Strategies of Collaboration in Multi-channel P2P VoD Streaming

Zhi Wang
Tsinghua University
Email: wangzhi04@mails.tsinghua.edu.cn

Chuan Wu
The University of Hong Kong
Email: cwu@cs.hku.hk

Lifeng Sun, Shiqiang Yang
Tsinghua University
Email: {sunlf,yangshq}@tsinghua.edu.cn

***Abstract*— As compared to live peer-to-peer (P2P) streaming, modern P2P video-on-demand (VoD) systems have brought much larger volumes of videos and more interactive controls to the Internet users. Nevertheless, the larger number of available videos and the flexibility of allowing users to jump back and forth in a video, have led to much fewer numbers of concurrent peers watching at a similar pace, that reduces the chance for collaborative chunk supply among peers and thus significantly increases the server bandwidth cost [1]. Towards the ultimate goal of maximizing peer resource utilization, in this paper, we design effective strategies for both cross-channel and intra-channel collaborations in multi-channel P2P VoD systems, such that individual peer's resources, including download/upload bandwidths and the cache capacity, are effectively utilized to maximize the streaming qualities in all the channels. In particular, each peer actively and strategically determines the supply-and-demand imbalance in different channels, as well as that among different chunks within each video, makes use of its surplus download capacity to fetch chunks with the most need, and then serves those chunks using its idle upload bandwidth, all without impairing its own streaming quality. Our extensive trace-driven simulations show the effectiveness of our strategies in reducing the server cost while guaranteeing high streaming qualities in the entire system, even during extreme scenarios such as unexpected flash crowds.**

I. Introduction

P2P VoD streaming represents the state of the art application of the P2P paradigm in today's Internet [4], [9], [2], that provides thousands of videos to millions of users with full interactive viewing control. To alleviate the streaming servers, P2P VoD streaming applications are based on the design philosophy of allowing peers watching the same video to exchange media chunks among themselves. Nevertheless, as compared to the live P2P streaming, the alleviation of server bandwidth is less significant in P2P VoD streaming, due mainly to the lower level of playback synchrony among VoD peers. It has been a common observation that a VoD peer's upload capacity idles as few neighbors request the chunks it currently caches, rendering a waste of peer resources [3].

The situation is further exacerbated when we consider the large number of video channels a P2P VoD system typically provides: The popularity of the channels is largely skewed, typically following the *zipf* distributions, where the majority are unpopular channels with a few tens of concurrent viewers or less [12]. Peers watching the unpopular videos often need to download video chunks from the streaming servers, as few concurrent peers are caching the chunks in need; ironically, the upload bandwidths at peers in those unpopular channels are largely idle and wasted due to the low chance of serving the chunks they cache. Existing measurements have shown that up to 70% of video chunks may still need to be supplied from the servers in modern P2P VoD systems [1].

To utilize peers' surplus upload and download bandwidths to assist in the streaming of the whole system, a few recent proposals advocate cross-channel help among the peers. In the context of P2P live streaming, Wu *et al*. [11] decouple the channel a peer is watching from the channel it assists in, and allocate the latter in a centralized fashion; Wang *et al*. [10] compare 3 potential categories of cross-channel bandwidth allocation designs, using linear programming models. P2P VoD streaming is more complicated than P2P live streaming, as not only cross-channel assistance is needed, but also collaborations to stream chunks belonging to different portions of the same video are required. Suh *et al*. [8] propose a hybrid pull-push mechanism, by which selected portions of a video are preloaded (pushed) onto peers before their streaming starts, while the peers request (pull) the missing chunks during streaming. Sharma *et al*. [7] have designed *dPAM*, a prefetching protocol that peers prefetch chunks they may watch in the future, in order to enhance the chunk distribution in a P2P VoD channel and thus alleviate the server load. Zhang *et al*. [13] utilize the idle capacities of Internet hosts to help in a P2P VoD channel, while the helpers may not be watching any channel at all. However, none of these proposals has taken collaboration of normal peers in multiple VoD channels into consideration.

