCPS can be maximized by adjusting the allocation of clusters' upload capacity and source's upload capacity. This explains the Equation (2). The first term in Equation (3) represents the average upload capacity per peer in cluster c; and the second term represents the cluster head's upload capacity (note that cluster head can be either the source server or a peer). Since the maximum value of streaming rate at cluster c, r_c^{max}, is governed by the theoretical upper bound (see Equation (1)), this leads to the Equation (3). Further, the amount of bandwidth allocated for the upper layer and low layer clusters must not surpass its total upload capacity, which explains Equation (4). Finally, for the source, the total allocated upload capacity for all clusters must not surpass the source's total upload capacity, which explains Equation (5).

3.2. Peer clustering strategies

For the same set of peers, different overlay topology leads to different *maximum supportable streaming rate*. We now investigate peer clustering strategies that enable *maximum supportable streaming rate* $r^{\mathcal{H}}$ approaching optimum upper bound r^{max}. We start with a simple numerical example to

illustrate the major factors affecting the HCPS's performance. The clustering strategies are laid out thereafter.

Assume a HCPS system possesses 400 peers and one source node. The cluster size is set to be 20. The peers are grouped into 20 base layer clusters and one top layer cluster for cluster heads. The optimization problem (2) is numerically solved to obtain $r^{\mathcal{H}}$, i.e., the *maximum supportable streaming rate* for HCPS overlay. Further assume that peers' upload capacities obey the following distribution, (128 kbps, 0.2), (384 kbps, 0.4), (1 Mbps, 0.25), and (5 Mbps, 0.15), where the first entry is the upload capacity and the second entry is associated probability. The distribution is drawn from the measurement study conducted in [23]. We compare the *maximum supportable video bit-rate* of the following two scenarios. In the first scenario, we select one cluster and assign its nodes' upload bandwidths according to the bandwidth distribution listed above. The rest of clusters use the exact same bandwidth profile. In the second scenario, each node individually selects its upload bandwidth according to the same probability distribution. In both scenarios, the node with the largest upload bandwidth is selected as the cluster head. We solve the optimization problem (2). The solution shows that the value of $r^{\mathcal{H}}$ of the first scenario is very close to the optimum upper bound, r^{max} ($> 99\% r^{max}$), while $r^{\mathcal{H}}$ only achieves roughly 60% of the optimum upper bound for the second scenario.

The difference between first and second scenario is significant, however not surprising. According to Equation (2), the *maximum supportable streaming rate*, $r^{\mathcal{H}}$, takes the minimum cluster streaming rate of all clusters. The cluster streaming rate (Equation (3)) is the minimum of cluster average upload capacity and the cluster head's rate. Intuitively, the peers should be divided into clusters with equal (similar) average upload capacity to avoid wasting resources. Based on the above discussion, we propose following heuristics:

- *The discrepancy of individual clusters' average upload capacity per peer should be minimized.*

- *The cluster head's upload capacity should be as large as possible.* The cluster head's capacity allocated for the base layer capacity has to be larger than the average upload capacity to avoid being the bottleneck. Furthermore, the cluster head also joins the upper layer cluster as a normal peer. Ideally, the cluster head's rate should be $\geq 2r^{\mathcal{H}}$ so that at least one $r^{\mathcal{H}}$ can be allocated to both the upper and lower cluster.

- *The number of peers in a cluster should be bounded from the above by a relatively small number.* The number of peers in a cluster determines the out-degree of peers. Large out-degree is prohibitive for peers to perform properly in real networks.

Below we describe the dynamic peer management in HCPS that realizes the above heuristics. HCPS system has a bootstrapping node whose task is to manage HCPS topology to balance the resources among clusters. Meanwhile, a cluster head manages the peers in its own cluster. The bootstrap node's task is to handle the peer arrival and peer departure in such a way that $r^{\mathcal{H}}$ is as close to the optimal as possible. Below we describe three key operations in HCPS that handle the peer dynamics: peer join, peer departure, and cluster re-balancing.

3.2.1. Peer join

Depending on the new arrival's estimated upload capacity compared to the current $r^{\mathcal{H}}$, the new peer is classified into three categories: *HPeer(resource rich peer)* if $u \geq r^{\mathcal{H}} + \theta$, *MPeer(resource medium peer)* if $r^{\mathcal{H}} - \theta < u < r^{\mathcal{H}} + \theta$, and *LPeer(resource poor peer)* otherwise, where u is new peer's upload capacity, and θ is a configuration parameter.

Denote by N^{max} the maximum number of peers allowed in a cluster. All clusters with less than N^{max} peers are eligible to accept the new peer. If the upload capacity of the new peer, u, is greater than some eligible cluster heads' upload capacity by a margin, the peer is assigned to the cluster with the smallest cluster head upload capacity. The new peer is to replace the original cluster head, and the original head becomes the normal peer and stay in the cluster.

If the new peer does not replace any cluster head, it is assigned to a cluster according to the peer type. Specifically, the peer is assigned to the cluster with the minimum average upload capacity if the peer is *HPeer*; the peer is assigned to the cluster with the smallest number of peers if it is *MPeer*; and the peer is assigned to the cluster with the maximum average upload capacity if it is *LPeer*. The idea behind this is to balance the upload resources among clusters.The new peer is redirected to the corresponding cluster head, and bootstrap node asks the cluster head to admit the new peer.

3.2.2. Peer departure

The handling of peer departure is straight forward. If the peer is a normal peer, it informs the cluster head of its departure. The cluster head takes the

peer off its cluster member list, and informs other peers in the same cluster about its departure.

If the departing peer is the cluster head, it informs the bootstrap node its departure. The bootstrap node selects the backup peer from existing peers in the cluster as the new cluster head. This promotion process requires the remaining peers of the cluster and other cluster heads to update member list, since both clusters linked by the head are influenced directly. Meanwhile, the new cluster head needs to take over the original management task. If a peer crashes, the handling is the same after the peer is sensed inactive through heart-beat ping messages.

3.2.3. Cluster re-balancing

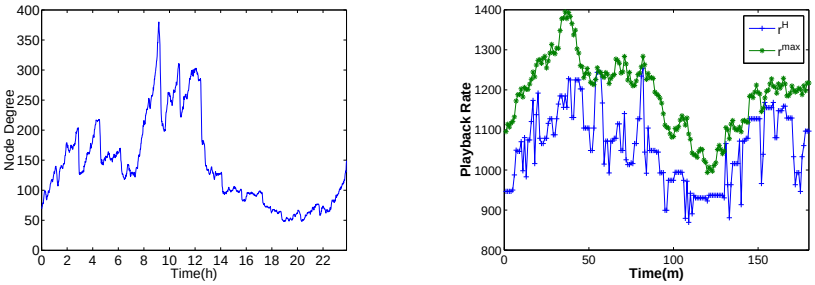
The clusters may lose balance in terms of the number of peers and the amount of resources in a cluster as the result of peer churn. In HCPS, the bootstrap node periodically attempts to re-balance the clusters. At the end of an epoch, the bootstrap node first attempts to balance the cluster size. The clusters are sorted in the descending order of cluster size. If the gap between the clusters with the largest and the smallest number of peers is greater than the $threshold = \max\{\alpha \cdot N^{max}, \beta \cdot \overline{N}\}$, where $\overline{N}$ is the average cluster size, these two clusters will be merged and then splitted into two balanced clusters. The above process continues until no clusters violates the condition.

