View-Upload Decoupling: A Redesign of Multi-Channel P2P Video Systems

Di Wu[†], Chao Liang[‡], Yong Liu[‡], and Keith Ross[†]
[†]Computer and Information Science [‡]Electrical & Computer Engineering
Polytechnic Institute of NYU, Brooklyn, NY, USA 11201
Email: dwu@poly.edu, cliang@photon.poly.edu, yongliu@poly.edu, ross@poly.edu

Abstract—**In current multi-channel live P2P video systems, there are several fundamental performance problems including exceedingly-large channel switching delays, long playback lags, and poor performance for less popular channels. These performance problems primarily stem from two intrinsic characteristics of multi-channel P2P video systems: channel churn and channel-resource imbalance. In this paper, we propose a radically different cross-channel P2P streaming framework, called View-Upload Decoupling (VUD). VUD strictly decouples peer downloading from uploading, bringing stability to multichannel systems and enabling cross-channel resource sharing. We propose a set of peer assignment and bandwidth allocation algorithms to properly provision bandwidth among channels, and introduce substream swarming to reduce the bandwidth overhead. We evaluate the performance of VUD via extensive simulations as well with a PlanetLab implementation. Our simulation and PlanetLab results show that VUD is resilient to channel churn, and achieves lower switching delay and better streaming quality. In particular, the streaming quality of small channels is greatly improved.**

I. Introduction

In recent years there have been several large-scale industrial deployments of P2P live video systems, including Coolstreaming [1], PPLive[2], and Sopcast [3], etc. Almost all live P2P video systems offer multiple channels. PPLive and its competitors each have over 100 channels; future user-generate systems will likely have thousands if not millions of live channels.

A. *Isolated-Channel P2P Video Systems*

A common practice in P2P video today is to organize peers viewing the same channel into a swarm, with peers in the same swarm redistributing video chunks exclusively to each other. We refer to such a design as "*isolated-channel*" P2P video systems. Recent studies of PPLive have identified several fundamental performance problems for isolated-channel systems:

Channel switching delay and playback lag. Measurement studies of PPLive and other live P2P streaming systems [4], [5] indicate that current channel-switching delays are typically on the order of 10-60 seconds. This is clearly undesirable, as users are accustomed to delays of under 3 seconds in current cable and satellite television systems and sub-second delays when switching pages on the Web. Furthermore, these measurement studies have shown that the playback lag, from when a live video frame is emitted from the source until it is played at the peer, wildly varies from one peer to another, with delays ranging from 5 to 60 seconds. Unfortunately, the BitTorrent-like mesh-pull architectures – currently used by most of the P2P video deployments – are inherently delay and lag prone.

Poor small-channel performance. In the upcoming years, we expect to see the emergence of *user-generated live channels*, for which any user can create its own temporary live video channel from a webcam or a hand-held wireless device. In the future, there will be, at any one time, thousands of such small channels, each emanating from a relatively low-speed connection (for example, wireless PDA) and each with 10-1000 viewing peers. Measurement work [4], [5] has revealed, however, that current P2P live streaming systems generally provide inconsistent and poor performance to channels with a small number of peers.

These performance problems primarily stem from two intrinsic characteristics of multi-channel P2P video systems: *channel churn* and *channel-resource imbalance*. In multi-channel P2P video systems, churn occurs on two different time scales: peers enter and leave the video application on long time scales; and peers change channels on short time scales. A recent study of a cable television system showed that users switch channels frequently [6]. Unfortunately, this channel churn brings enormous instability to the system. For example, when a peer switches from channel A to channel B, it stops uploading to its neighbors in swarm A. Those neighbors have to find new data feed to maintain steady video inflow. And in swarm B, the newly joining peer has to find new neighbors with enough bandwidth and content to download from, leading to excessive channel switching delays.