Focusing on the most challenging scenario of multi-channel P2P VoD streaming, our work aims to clearly address the following critical questions: how would a peer *actively* and *dynamically* decide when it has spare capacities to assist in the streaming of other chunks/channels (that it is not watching)? Which chunks or channels should it help? How should it best utilize its capacities to achieve most effective cross-channel and intra-channel chunk upload?

Our answers to the above questions are a set of effective strategies designed to support cross- and intra-channel collaborations in multi-channel P2P VoD systems, which make full utilization of the surplus bandwidths and cache capacities at peers. In particular, each peer actively and strategically determines the supply-and-demand imbalance across channels, as well as that among different chunks within a video. As

a fully adaptive design, a peer schedules its download and upload over time, contributing to cross- or intra-channel collaborative streaming when its bandwidths are abundant, or temporarily suspending the assistance upon the increase of resource demand in its own viewing channel. We extensively evaluate our design using simulations driven by real-world P2P VoD traces, to demonstrate its effectiveness in reducing the server cost and guaranteeing high streaming qualities, even during extreme scenarios such as unexpected flash crowds.

The rest of this paper is organized as follows. We present our system model and design of the strategies in Sec. II, discuss their practical implementation in Sec. III, evaluate the design and implementation using trace-driven simulations in Sec. IV, and finally conclude the paper in Sec. V.

II. Strategies of Collaboration: Model and Design

We consider a typical pull-based P2P VoD streaming system providing multiple video channels to the users (peers). Each video is divided into a consecutive stream of chunks (referred to *video segments* hereinafter) with an equal length. When a peer tunes in to watch a video, a tracker server assigns it a list of other online peers caching similar portions of the same video. The peer then exchanges *segment map* with those neighbors, which indicates the available segments the peers currently cache in their local buffers, and then requests segments from neighbors for its video playback. To guarantee smooth playback, a peer may resort to the dedicated streaming server for urgent segments, which have not been received immediately before the playback deadlines.

In modern P2P VoD systems [6], [9], peers' download bandwidths are commonly abundant (*e.g.*, $1-3$ Mbps ADSL connections) as compared to the representative streaming bit rates ($500-800$ Kbps); caches allocated at individual peers are typically as large as $2-3$ Gbytes [4]. Our design makes full utilization of a peer's resources to assist in the streaming of other channels or segments it is currently not watching. We refer to the channel it is watching as *viewing channel*, and the channel it assists in as *helping channel*, respectively. Correspondingly, we refer to peers who are watching a channel as *viewing peers* of the channel, and those who are helping but not watching as *helper peers* of the channel. We first address the question which channel/segment a peer should fetch to assist in when it has spare capacities (Sec. II-A), and then discuss the dynamic strategies how a peer actively serves as a helper (Sec. II-B). We list important notations in our design in Table I.

TABLE I
Important Notations

Symbol	Definition
T	The length of each round in seconds
M	The total number of channels in the P2P VoD system
$u_c(t)$	The server upload capacity needed to support the streaming in channel c in round t
b_c	The streaming bitrate of channel c
$X_c(t)$	The set of peers who are watching channel c in round t
$r_c(t)$	The total upload bandwidth from peers to serve viewing peers in channel c in round t
$V(t)$	The channel resource vector at round t
F	The number of consecutive segments a peer downloads during each fetching as helpers
$Q_c^s(t)$	The number of copies of segment s in channel c served by peers in round t
$K_c^s(t)$	The number of copies of segment s in channel c served by the server in round t
$S_c(t)$	The segment resource vector of channel c at round t
$N(t)$	The total number of segments a peer uploads in round t
$w(t)$	The fraction of streaming segments over all the segments a peer uploads in round t
α,β	Thresholds in helping state transition
U_p	The upload capacity of peer p as the maximum number of segments that can be uploaded in a round
NH	The state when a peer is not actively serving others
SH	The state when a peer serves more streaming requests than helping requests
FH	The state when a peer serves more helping requests than streaming requests

A. Strategies of Helping Channel/Segment Selection

A peer first decides which channel it will assist in and then the segments it will fetch to serve as a supplier. A *channel resource vector* and a *segment resource vector* are employed for the two purposes, respectively. In our design, we divide time into rounds for the protocol execution, each corresponding to T seconds. The selection of channel/segment may happen every one or a few rounds, depending on the state a peer is currently in, which will be discussed in detail in Sec. II-B.