In the second phase of cluster re-balancing, the bootstrap node tries to balance the resources. The clusters are sorted in the descending order of average upload capacity per peer. If the average upload capacity difference of the clusters with highest and lowest upload capacity is greater than the threshold of $\theta \cdot \overline{u}$, where $\overline{u}$ is the average upload capacity, these two clusters will be merged and then splitted into two balanced clusters.

3.3. Maximum supportable streaming rate vs. optimum upper bound

The effectiveness of peer clustering strategies is evaluated using flow level simulation next. The simulation is driven by the trace collected from the measurement study of a large scale P2P live streaming system PPLive [2]. We extract the information of peers arrival and their life time from a one-day PPLive channel trace collected at April 3, 2006. The upload capacity of peers are assigned randomly according to the distribution described in Section III. The server upload capacity is set to be $2Mbps$.

Fig. 2(a) depicts the number of peers in the system over the period of 24 hours. Peer churn is evident in this figure. The number of peers peaks around 9:00AM-12:00PM when the system has more than 200 concurrent users.



(a) One-day degree evolution of popular channel

(b) $r^{\mathcal{H}}$ vs. r^{max}

Figure 2: Rate Evolution in Peak Time

Fig. 2(b) plots the HCPS streaming rate vs. optimum upper bound during the peak hour (9:00AM-12:00PM). The curve of HCPS streaming rate follows the trend of optimum streaming rate closely. The oscillation of optimum streaming rate is due to the upload capacity change caused by peer churn. The oscillation of HCPS streaming rate is due to the following two reasons: (i) the upload capacity change; and (ii) the resource imbalance among HCPS clusters. While the first factor is caused by the nature of P2P system, the second factor is mitigated by intelligent clustering strategies as laid out in Section 4. To better understand the persistent performance, Table 1 reports the average streaming rate over a four hour period during the day. The HCPS *maximum supportable streaming rate* is within 90% of the optimum upper bound all the time.

Table 1: Average Rate per Time Zone

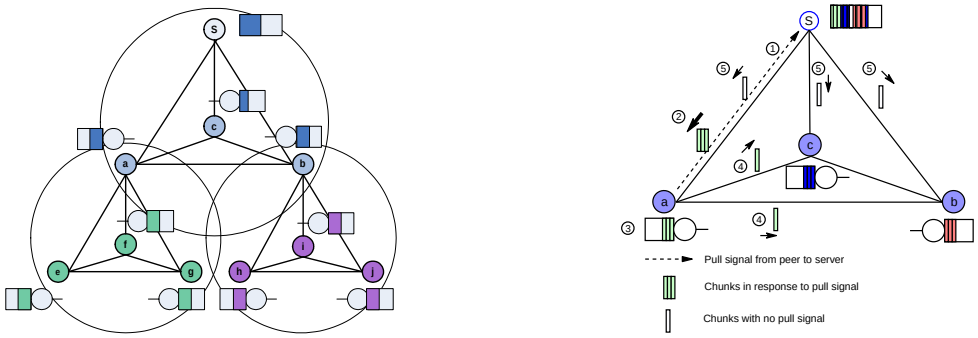
Time Zone(h)	0-4	4-8	8-12	12-16	16-20	20-24
Perfect(Mbps)	1.15	1.21	1.18	1.21	1.01	1.14
HCPS(Mbps)	1.07	1.11	1.04	1.12	8.99	1.06

4. Distributed queue-based chunk scheduling algorithms

In Section 3, we studied how to construct HCPS overlay to have large *maximum supportable streaming rate*. In this section, we describe distributed queue-based chunk scheduling algorithms that achieve the *maximum supportable streaming rate*, $r^{\mathcal{H}}$. Designing a distributed chunk scheduling algorithm that achieves the *maximum supportable streaming rate* is not trivial because of *content bottleneck* effect. Peers' uplink bandwidths are wasted whenever chunks requested by a peer's connected neighbors are not available at this peer. In Eqn. (2), *maximum supportable stream rate* $r^{\mathcal{H}}$ is computed without considering *content bottleneck*.

The basic building blocks of distributed queue-based chunk scheduling algorithms are queues. Data chunks are pulled/pushed from server/clusterhead to peers, cached at peers' queue, and relayed from peers to its neighbors. The availability of upload capacity is inferred from the queue status such as the queue size or if the queue is empty. Signals are passed between peers and server to convey the information if a peer's upload capacity is available. Queue-based design enables peers to be self-adaptive to the dynamic network environment, and automatically converges to the *maximum supportable streaming rate*. Fig. 3(a) depicts a HCPS system with one server and nine peers grouped into three clusters. Node a and b are cluster heads and join both top level and bottom level clusters. Fig. 3(b) zooms into the top-level cluster and illustrates the data and signaling exchanges among peers and source server. Each peer maintains several local queues including a forward queue. Using peer a as an example, the signal and data flow is described next. Pull signals are sent from peer a to the server whenever the forward queues become empty (or have fallen below a threshold) (step 1 in Fig. 3(b)). The server responds to the pull signal by sending three data chunks back to peer a (step 2). These chunks will be stored in the peer a's forward queue (step 3) and be relayed to peer b and peer c (step 4). When the server has responded to all 'pull' signals on its 'pull' signal queue, it serves one duplicated data chunks to all peers (step 5). These data chunks will not be stored in peer a's, b's and c's forward queues and will not be relayed further.

There are three types of peers in the HCPS system: *normal peer*, *cluster head*, and *source server*. In the following the scheduling algorithm and the queuing model of different peers are described.



(a) Queue-based chunk scheduling with nine peers

(b) Queue-based chunk scheduling in top cluster

Figure 3: HCPS Overlay using queue-based chunk scheduling: an example with one source and nine peers grouped into three clusters

4.1. Scheduling and queuing model of normal peer

Fig. 4 depicts the queuing model for normal peers. A peer maintains a playback buffer that stores all received streaming content from the source server and other peers. The received content from different nodes is assembled in the playback buffer in playback order. The peer's media player renders/displays the content from this buffer. Meanwhile, the peer maintains a forwarding queue which is used to forward content to all other peers. The source server marks the data content either as F-marked content or NF-marked content before transmitting them. *F (forwarding)* represents content that should be relayed/forwarded to other peers. *NF (non-forwarding)* indicates that content is intended for this peer only and no forwarding is required. *NF* content is filtered out at peers. *F* content is stored in the forward queue, marked as NF content, and forwarded to other peers. Because the relayed content is always marked as *NF* at the relaying peer, data content is relayed at most once, which reduces the content distribution time and startup delay. In order to fully utilize a peer's upload capacity, the peer's forwarding queue should be kept busy. A signal is sent to the source server to request more content whenever the forwarding queue becomes empty. This is termed a 'pull' signal. The rules for marking the content at the source server and cluster heads are described next.