To understand channel-resource imbalance, recall that a channel's instantaneous resource index [7], [8] is defined as

$$\rho := \frac{u_s + \sum_{i=1}^{n} u_i}{nr} \tag{1}$$

where r is the bit rate of the channel, n is the number of peers viewing the channel, u_s is the server upload capacity, and u_i is the fraction of peer i's upload capacity devoted to the channel. A channel's instantaneous resource index indicates to what degree the upload supply matches the download demand for the channel. In particular, the streaming rate r is not sustainable if $\rho < 1$ for extended periods. In an isolated-channel design, many channels may have poor performance simply because they are resource-index poor. Moreover, channel switching and resource-index imbalance can conspire, leading to extremely poor performance for small channels.

B. View-Upload Decoupling

In this paper, we propose a radically different cross-channel P2P streaming framework, which we refer to as *View-Upload Decoupling (VUD)*. VUD strictly decouples what a peer uploads from what it views, bringing stability to multi-channel systems and enabling cross-channel resource sharing.

In VUD, each peer is assigned to one or more channels, with the assignments made *independently of what the peer is viewing*. For each assigned channel, the peer distributes (that is, uploads) the channel. This has the effect of creating a semi-permanent *distribution swarm* for each channel, which is formed by peers responsible for uploading that channel. This novel approach has two major advantages over isolated-channel designs:

- **Channel Churn Immunity.** VUD is *immune* to channel churn. With these more stable distribution swarms, a peer's viewing experience and channel-switching delay can be dramatically improved.
- **Cross-Channel Multiplexing.** VUD enables cross-channel resource sharing. In particular, distribution swarms can be properly provisioned and adapted in response to the evolving channel popularity and achieve a cross-channel "multiplexing gain".

However, decoupling viewing from uploading requires more upload bandwidth overhead. To minimize this overhead, we also propose a substream-swarming enhancement. With substream swarming, a peer in a distribution swarm only downloads a small portion of the video stream, called a substream, and uploads the substream to multiple viewers. This way, peers in distribution swarms act as bandwidth amplifiers. We will show that VUD combined with substream-swarming dramatically improves viewing and channel-switching performance without requiring significant upload-bandwidth overhead.

We make the following contributions in this paper:

- To the best of our knowledge, this is the first P2P live streaming design that strictly decouples peer viewing from uploading. It addresses two fundamental problems that exist in current P2P streaming system designs: channel churn and channel-resource imbalance.
- We propose a set of peer assignment and bandwidth allocation algorithms to realize the proper provision of bandwidth among distribution groups. It eliminates the problem of resource shortage in small channels. When channel popularity changes, the algorithms can adjust bandwidth allocation dynamically.
- We introduce a substream enhancement to reduce the bandwidth overhead incurred by decoupling of viewing and uploading. We provide an analytical study of bandwidth overhead of VUD using substreams.
- We evaluate the performance of VUD via extensive simulations. The simulation results show that VUD significantly reduces switching delay, chunk miss ratio and playback lags. Moreover, VUD greatly improves the streaming quality of small channels. We also perform a preliminary PlanetLab evaluation of VUD.

The remainder of this paper is structured as follows. Section II summarizes related work. The detailed design of our cross-channel streaming system is presented in Section III. In Section III-A, we analyze the bandwidth overhead of our design. In Section IV, we describe the simulation methodologies and results related to the experiments conducted to verify the performance. The paper is summarized in Section V.

II. Related Work

P2P video streaming has attracted lots of research activities in recent years. Existing P2P video systems fall into two categories: tree-based and mesh-based. Most of previous research work focuses on the design and improvement of isolated-channel P2P streaming systems, paying little attention to the optimization of multi-channel P2P streaming systems. In the multi-channel setting, there are few related published papers. In [9], Wu et al. proposed a server bandwidth provisioning algorithm to adjust the supply of server bandwidth to different channels dynamically. Liao et al. [10] introduced inter-overlay cooperation in their system called AnySee to balance the resources among channels, and optimize streaming paths. In [11], Gan et al. proposed a reputation-based incentive mechanism to stimulate peers with spare bandwidth in resource-rich channels to help peers in resource-poor channels. Although the above works consider the balance of bandwidth resource among channels, they cannot solve the fundamental performance degradation incurred by channel churn. Our design differs in that, by strictly decoupling peer viewing and uploading, we address both channel churn problem and channel-resource imbalance problem simultaneously.