1) Selection of the Helping Channel: We use a *channel resource vector* to indicate the upload resource shortage within each channel. Let channel resource index $u_c(t)$ denote the upload capacity needed from the dedicated server to support the streaming (*i.e.*, downloading segments for playback) in channel c in round t, which is defined as:

$$u_c(t) = b_c\,|X_c(t)| - r_c(t)$$

Here b_c is the bitrate of channel c and $X_c(t)$ is the set of viewing peers in the channel in round t. $r_c(t)$ represents the overall amount of upload bandwidth provided by peers to support the viewing peers in $X_c(t)$. We note that $r_c(t)$ includes the upload bandwidth from viewing peers, as well as the net contribution (upload bandwidth to serve segments minus bandwidth needed to download those segments) from the helper peers. We will discuss how to derive $u_c(t)$ when we detail the practical implementation of our design in Sec. III.

We further normalize the vector $\{u_1(t), u_2(t), \ldots, u_M(t)\}$ using $\bar{u}_c(t) = \frac{u_c(t)}{\sum_{i=1}^{M} u_i(t)}$, $c = 1, \ldots, M$, and derive the channel resource vector $V(t) = \{\bar{u}_1(t), \bar{u}_2(t), \ldots, \bar{u}_M(t)\}$. We use each element in $V(t)$ as the probability in our helping channel selection: channels with larger $\bar{u}_c(t)$, *i.e.*, relatively more bandwidth insufficiency from peer contributions, are more likely to be selected by a helper peer. The helping channel selected by a peer can be a different channel from its own viewing channel (the cross-channel assistance scenario) or can be the same as its viewing channel as well (the intra-channel help scenario).

2) Selection of Segments in the Helping Channel: After the helping channel is selected, a peer chooses the segments to fetch in the channel. In our design, the peer will select a starting segment, and then download F consecutive segments

in the stream from the starting segment on, where F is an implementation parameter in our experiments.

We use a segment resource vector to decide the starting position. Let $g_c^s(t)$ be the ratio of segment s in channel c downloaded directly from the server, over the total number of segment s downloaded in round t: $g_c^s(t) = \frac{K_c^s(t)}{Q_c^s(t)+K_c^s(t)}$ where $Q_c^s(t)$ is the number of copies of segment s supplied by peers in round t and $K_c^s(t)$ is the number served by the dedicated server. We consider the average ratio of segments served by the server over all the F consecutive segments starting from segment s, $\mu_c^s(t)$, defined as follows:

$$\mu_c^s(t) = \frac{\sum_{k=s}^{s+F'-1} g_c^k(t)}{F'}, F' = \begin{cases} F & s+F-1 \le L_c \\ L_c - s + 1 & \text{otherwise} \end{cases}$$

Here L_c is the total number of segments in channel c. We normalize the vector $\{\mu_c^1(t), \mu_c^2(t), \ldots, \mu_c^{L_c}(t)\}$ and derive the segment resource vector $S_c(t) = \{\bar{\mu}_c^1(t), \bar{\mu}_c^2(t), \ldots, \bar{\mu}_c^{L_c}(t)\}$, where $\bar{\mu}_c^s(t) = \frac{\mu_c^s(t)}{\sum_{i=1}^{L_c} \mu_c^i(t)}$, $s = 1, \ldots, L_c$. We use each element in $S_c(t)$ as the probability in our segment selection within helping channel c: the segments starting from s with larger $\bar{\mu}_c^s(t)$, *i.e.*, with relatively more bandwidth demand from the server, are more likely to be selected. If the selected F segments are already cached by the peer, it will run the channel/segment selection again.