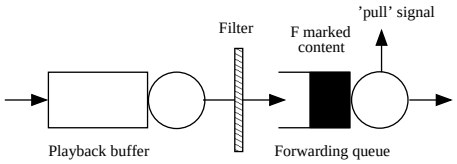


Figure 4: Queue Model of Peers: some video in playback buffer will be moved to forwarding queue to be forwarded to other peers.

4.2. Scheduling and queueing model of cluster head

Cluster heads joins two clusters. That is, a cluster head will be a member of two clusters concurrently. A cluster head behaves as a normal peer in the upper-level cluster and as the source proxy for the lower-level cluster. The queuing model of the cluster head, thus, is two levels as well, as shown in Fig. 5. As a normal node in the upper-level cluster, the cluster head receives the content from peers within the same cluster as well as from the source server. It relays the 'F' marked content to other peers in the same upper level cluster and issues 'pull' signals to the source server when it needs more content. At the upper level, the cluster head also may issue a throttle signal to the source server, which is described in more detail below.

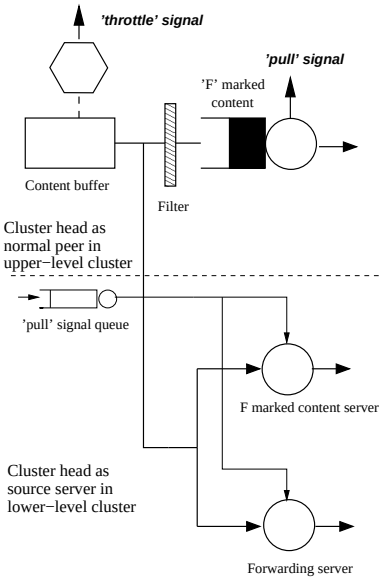


Figure 5: Queue Model of Cluster Head

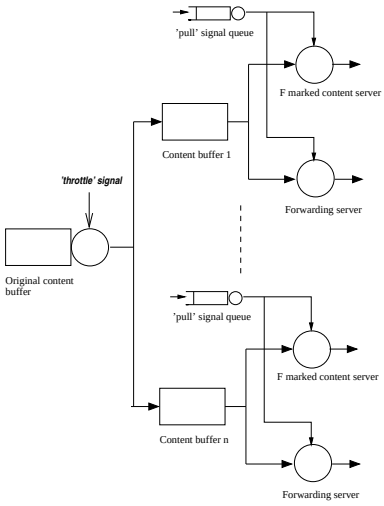


Figure 6: Queue Model of Source Server

Still referring to Fig. 5, as the source in the lower-level cluster, the cluster

head has two queues: a content queue and a signal queue. The content queue is a multi-server queue with two servers: an 'F' marked content server and a forwarding server. Which server to use depends on the status of the signal queue. Specifically, if there is 'pull' signal in the signal queue, a small chunk of content is taken off content buffer, marked as 'F', and served by the 'F' marked content server to the peer that issued the 'pull' signal. The 'pull' signal is then removed from the 'pull' signal queue. On the other hand, if the signal queue is empty, the server takes a small chunk of content (data chunk) from the content buffer and transfers it to the forwarding server. The forwarding server marks the data chunk as 'NF' and sends it to all peers in the same cluster.

A cluster head's upload capacity is shared between upper-level cluster and lower-level cluster. In order to achieve the *maximum supportable streaming rate* of a HCPS system, the forwarding server and 'F' marked content server in the lower-level cluster always has priority over the forwarding queue in the upper-level cluster. Specifically, the cluster head will not serve the forwarding queuing in the upper-level until the content in the playback buffer for the lower-level cluster has been fully served.

A lower-level cluster can be overwhelmed by the upper-level cluster if the streaming rate supported at the upper-level cluster is larger than the streaming rate supported by the lower-level cluster. If the entire upload capacity of the cluster head has been used in the lower-level, yet the content accumulated in the upper-level content buffer continues to increase, it can be inferred that the current streaming rate is too large to be supported by the lower-level cluster. A feedback mechanism at the playback buffer of the cluster head is introduced. The playback buffer has a content rate estimator that continuously estimates the incoming streaming rate. A threshold is set at the playback buffer. If the received content is over the threshold for an extended period of time, say T, the cluster head will send a throttle signal together with the estimated incoming streaming rate to the source server. The signal reports to the source server that the current streaming rate surpasses the rate that can be consumed by the lower-level cluster headed by this node. The source server may choose to respond to the 'throttle' signal and acts correspondingly to reduce the streaming rate. As an alternative, source may choose not to slow the current streaming rate. In such scenario, the peers in the cluster headed by this cluster head will experience degraded viewing quality such as frequent frame freezing. However, the quality degradation will not spill over to other clusters.

4.3. Scheduling and queueing model of source

The source server in HCPS system may participate in one or multiple top-level clusters. The source server has one sub-server for each top-level cluster, as shown in Fig. 6. The source server maintains an original content queue that stores the data/streaming content. It also handles the 'throttle' signals from the lower level clusters and from cluster heads the source server serves at the top-level clusters. The server regulates the streaming rate according to the 'throttle' signals from the peers. The server's upload capacity is shared among all top-level clusters. The bandwidth sharing follows the following rules: (i) the cluster that lags behind other clusters significantly (by a threshold in terms of content queue size) has the highest priority to use the upload capacity; and (ii) if all content queues are of the same/similar size, then clusters are served in a round robin fashion.

4.4. Optimality of queue-based chunk scheduling algorithms

Theorem 1. *Assume that the propagation delay between peers and between a peer and the server is negligible and the data content can be transmitted at an arbitrary small amount. The distributed queue-based chunk scheduling algorithm achieves the optimum upper bound, r^{max}, over a fully connected mesh, and achieves the maximum supportable streaming rate, $r^{\mathcal{H}}$, over the hierarchically cluster P2P overlay, $\mathcal{H}$.*

See Appendix for the proof. Below we discuss the implementation considerations in realizing the distributed queue based chunk scheduling algorithms in practice.

5. Implementation considerations

We implement a full functioning HCPS live streaming system which not only serves as a proof of concept but also allows us to evaluate the system performance over the real network. Below we use the source server as an example to illustrate the design philosophy. The architecture design aims at solving practical implementation challenges including the impact of chunk size, network congestion, and peer churn. The same design philosophy can be applied to the design of other components such as normal peers and cluster heads.