III. View-Upload Decoupling with Substreams

We now describe more specifically VUD. Each video is divided into substreams (for example, to create K substreams for a video, the video source could assign every Kth constant-size chunk to a substream). As illustrated in Figure 1, the server divides the channel into K substreams. For each substream, there is a subset of peers, called a *substream distribution group*. Server only uploads each substream to one peer in each distribution group. Peers in the same distribution group upload and download the substream in P2P fashion. Finally, each viewer downloads all substreams from all distribution groups. Below we list some critical properties and observations about VUD:

1) The groups are semi-permanent, that is, the groups remain constant over medium time scales, and do not change as peers channel surf. However, the groups do evolve on a longer time scale to adapt to evolving channel popularity and peer churn.
2) If a peer is assigned to a distribution group for a substream, then it seeks to receive the complete substream. It redistributes the substream chunks to other peers in its distribution group and to peers outside the group that are currently viewing the corresponding channel.

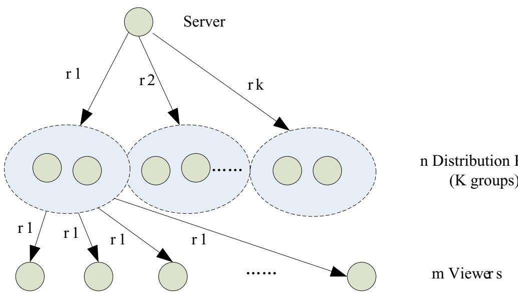


Fig. 1. Substream distribution groups in one channel: the server distributes one substream to each distribution group, which then distributes to the viewers.

3) A peer may belong to more than one distribution group, in which case it distributes more than one substream. If a peer is assigned to more than one subtream, it needs to allocate its upload bandwidth to its assigned substreams.
4) Intuitively, the aggregate upload capacity of a substream distribution group should reflect the demand for the substream. The greater the average channel demand, the larger the corresponding substream groups.
5) Intuitively, the peers in a distribution group should be chosen in regions that match the geographical demand. However, to expose the intrinsic advantages of VUD, we postpone locality considerations to subsequent work.
6) Within a distribution group, we do not require the deployment of a specific distribution mechanism. We allow for trees [12], mesh-pull [1] and meshes with push-pull [13].

A. Limiting VUD Overhead

One immediate concern for the VUD design is its bandwidth overhead. The peers in a substream distribution group need to first download video of their substream, either directly from the server or from other distribution peers in the same group, before they can upload the substream to viewers. Consequently, the aggregate download demand for each substream increases proportionally with the number of distribution peers in that substream. We define the *VUD overhead* of a channel as the ratio between the total bandwidth (from the server and distribution peers) utilized to upload video to distribution peers and the required upload bandwidth to serve the viewers of this channel.

To study the VUD overhead, we focus on one VUD channel with streaming rate r, m viewers and n distribution peers. Distribution peer i has upload capacity of u_i. Immediately we have the following result:

Proposition 1: The VUD overhead of any channel with m viewers has an achievable lower bound of $\frac{1}{m}$.

Proof: Please refer to our technical report [14]. ■

Essentially, to achieve the lowest VUD overhead, each channel is divided into many very fine substreams, each of which is distributed by a single distribution peer. However, for channels with a large number of viewers, it is impractical to have so many substreams. It is also unrealistic to have each distribution peer upload to all viewers. In practice, a channel is divided into a small number of substreams, each of which is distributed by a distribution group. The VUD overhead is determined by how substreams are divided and how distribution groups are formed. Next, we study how to design VUD to achieve a reasonably low overhead.