B. Strategies of Dynamic Helping

In this section, we discuss a peer's dynamic strategies on when to assist in the helping channel and when to focus on the streaming of its own viewing channel, as well as how to schedule its upload of segments to viewing and helper peers, respectively. We divide peers' requests for segments into two types: a *streaming request* refers to the request for a segment to be played by the requesting peer; and a *helping request* corresponds to the request for a segment from a helper peer.

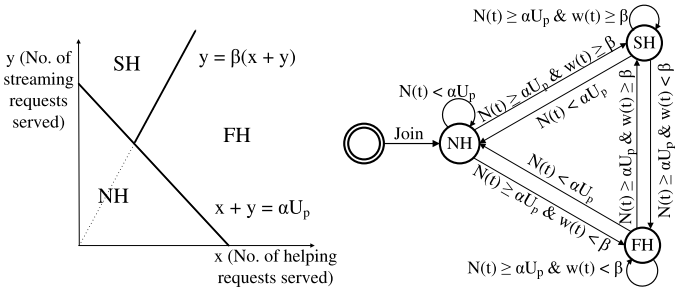
1) Switching among Helping States: We describe a peer's dynamic behavior using 3 helping *states*, which are decided by the amount of streaming and helping requests it has served in the previous round:

Non-active helping (NH): the state when the utilization level of a peer's upload capacity is relatively low, given that the total number of segments it has uploaded to others in the previous round is less than αU_p, where $\alpha \in (0,1)$ is a threshold parameter and U_p is the upload capacity of the peer.

Streaming helping (SH): the state when a peer is serving more streaming requests than helping requests, such that the total number of segments it has uploaded in the round is no smaller than αU_p, and the fraction of segments uploaded for streaming requests is no lower than $\beta \in (0,1]$.

Fetching helping (FH): the state when a peer serves more helping requests than streaming requests, such that the total number of segments it has uploaded in the round is no smaller than αU_p, and the fraction of segments uploaded for helping requests is no lower than $1-\beta$ (*i.e.*, the fraction of upload to address streaming requests is lower than β).

Fig. 1 (a) illustrates the definition of the three states. Let $N(t)$ denote the total number of segments the peer uploads in round t, and $w(t)$ be the fraction of the streaming segments uploaded. Figure 1 (b) further gives the transition among the three states, that occurs at the end of each round.



(a) State division (b) State transition

Fig. 1. Three helping states at each peer.

2) Segment Fetching from Helping Channels: At the beginning of a round, if a peer is in the SH state, it downloads only the segments for viewing but not any segments for helping. For a peer in the NH state, excepting downloading segments for playback, it carries out a new helping channel and segment selection, and fetches segments that are chosen, as long as the previous fetching process (downloading F consecutive segments with respect to the previous helping channel/segment selection) is done. For peers in the FH state, they carry out new channel selection and segment fetching less frequently, *i.e.*, every several rounds. In our experiments, a peer selects a new helping channel and the corresponding segments if it has remained in the FH state for 5 rounds.

The design rationale lies that we aim to guarantee sufficient upload bandwidth to serve the streaming requests, while maximally utilizing the surplus upload capacity to distribute segments to helper peers. A peer in the SH state does not need to fetch more segments from any helping channel, since the segments its caches are already popular, as requested by many viewing peers; in this way, the upload consumption to serve this peer's helping requests can also be saved. A peer in the NH state or the FH state may still need to retrieve more segments from the helping channels, in order to improve the utilization of its upload bandwidth. Peers in the FH state perform channel selection and segment fetching less frequently than those in the NH state, as the upload capacities of the former are already better utilized as amplifiers for helping requests.