5.1. Source server architecture

The source server maintains one sub-server for each cluster (see Fig. 6). For the ease of illustration, we assume there is one top-level cluster in the system. We also assume that the content source server is the bootstrap node. As the bootstrap node, the content source server manages peer information (such as peer id, IP address, port number, etc.) and replies to the request for peer list from incoming new peers.

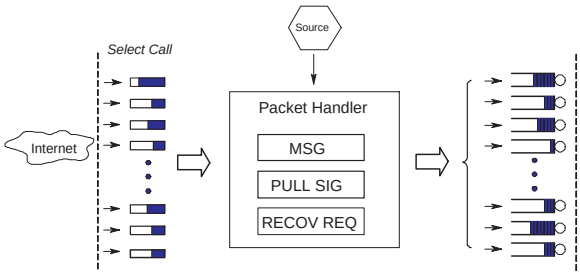


Figure 7: Server Architecture

Figure 7 illustrates the architecture of the source server. Using the 'select call' mechanism to monitor the connections with peers, the server maintains a set of input buffers to store received data. There are three types of incoming messages: *management message*, *pull signal*, and *missing chunk recovery request*. Correspondingly three independent queues are formed for these messages. If the output of handling these messages needs to be transmitted to remote peers, the output is put on the per-peer out-unit.

There is one out-unit for each destination peer to handle the data transmission process. Figure 8 depicts an example. Each out-unit has four queues for a given peer: management message queue, F-marked content queue, NF-marked content queue, and missing chunk recovery queue. The management message queue stores responses to management requests. An example of a management request is when a new peer has just joined the P2P system and requests the peer list. The F/NF marked content queue stores the F/NF marked content intended for this peer. Finally, chunk recovery queue stores the missing chunks requested by the peer.

Different queues are used for different types of traffic in order to prioritize the traffic types. Specifically, management messages have the highest priority, followed by F-marked content, and NF-marked content. The priority of recovery chunks can be adjusted based on the design requirement. Manage-

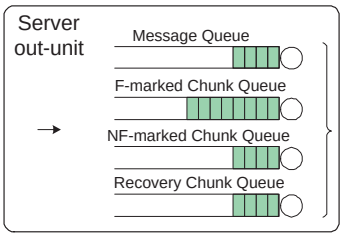


Figure 8: Server Out-unit Queues

ment messages have the highest priority because it is important for the system to run smoothly. The content source server replies to each 'pull' signal with F-marked chunks. F-marked chunks are further relayed to other peers by the receiving peer. The content source server sends out a NF-marked chunk to all peers when the 'pull' signal queue is empty. NF-marked chunks are used by the destination peer only and will not be relayed further. Therefore, serving F-marked chunk promptly improves the utilization of peers' upload capacity and increases the overall P2P system streaming rate.

Another reason for using separate queues is to deal with bandwidth fluctuation and congestion inside the network. Many P2P researchers assume that server/peer's upload capacity is the bottleneck. In our experiments over PlanetLab, it has been observed that some peers may slow down significantly due to congestion inside networks. If all the peers share the same queue, the uploading to the slowest peer will block the uploading to remaining peers. This is similar to the head-of-line blocking problem in input-queued switch design. Separate queues avoid inefficient blocking caused by slow peers. Peers' and cluster heads' architectures employ the similar design principle and are omitted here.

5.2. Impact of chunk size and propagation delay

In the optimality proof, it was assumed that the chunk size is arbitrarily small and the propagation delay was negligible. In practice, the chunk size is on the order of kilo-bytes to avoid excessive transmission overhead caused by protocol headers. The propagation delay is on the order of tens to hundreds of milliseconds. Hence, it is necessary to adjust the timing of issuing 'pull' signals by the peers and increase the number of F-marked chunks served at the content source server to allow the decentralized scheduling method to better utilize peers' upload capacities.

At the server/cluster-head side, K F-marked chunks are transmitted as a batch in response to a 'pull' signal from a requesting peer (via the F-marked content queue). A larger value of K would reduce the 'pull' signal frequency and thus reduce the signaling overhead. This, however, increases peers' threshold to be shown in Equation (6). Denote by T_i the threshold for peer i to issue 'pull' signal. A 'pull' signal is sent to server whenever the number of chunks in the queue is less than or equal to T_i. The time to empty the forwarding queue with T_i chunks is $t_i^{empty} = (M-1)T_i\delta/u_i$, where M is the number of peers in a cluster. Meanwhile, it takes $t_i^{receive} = 2t_{si} + K\delta/u_s + t_q$ for peer i to receive K chunks after it issues a pull signal. Here t_{si} is the propagation delay between the source server and peer i, $K\delta/u_s$ is the time required for server to transmit K chunks, and t_q is queuing delay seen by the 'pull' signal at the server pull signal queue. In order to receive the chunks before the forwarding queue becomes fully drained, $t_i^{empty} = t_i^{receive}$. This leads to:

$$T_i = \frac{(2t_{si} + K\delta/u_s + t_q)u_i}{(M-1)\delta}. \tag{6}$$

All quantities are known except t_q, the queuing delay incurred at the server side signal queue. If the source server is the bottleneck (case 1 in the optimality proof), the selection of T_i would not affect the streaming rate as long as the server is always busy. If server resource is rich (case 2 in the optimality proof), since the service rate of signal queue is faster than the pull signal rate, t_q is very small. So we set t_q to be zero. Following 'pull' signal threshold formula is used to guide the threshold selection:

$$T_i = \frac{(2t_{si} + K\delta/u_s)u_i}{(M-1)\delta}. \tag{7}$$

5.3. Missing chunk recovery

Peer churn and network congestion may cause chunk losses. Sudden peer departure, such as node or connection failure, leaves the system no time to reschedule the chunks still buffered in the peer's out-unit. In case the network routes are congested to some destinations, the chunks waiting to be transmitted may overflow the queue in the out-unit, which leads to chunk losses at the receiving end. The missing chunk recovery scheme enables the peers to recover the missing chunks to avoid viewing quality degradation.

Each peer maintains a playback buffer to store the video chunks received from the server and other peers. The playback buffer has three windows:

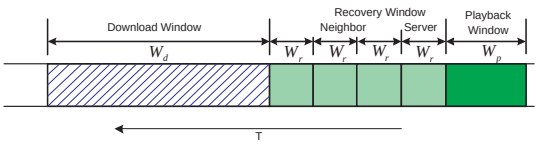


Figure 9: Missing Chunk Recovery

playback window, recovery window, and download window. W_p, W_r and W_d denote the size (in terms of number of chunks) of playback window, recovery window, and download window, respectively. The media player renders/displays the content from the playback window. Missing chunks in the recovery window are recovered using the method described below. Finally, the chunks in the downloading window are pulled and pushed among the server and the other peers.