3) Upload Schedule To Serve Different Requests: A peer p may fetch and cache segments from different channels, as a result of the dynamic channel and segment selection strategies executed over time (a *LRU* cache replacement strategy is applied when the peer cache becomes full). Therefore, it may be assigned by the tracker server to serve viewing and helper peers in multiple channels. To schedule the upload of segments at peer p, requests from the neighbors are added into a priority queue upon reception, where the priority of a request is decided as follows: (1) streaming requests have higher priorities over helping requests for segments in all channels; (2) among the streaming requests, a request for a segment in peer p's viewing channel is further prioritized; (3) the streaming requests in p's viewing channel and those in its helping channels, are prioritized based on deadlines of

the segment playback, respectively; (4) among the helping requests, one is prioritized if it corresponds to a segment or channel with larger values of segment or channel resource indices. The above priority rules are also applied to upload scheduling at the server, excepted (2).

In addition, a peer serves the requests within its upload capacity: if the number of requests received in one round is higher than its upload capacity, any unaddressed request will remain in the queue, until (1) it has stayed in the queue over 3 rounds (with respect to a helping request), or (2) the playback deadline of the requested segment has been missed (for a streaming request).

III. Practical Implementation of Helping Segment/Channel Selection

The dynamic upload and download strategies for each peer, as discussed in Sec. II-B, rely completely on a peer's local upload and download statistics, and therefore can be implemented in a fully distributed fashion. On the other hand, the dynamic segment and channel selection strategies, as discussed in Sec. II-A, may require global information such as the amount of server and peer upload bandwidth contribution, as well as statistics of segment download in each channel.

In helping channel selection, the channel resource index $u_c(t)$ for each channel c in round t, can be estimated at the server, as the amount of upload bandwidth the server supplies to each channel to address the *streaming requests* from viewing peers of the channel in the round. Therefore, the channel resource vector, $V(t)$, can be derived by the server and sent to peers who are making channel selection decisions.

After the helping channel is chosen, segment selection within the channel can be achieved in two ways:

Centralized segment resource vector computation: A simple centralized implementation is to have each peer periodically report its local statistics to the server, regarding where each of its received segments is from (the server or other peers). Using these statistics, the server can calculate the numbers of copies of each segment served from the server and from the peers, respectively, and derive the segment resource vector for the channel. As such a centralized implementation will be very costly in large-scale P2P VoD systems, we seek to design a distributed approach, in which peers estimate segment resource vectors based on local information exchanged with their neighbors.

Distributed segment resource vector computation: Each peer watching or helping channel c keeps a record on where the segments of the channel are downloaded from, *i.e.*, from other peers or the server. The record is a vector $D_c = \{\kappa_{s_1}, \kappa_{s_2}, \ldots, \kappa_{s_i}, \ldots\}$ where $\kappa_{s_i} = 0$ indicates that segment s_i is downloaded from the server and $\kappa_{s_i} = 1$ from another peer. When a helping peer p has selected to assist in channel c and joined its overlay, the tracker assigns p a number of neighbors caching segments in the channel. Peer p asks these neighbors for their download records D_c's, and merges them to derive its local values of $Q_c^s(t)$ and $K_c^s(t)$, *i.e.*, the total numbers of copies of each segment s downloaded from the peers and the server, respectively, as defined in Sec. II-A2. As it is possible that none of its neighbors has downloaded a specific segment s, we allow peer p to further exchange download records with 2-hop neighbors, in order to maximally infer resource information on all the segments in the channel. Then the peer locally calculates a segment resource vector $S_c(t)$ for its helping channel c, based on which it carries out helping segment selection.

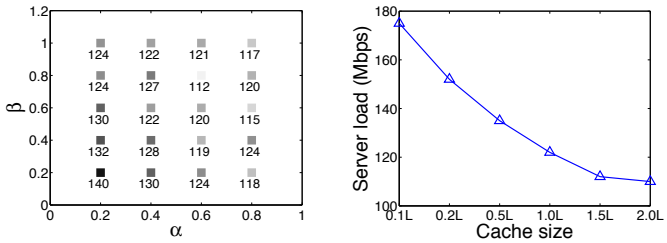
Such a distributed implementation can be practically integrated into real-world P2P VoD systems, as only a small number of additional message exchanges are required among the neighboring peers. Though the local segment resource vector is an approximation of one derived with global information, we will show with our experiments that it is able to achieve comparable satisfactory performance.

IV. Trace-driven Performance Evaluation

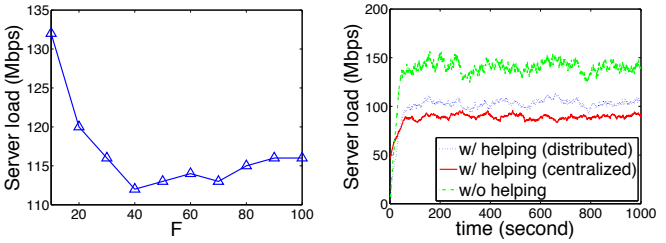
Our extensive evaluations of the strategies are driven by real-world traces, collected from the Orbit Network [5] over a 3-month span in 2009, which is a commercial P2P VoD system in China with thousands of concurrent online users. We use the following statistics from the traces in our experiments: (1) the zipf-like popularity distribution of 3000 channels in the system; (2) the poisson-like arrival patterns during regular times and during a flash crowd scenario; (3) peer session lengths, that a peer on average stays in a channel for $1/5$ of the video length; (4) the number of videos a peer watches before leaving the system, which follows a *Pareto* distribution with range 1 to 70 and shape parameter $k = 2$. The bitrates of videos in our experiments are 800 Kbps and the video durations are 900 seconds each. Each segment in a video has a fixed size of 16 KB. The average number of concurrent peers in our experiments is 600 during regular times, and 2500 during the flash crowd scenario. The average interval between two VCR commands issued by each peer is 5 minutes. Peers have heterogeneous upload (download) bandwidths which follow a Pareto distribution with range 512 Kbps to 10 Mbps (2 Mbps to 10 Mbps) and shape parameter $k = 3$. We set the length of each round T to 5 seconds. We evaluate distributed implementation of our segment selection strategy in the experiments, unless stated otherwise.

As the main purpose of our design is to maximally reduce server bandwidth consumption, we evaluate *server load* with our strategies, the average bandwidth consumption to serve all requests from the peers, under different parameter settings. We first investigate the impact of different threshold parameters α and β, applied to decide the helping states of peers. The numbers below the plotted points in Fig. 2 (a) represent the server load in Mbps. We observe a general trend that when α becomes larger, more peers are in the NH state (where they keep fetching segments to help others) and the server load becomes lower; a relatively large β around 0.8 achieves the lowest server load. We use the best pair of parameter values, $\alpha = 0.6, \beta = 0.8$, in the following experiments.

We then evaluate the impact of peer cache sizes, varying from $0.1L$ to $2L$, where L is the length of a video. Recall that

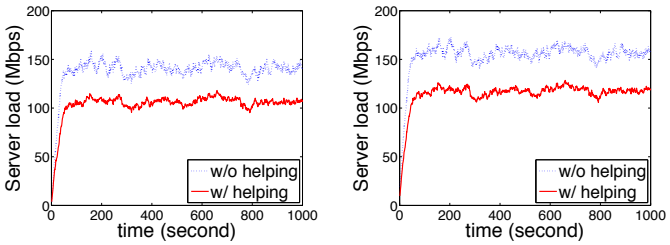


(a) α, β (b) Cache size

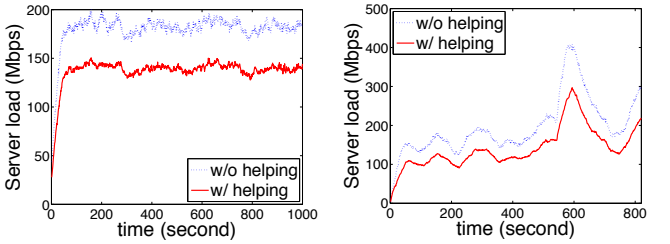


(c) F (d) Different algorithms

Fig. 2. Server load under different parameters and implementations.