Heuristics are employed to recover the missing chunks. If peers leave gracefully, the server is notified and the F-marked chunks waiting in the out-unit will be assigned to other peers. The missing chunks falling into the recovery window are recovered as follows. First, the recovery window is further divided into four sub-windows. Peers send the chunk recovery messages to the source server directly if the missing chunks are in the window closest in time to the playback window. These chunks are urgently needed otherwise the content quality will be impaired. An attempt is made to recover the missing chunks in the other three sub-windows from other peers. A peer randomly selects three recovery peers from the peer list, and associates one with each sub-window. The peer needs recovery chunks sends chunk recovery messages to the corresponding recovery peers. By randomly selecting a recovery peer, the recovery workload is evenly distributed among all peers.

5.4. Optimality of startup delay

We derive the startup delay here because the queue size has been derived in 5.2. A peer's startup delay, τ, is $\tau = h \cdot \tau_{intra-cluster}$, where h is the height at which the cluster resides in the HCPS overlay, and $\tau_{intra-cluster}$ denote the delay incurred in distributed content within a cluster. Assume there are N peers in the system and the cluster size is M, the value of h is at the order of $O(logN/M)$ if the cluster tree is balanced.

The delay incurred inside a cluster comprises of two parts: the time spending waiting in the queue, and the time transmitting a chunk to all other peers in the same cluster. Hence $\tau_{intra-cluster} = \max_i\{T_i \cdot \delta(M-1)/u_i + (M-$

$1)\delta/u_i\}$. We take the maximum overall all peer in the cluster is because the slowest peer determines the startup delay of this cluster. Plug T_i into the above equation, we have $\tau_{intra-cluster} = \max_i\{(2t_{si} + K\delta)/u_s + (n-1)\delta/u_i\}$. t_{si}, K, δ, M, and u_s (cluster head's upload bandwidth devoted to the cluster under consideration) are all constant. Thus the value of $\tau_{intra-cluster}$ is at the order of $O(M)$. Therefore, the startup delay is at the order of $O(M \cdot logN/M) = O(logN)$, which is optimal as proven in [8].

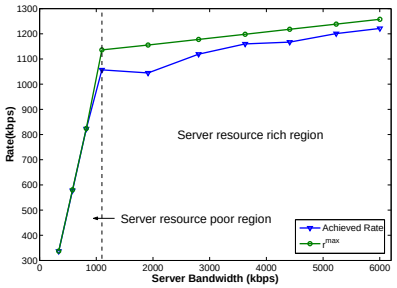
6. Performance Evaluation

Next we examine the performance of HCPS via experiments over PlanetLab [9]. We started with the single cluster experiments to understand the performance of queue-based algorithms and the system dynamics in the face of peer churn and bandwidth variations. We then look into a larger scale system with multiple clusters, investigating the cluster head overhead, the effect of clustering on streaming delay, and the adaptiveness of HCPS.

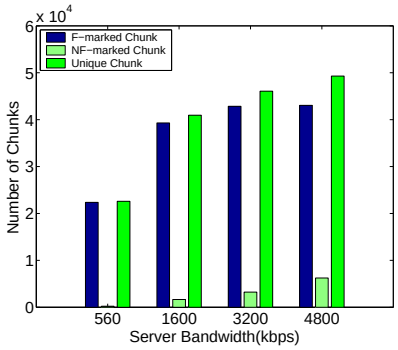
We select PlanetLab nodes with sufficient bandwidth and use software package Trickle [24] to set a node's upload capacity. In our HCPS streaming system, all connections between nodes are TCP connections. TCP connections avoid network layer data losses, and facilitate the use of Trickle [24] to set a node's upload capacity. In our experiments, we observe that Trickle is not 100% accurate on setting the available bandwidth. The obtained upload bandwidth is slightly larger ($< 8\%$) than the value we set using Trickle. To account for this error, we measure the actual upload bandwidth, and use the measured rate for plotting the graphs. The upload capacities of peers are assigned randomly according to the distribution: (128kbps, 0.2), (384kbps, 0.4), (1Mbps, 0.25), and (4Mbps, 0.15). The largest uplink speed is adjusted from 5 Mbps [23] to 4 Mbps to ensure that PlanetLab nodes have sufficient bandwidth to emulate the targeted rate. We follow the common assumption that the downlink bandwidth is not the bottleneck and thus do not set limit on them. The chunk size is chosen to be one KBytes. In the following, we first evaluate the queue-based chunk scheduling algorithm over a fully connected mesh. We then investigate its performance over hierarchically clustered P2P overlay.

6.1. Performance of queue-based chunk scheduling over a fully connected mesh

Forty peers join the system at the beginning of the experiment, and stay for the entire duration of the experiment. The content source server's upload capacity varies from 320 kbps to 5.6 Mbps. For each server upload capacity setting, we run an experiment for 5 mins. The achieved streaming rate is collected every 10 seconds and the average value is reported at the end of each experiment.



(a) Achieved Rate vs. Optimum Upper Bound



(b) Distribution of Chunks from Server

Figure 10: Optimality Evaluation Results

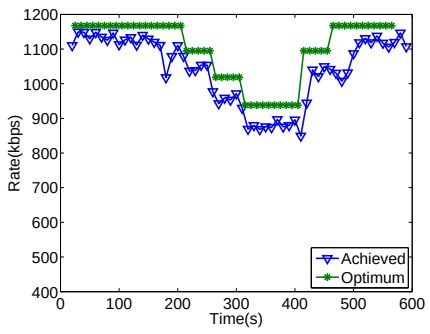
Fig. 10(a) shows the achieved streaming rate (measured at experiments) vs. the optimum upper bound with different server bandwidths. The difference never exceeds 10% of the optimum rate possible in the system. The curves exhibit two segments with turning point at around 1.1 Mbps. According to $r^{max} = \min\{u_s, \frac{u_s+\sum_{i=1}^{n} u_i}{n}\}$, the server bandwidth is the bottleneck when it is smaller than 1.1 Mbps (server resource poor scenario). The streaming rate is equal to the source rate. As the server bandwidth becomes greater than 1.1 Mbps, the peers' average upload capacity becomes the bottleneck (server resource rich scenario). The streaming rate still increases linearly, however, with a smaller slope. Notice that queue-based chunk scheduling performs better in the server resource poor scenario than in the server resource rich scenario. We plot the numbers of F-marked and NF-marked chunks sent out by the source server to explain what causes the difference.

As shown in Fig. 10(b), when the server bandwidth is 560 kbps (source resource poor scenario), very few NF-marked chunks are transmitted. In

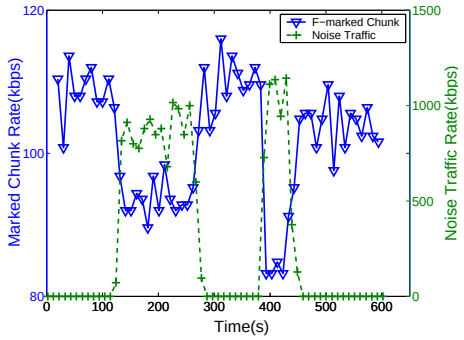
theory, no NF-marked chunks should be sent in this scenario since signal queue is always non-empty. We do see several NF-marked chunks, which is caused by the background noise traffic in the network. The interference of the background traffic occasionally causes the server's pull signal queue becomes empty. In contrast, more and more NF-marked chunks are sent by the server as its uplink capacity increases beyond 1.1 Mbps (source resource rich scenarios). In the server resource poor scenario, the server sends out F-marked chunks exclusively. As long as the pull signal queue is not empty, the optimum streaming rate can be achieved. In the server resource rich scenario, the server sends out both F-marked and NF-marked chunks. If F-marked chunks are delayed at server or along the route from the server to peers due to the bandwidth variations or peer churn, peers can not receive F-marked chunks promptly. Peers' forward queues become idle and upload bandwidth is wasted. Nevertheless, queue-based chunk scheduling always achieve the streaming rate within 10% of the optimum upper bound.

6.2. Adaptiveness to peer churn and bandwidth variations

Peer churn has been identified as one of the major disruptions to the performance of p2p system. We study how queue-based chunk scheduling performs in face of peer churns next. In this 10 minutes experiment, the server bandwidth is set to be 2.4 Mbps. Three peers with the bandwidth of 4 Mbps are selected to leave the system at time of 200 seconds, 250 seconds, and 300 seconds, respectively. Two peers rejoin the system at time of 400 seconds, and the third one rejoins the system at time of 450 seconds.



(a) Streaming Rate under Peer Churn



(b) Marked Chunk Receiving Rate with Background Traffic

Figure 11: Adaptiveness to Peer Churn and Bandwidth Variations

Fig. 11(a) depicts the achieved rate vs. optimum rate every 10 seconds. Although the departure and the join of a peer does introduce oscillation to the achieved streaming rate, overall the achieved streaming rate tracks the optimum rate closely. The difference between them never exceeds 12% of the optimum rate.

In addition to peer churn, the network bandwidth varies over time due to background noise traffic. To evaluate queue-based chunk scheduling's adaptiveness to network bandwidth variations, the following experiment is conducted. We set up a sink on a separate PlanetLab node not participating in P2P streaming. One peer in the streaming system with upload capacity of 4 Mbps is selected to establish multiple parallel TCP connections to the sink. Each TCP connection sends out garbage data to the sink. The noise traffic generated by those TCP connections causes variations in the bandwidth available for the P2P video threads on the selected peer.

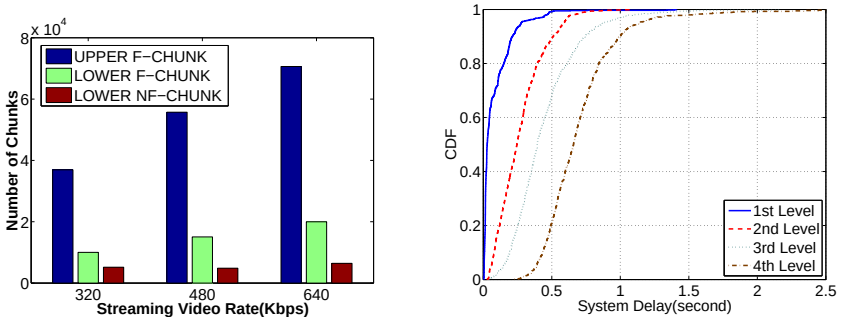
Fig. 11(b) depicts the rate at which the F-marked chunks are received at the selected peer together with the sending rate of noise traffic. During time periods of (120 sec, 280 sec) and (380 sec, 450 sec), the noise traffic threads are on. The queue-based chunk scheduling method adapts quickly to the decreasing available bandwidth by reducing its pull signal rate. Consequently, the server reduces the rate of F-marked chunks sent to the selected peer. When the noise traffic is turned off, the server sends more F-marked chunks to the selected peer to fully utilize its available uploading bandwidth. The self-adaptiveness of the queue-based chunk scheduling method makes the overall achieved streaming rate close to the optimum rate.

6.3. Cluster head overhead in HCPS

Cluster heads play a crucial role in HCPS since they *glue* clusters together. At upper-level clusters, cluster heads forward 'F' marked chunks to other neighbors as a normal node; at lower-level clusters, cluster heads behave as proxy servers, serving *pull* requests with 'F' marked chunks and broadcasting 'NF' marked chunks when spare bandwidth becomes available.

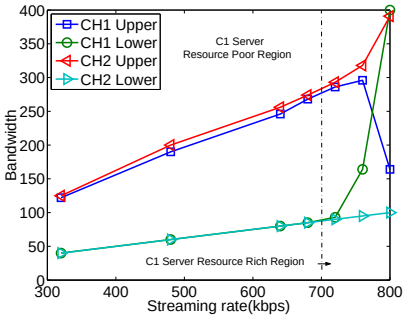
In this experiment, four clusters are used and each cluster has 20 nodes whose upload bandwidth obeys the distribution as listed at the beginning of this section. One cluster is placed at the top level and retrieves the data content directly from the source server. The other three clusters are placed at the second level. The cluster heads of the second level clusters are members of the top level cluster (the top level cluster also has some normal peers that are not cluster heads). The source server has the uplink bandwidth of

4Mbps. The peers in the top level cluster join the system at the beginning of the experiment. The peers in second level clusters join after 20 seconds so that the top-level cluster is at stable state. The experiment lasts for 5 minutes. Based on our off-line data analysis, the experiment duration of 300 seconds is sufficient since the systems reaches the steady state within tens of seconds.



(a) Chunk Distribution of Cluster Head

(b) CDF of system delay at different levels



(c) Bandwidth usage of cluster heads on different levels

Figure 12: Experiment results of HCPS with multiple clusters

Fig. 12(a) shows the total number of chunks transmitted by a cluster head as the normal peer in upper-level cluster and as the server in lower-level cluster. Different streaming rates are tested to check their impact. Majority of cluster head bandwidth has been consumed at the upper level, while the bandwidth allocated to the lower level roughly equals to the streaming rate. The streaming rates of 320 kbps, 480 kbps, and 640 kbps, are smaller than the average bandwidth of each cluster. Hence the cluster head only needs

to respond to the "pull" signals with F-marked chunks at the lower level. F-marked chunks are quickly distributed to all peers by chunk relaying at peers. There are some 'NF' marked chunks sent out by the cluster head at lower level. A close look reveals that these 'NF' marked chunks are generated at the time when the lower level clusters just join the system. Since the TCP connections have not been fully set up and 'pull' signals are yet to be issued, the cluster head quickly pushes out 'NF' marked chunks to boost the peer start. Once the system reaches the steady state, peers' upload bandwidth is better utilized and no 'NF' chunks are sent/required. Although the cluster head enrolls into two clusters, no significant bandwidth overhead is incurred.