(a) $M = 50$ (b) $M = 100$



(c) $M = 200$ (d) During flash crowd

Fig. 3. Server load with different channel numbers and during flash crowd.

we apply LRU cache replacement strategy. Fig. 2 (b) shows that the larger the peer cache is, the more significantly server load is alleviated. We have used $1.5L$ as the default peer cache size in the rest of our experiments.

We further evaluate the impact of the number of consecutive segments to be downloaded, F, in our helping segment selection strategy. Fig. 2 (c) shows that a small F leads to high server load, as few segments are consecutively cached in a helper peer and thus any viewing peer cannot download continuously from this helper. On the other hand, a large F makes the protocol less adaptive to the segment selection and the many segments may take too long to download. 40 segments have been shown to be the best choice resulting in the lowest server load. We adopt $F = 40$ in other experiments.

Next, we compare in Fig. 2 (d) the performance among distributed implementation of our strategies (with respect to helping segment selection), centralized implementation, and a native P2P VoD streaming scheme without cross- and intra-channel assistance. We observe a significant reduction of server load using our multi-channel collaborative strategies, as compared to the native one. In addition, our distributed implementation is only subjected to a 10% performance downgrade as compared to the centralized one, with much simplified implementation to be readily applicable in practical systems.

We next vary the number of concurrent channels (while the total number of peers in the system is maintained around 600), and investigate its impact on the performance of our distributed implementation and the native scheme, in Fig. 3 (a)(b)(c). We notice that with the increase of channels, both schemes incur higher server load. The reason lies that peers are more distributed into various channels with lower probability of mutual help. However, the increase of server load using our strategies is much lower than that of the native scheme. Especially during the flash crowd scenario shown in Fig. 3 (d), which simulates a real flash crowd captured by our traces, and our strategies save much more server bandwidth under the unexpected bursty arrival of users.

V. Concluding Remarks

This paper proposes effective strategies for both cross-channel and intra-channel collaborations in multi-channel P2P VoD systems, to maximize the utilization of individual peer's resources for the best streaming qualities in the entire system. Our strategies feature highly adaptive helping channel and segment selection at each peer, that dynamically determines the supply-and-demand imbalance among different segments in various channels, as well as fully active scheduling of each peer's surplus upload capacity, to serve the most needed segments without impairing its own streaming quality. Our distributed implementation of the strategies is evaluated using real-world P2P VoD traces and has been shown to be simple, effective, and practical to achieve our design objectives.

References

[1] B. Cheng, X. Liu, Z. Zhang and H. Jin. A measurement study of a peer-to-peer video-on-demand system. In *IPTPS*, 2007.

[2] T. T. Do, K. A. Hua, and M. A. Tantaoui. P2VoD: providing fault tolerant video-on-demand streaming in peer-to-peer environment. *IEEE ICC*, 2004.

[3] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? *ACM SIGCOMM*, 2007.

[4] Y. Huang, T. Z. Fu, D. M. Chiu, J. C. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. *ACM SIGCOMM*, 2008.

[5] Orbit Networks. "http://www.anyplex.com".

[6] PPLive. "http://www.pplive.com".

[7] A. Sharma, A. Bestavros, and I. Matta. dPAM: a distributed prefetching protocol for scalable asynchronous multicast in p2p systems. *IEEE INFOCOM*, 2005.

[8] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello. Push-to-peer video-on-demand system: design and evaluation. *IEEE JSAC*, 2007.

[9] UUSee. "http://www.uusee.com".

[10] M. Wang, L. Xu, and B. Ramamurth. Linear programming models for multi-channel p2p streaming systems. *IEEE INFOCOM*, 2010.

[11] D. Wu, C. Liang, Y. Liu, and K. Ross. View-upload decoupling: A redesign of multi-channel p2p video systems. *IEEE INFOCOM*, 2009.

[12] H. Yu, D. Zheng, B. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. *ACM SIGOPS*, 2006.

[13] H. Zhang, J. Wang, M. Chen, and K. Ramchandran. Scaling peer-to-peer video-on-demand systems using helpers. *IEEE ICIP*, 2009.