6.4. Effect of clustering on streaming delay

Video start-up delay is a key metric in P2P live streaming. HCPS achieves the scalability by grouping peers into clusters and forming hierarchy among clusters. Intuitively, the chunk delay, from the video source to a destination peer, will increase as the depth of the cluster hierarchy grows. Below we study the effect of clustering on chunk delays.

Again four clusters are used with each cluster having 20 peers. To facilitate the investigation of delay, four clusters are placed on different levels of a four-level hierarchy, one on top of another. Each cluster is headed by one node with the largest bandwidth at the upper level cluster. The streaming rate is set to be 320kbps.

Fig. 12(b) shows the cumulative distribution (CDF) of chunk delays observed at a randomly selected peer at different cluster levels. As expected, the lower level peers experience longer delays than the peers in upper levels, and the delay increases approximately linearly with the number of levels. However, even at level four, the average delay is still under one second. This is due to the facts that (1) HCPS gives higher priority to lower level clusters so that the fresh chunks can be distributed to lower level clusters as fast as possible; and (2) within a cluster, the chunk is relayed at most once, which also reduces the end-to-end delay.

6.5. Adaptiveness of hierarchically clustered P2P streaming system

Dynamic network environment imposes significant challenges to peer-to-peer applications, especially in maximizing the throughput. Here we want to demonstrate the adaptiveness of HCPS, i.e., HCPS cluster heads are able to adjust the amount of bandwidth resources distributed to different clusters so as to maximize the overall throughput. In this experiment, one cluster

is placed at top level and three clusters are placed at the second level. We select one cluster in the second level and intentionally reduce the bandwidth of peers in this cluster. Specifically, two peers in this cluster are assigned with the bandwidth of 384 kbps instead of 4 Mbps which they originally get. The selected cluster thus has less average bandwidth per peer than other two clusters.

Fig. 12(c) depicts the bandwidth usage of two cluster heads at both upper and lower levels. In the figure, CH1 denotes the cluster head corresponding to the lower cluster (C1) with less bandwidth resource. CH2 denotes another cluster head (cluster C2) with regular bandwidth distribution. We then test how cluster heads dynamically allocate their bandwidth between upper and lower levels when the streaming rate changes. When the streaming rate is low, both cluster heads assign the amount of bandwidth equal to streaming rate to the lower level clusters, and spend more bandwidth on forwarding 'F' marked chunks at the upper level clusters. As the streaming rate continues to increase and exceeds the normal peers' average upload bandwidth at cluster C1 (labeled as *C1 server resource rich region*), the cluster heads behave differently. The bandwidth resource from normal peers in C1 cluster is not enough to support present streaming rate. Therefore cluster head CH1 devotes more and more bandwidth to the lower level cluster. The extra bandwidth allows CH1 to push out 'NF' marked chunks. Meanwhile, cluster head CH2 only contribute bandwidth equal to the streaming rate to the lower level cluster since the normal peers in C2 provide sufficient bandwidth. This allows CH2 to spend more bandwidth at upper level cluster to compensate for the reduced bandwidth contribution from CH1. It is the adaptiveness of HCPS that optimizes the bandwidth resource usage and enables the entire system to support larger streaming rates.

7. Conclusions

In this paper, we proposed the hierarchically clustered P2P streaming (HCPS) that is capable of providing live streaming service to a large number of users with video rate approaching the optimum upper bound. Peers share chunks with other peers using the distributed scheduling mechanism. The peer uplink bandwidth resources are efficiently utilized, and the system as a whole operates close to the optimal point. The system is implemented and is used to conduct experiments over the Internet. The results demonstrate the optimality and the adaptiveness of the proposed HCPS framework.

For future work, we plan to deploy the HCPS system in real field to provide streaming service to users with dedicated end devices. The dedicated end devices, such as set-top boxes and home gateways, stay online for long period of time and shall benefit from HCPS's hierarchical design. The deployment will help us gain valuable insights and validate our design.

Appendix A. Proof of Theorem 1

Proof: Suppose the video content is divided into small chunks. The server sends out *one* chunk each time it serves a 'pull' signal. A peer issues a pull signal to the server whenever the forwarding queue becomes empty. δ denotes the chunk size. u_s is content source server's upload capacity, u_i is peer i's upload capacity, and n is the number of peers in the system. The optimum upper bound, r^{max}, is: $r^{max} = \min\{u_s, \frac{u_s+\sum_{i=1}^{n} u_i}{n}\}$.

For peer i, it takes time of $(n-1)\delta/u_i$ to forward one data chunk to all peers. Let r_i be the maximum rate at which the 'pull' signal is issued from peer i. Hence $r_i = u_i/(n-1)\delta$. The maximum aggregated rate of 'pull' signal received at server, r, is $r = \sum_{i=1}^{n} r_i = \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$. It takes server δ/u_s to serve a pull signal. Hence the maximum 'pull' signal rate a server can accommodate is u_s/δ. Now consider the following two scenarios/cases:

• Case 1: $u_s/\delta \leq \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$

In this scenario, the server cannot handle the 'pull' signal at maximum rate. The signal queue at the server side is hence never empty and the entire server bandwidth is used to transmit F-marked content to peers. In contrast, a peer's forward queue becomes idle while waiting for the new data content from the source server. Since each peer has sufficient upload bandwidth to relay the F-marked content (received from the server) to all other peers, the peers receive content sent out by the server at the maximum rate.

The maximum supportable streaming rate is equal to the server's upload capacity. The condition $u_s/\delta \leq \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$ is equivalent to $u_s \leq \frac{u_s+\sum_{i=1}^{n} u_i}{n}$, i.e., the server resource poor scenario. Hence the streaming rate is consistent with r^{max} and the maximum streaming rate is reached.

• Case 2: $u_s/\delta > \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$

In this scenario, the server has the upload capacity to service the 'pull' signals at the maximum rate. During the time period when the 'pull' signal queue is empty, the server transmits duplicate NF-marked content to all peers. The amount of upload capacity used to serve F-marked content is $r\delta = \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}\delta =$

$\frac{\sum_{i=1}^{n} u_i}{n-1}$.

The server's upload bandwidth used to serve NF-marked content is therefore $u_s - \frac{\sum_{i=1}^{n} u_i}{n-1}$. For each individual peers, the rate of receiving NF-marked content from server is $(u_s - \frac{\sum_{i=1}^{n} u_i}{n-1})/n$ since there are n peers in the system. The streaming rate at peers is:

$$\frac{\sum_{i=1}^{n} u_i}{n-1} + (u_s - \frac{\sum_{i=1}^{n} u_i}{n-1})/n = \frac{u_s + \sum_{i=1}^{n} u_i}{n}. \tag{A.1}$$

The condition $u_s/\delta > \frac{\sum_{i=1}^{n} u_i}{(n-1)\delta}$ is equivalent to $u_s > \frac{u_s + \sum_{i=1}^{n} u_i}{n}$, i.e., the server resource rich scenario. Again, the streaming rate reaches r^{max}. This concludes the proof of first claim of theorem.

The proof of second claim is done by contradiction. Denote by by $r^{\mathcal{H}}$ the maximum supportable streaming rate of HCPS, and by r^d the maximum steaming rate that can be reached using distributed queue-based chunk scheduling algorithms. Suppose queue-based chunk scheduling algorithms cannot support the maximum supportable streaming rate, $r^d < r^{\mathcal{H}}$. Hence there is at least one cluster in HCPS, say cluster k, whose streaming rate, $(r_k^{max})^d$, is less than $r^{\mathcal{H}}$, i.e., $(r_k^{max})^d < r^{\mathcal{H}}$. Let $(r_k^{max})^*$ be cluster d's streaming rate when achiving maximum supportable streaming rate $r^{\mathcal{H}}$. $r^{\mathcal{H}} \leq (r_k^{max})^*$, thus $(r_k^{max})^d < (r_k^{max})^*$.

Denote by $(u_i^{t_i})^*$, $(u_i^{h_i})^*$, and $(u_c^s)^*$ the optimal bandwidth allocation that achieves maximum supportable streaming rate of $r^{\mathcal{H}}$. According to Eqn. (3),

$$(r_k^{max})^* = \min\Big\{\frac{\sum_{\forall i, t_i \in V_k}(u_i^{t_i})^* + \sum_{\forall i, h_i \in V_k}(u_i^{h_i})^*}{|V_k|}, \sum_{\forall i, h_i \in V_k}(u_i^{h_i})^* + (u_c^s)^*\Big\} \tag{A.2}$$

Further denote by $(u_i^{t_i})^d$, $(u_i^{h_i})^d$, and $(u_c^s)^d$ the bandwidth allocation in distributed queue-based chunk scheduling algorithms. According to the first claim, queue-based chunk scheduling in one cluster can achieve the optimum upper bound. Thus

$$(r_k^{max})^d = \min\Big\{\frac{\sum_{\forall i, t_i \in V_k}(u_i^{t_i})^d + \sum_{\forall i, h_i \in V_k}(u_i^{h_i})^d}{|V_k|}, \sum_{\forall i, h_i \in V_k}(u_i^{h_i})^d + (u_c^s)^d\Big\} \tag{A.3}$$

Therefore:

$$\min\Big\{\frac{\sum_{\forall i,t_i\in V_k}(u_i^{t_i})^d+\sum_{\forall i,h_i\in V_k}(u_i^{h_i})^d}{|V_k|},\ \ \sum_{\forall i,h_i\in V_k}(u_i^{h_i})^d+(u_c^s)^d\Big\}$$
$$<\min\Big\{\frac{\sum_{\forall i,t_i\in V_k}(u_i^{t_i})^*+\sum_{\forall i,h_i\in V_k}(u_i^{h_i})^*}{|V_k|},\ \ \sum_{\forall i,h_i\in V_k}(u_i^{h_i})^*+(u_c^s)^*\Big\}. \quad \text{(A.4)}$$

Now we argue that Eqn. (A.4) does not hold. Normal peers in cluster k devote their entire upload bandwidth solely to cluster k. Hence they are equal at both sides of Eqn. (A.4). For cluster heads that are normal peers in cluster k and are cluster heads in other clusters, they will be able to devote equal or more upload bandwidth to cluster k in HCPS. These peers use equal/less bandwidth in the clusters they head due to the fact $r^d < r^{\mathcal{H}}$. Finally, the cluster head of cluster k in HCPS is able to contribute as much bandwidth as in optimal solution. Hence the inequality in Eqn. (A.4) does not hold, which leads to the contradiction and concludes the proof. ∎

Note that in case 2 where the aggregate 'pull' signal arrival rate is smaller than the server's service rate, it is assumed that the peers receive F-marked content immediately after issuing the 'pull' signal. The above assumption is true only if the 'pull' signal does not encounter any queuing delay and can be serviced immediately by the content source server. This means that (i) no two 'pull' signals arrive at the exact same time and (ii) a 'pull' signal can be serviced before the arrival of next incoming 'pull' signal. Assumption (i) is commonly used in queuing theory and is reasonable since a P2P system is a distributed system with respect to peers generating 'pull' signals. The probability that two 'pull' signals arrive at exactly the same time is low. Assumption (ii) means that the data can be transmitted in arbitrary small amounts, i.e., the size of data chunk, δ, can be arbitrarily small. In practice, the size of data chunks is limited in order to reduce the overhead associated with data transfers.

References

[1] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A data-driven overlay network for live media streaming," in *Proceedings of IEEE INFOCOM*, 2005.

[2] PPLive, "PPLive Homepage," http://www.pplive.com.

[3] SopCast, "SopCast Homepage," http://www.sopcast.org.

[4] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *Proceedings of IEEE INFOCOM*, 2007.

[5] C. Liang, Y. Guo, and Y. Liu, "Is random scheduling enough for p2p live streaming?" in *IEEE ICDCS*, 2008.

[6] N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, August 2007.

[7] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proceedings of IEEE INFOCOM*, 2007.

[8] T. Bonald, L. Massouli, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *Proceedings of ACM SIGMETRICS*, 2008.

[9] PlanetLab, "PlanetLab Homepage," http://www.planet-lab.org.

[10] BT, "Bittorent Homepage," http://www.bittorrent.com.

[11] unkown, "Skype webpage," http://www.skype.com/.

[12] Y.-H. Chu, S. G.Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS*, 2000.

[13] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," in *Proceedings of IEEE INFOCOM*, 2007.

[14] X. Hei, Y. Liu, and K. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE Journal on Selected Areas in Communications, the special issue on advances in P2P streaming*, 2008.

[15] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, November 2007.

[16] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *Proceedings of IEEE INFOCOM*, 2007.

[17] ——, "R^2: Random push with random network coding in live peer-to-peer streaming," in *IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, December 2007.

[18] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the Power of Pull-based Streaming Protocol: Can We Do Better?" *IEEE Journal on Selected Areas in Communications*, 2007.

[19] A. Amis, R. Prakash, D. Huynh, and T. Vuong, "Max-min d-cluster formation in wireless ad hoc networks," in *Proceedings of IEEE INFOCOM*, 2000.

[20] O. Younis and S. Fahmy, "Distributed clustering in ad-hoc sensor networks: a hybrid, energy-efficient approach," in *Proceedings of IEEE INFOCOM*, 2004.

[21] M. Zhao and Y. Yang, "A framework for mobile data gathering with load balanced clustering and MIMO uploading," in *Proceedings of IEEE INFOCOM*, 2011.

[22] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proceedings of ACM SIGCOMM*, 2002.

[23] C. H. Ashwin R. Bharambe and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Network Performance Mechanisms," in *Proceedings of IEEE INFOCOM*, 2006.

[24] Trickle, "Trickle Homepage," http://monkey.org/~marius/pages/?page=trickle.