The VUD substream strategy will be characterized by the following variables for each substream:

- r_k: the substream rate for the kth substream group, with $\sum_{i=1}^{K} r_k = r$;
- $\mathcal{D}_k$: the group of distribution peers for substream k;
- $n_k = |\mathcal{D}_k|$: the number of distribution peers in $\mathcal{D}_k$, with $\sum_{i=1}^{K} n_k = n$;

Each distribution peer in $\mathcal{D}_k$ needs to download video at rate r_k. The total upload bandwidth consumed by all distribution peers in $\mathcal{D}_k$ is $n_k r_k$. Since the server only uploads one copy of substream k to one peer in $\mathcal{D}_k$, distribution peers will utilize $(n_k - 1)r_k$ of their upload bandwidth to disseminate the substream among themselves. To serve all viewers with substream k, distribution peers in $\mathcal{D}_k$ should upload to all viewers at an aggregate rate

$$m r_k \leq \sum_{i \in \mathcal{D}_k} u_i - (n_k - 1) r_k. \qquad (2)$$

The channel VUD overhead can be calculated as

$$\xi = \frac{\sum_{k=1}^{K} n_k r_k}{\sum_{k=1}^{K} m r_k} = \frac{1}{mr} \sum_{k=1}^{K} n_k r_k$$

Proposition 2: When peers have homogeneous upload capacity, the substreams need to be divided equally to achieve the minimum VUD overhead.

Proof: Please refer to our technical report [14]. ■

The minimum VUD overhead ξ^* decreases almost proportionally to peer uploading bandwidth and the number of substreams. For heterogeneous peers, the channel can also be divided into K equal-rate substreams. Distribution peers can be assigned to distribution groups to keep the average upload bandwidth within each group balanced, i.e., $\bar{u}^k \approx \bar{u}, \forall k$, where $\bar{u}^k$ is the average upload bandwidth in group k. Following the similar procedure as in the homogeneous case, it can be shown that the achieved VUD overhead is approximately $\frac{(m-1)r}{m(\bar{u}K-r)}$. If the system resource index is $\rho = 1.2$, the average distribution peer upload bandwidth is 1.2 times the streaming rate. It is sufficient to divide the channel into ten sub-streams to bring the VUD overhead down to 9%. VUD overhead will be *implicitly* bounded when we assign peers to achieve high resource index for all channels, we will discuss this in the following section.

B. Adaptive Peer Assignment

In this section, we present an adaptive algorithm that (i) assigns peers to substream groups; and (ii) for each peer, determines the fraction of upload capacity the peer assigns to each substream it is handling. Ideally, we would like such an assignment to balance the resource index across substreams and adapt to variations in channel demand. To this end, we introduce the following notation:

- K: the total number of substreams among all the channels;
- n: the total number of peers;
- r_k: the video rate of substream k;
- u_s: the upload capacity of servers;
- u_s^k: the server bandwidth allocated to substream k;
- u_i: the upload capacity of peer i;
- d_i^k: $d_i^k = 1$ iff peer i is in distribution group k;
- u_i^k: the allocated upload bandwidth of peer i for distributing substream k;
- u_{min}^k: the minimum upload bandwidth required to join distribution group k. (If $u_i^k \leq r_k$, peer i downloads more than uploads in group k. We require $u_i^k \geq 2r_k$);
- n_k is the number of peers in substream distribution group k, $n_k = \sum_{i=1}^{n} d_i^k$;
- m_k is the average number of peers viewing substream k over a short time scale (say, 5 minutes).

In assigning peers to groups, and allocating upload bandwidth to substreams, our primary goal is to balance the resource index across substreams. For a substream demand profile $\{m_k, k = 1, \ldots, K\}$, one can search for an optimal peer assignment $\{d_i^k\}$ and peer bandwidth allocation $\{u_i^k\}$ to balance the resource indexes $\{\rho_k\}$ across all substreams:

$$\max_{(\{d_i^k\},\{u_i^k\})} \min_k \rho_k = \frac{u_s^k + \sum_i u_i^k}{r_k(m_k + n_k)}, \tag{3}$$

subject to

$$d_i^k u_{min}^k \leq u_i^k \leq d_i^k u_i, \quad \sum_{k=1}^{K} u_i^k \leq u_i, \tag{4}$$

which reflects that a distribution peer's upload bandwidth constraints. The problem is a NP-hard mixed fractional problem. We will instead resort heuristic algorithm to solve it. As the average demands $\{m_k\}$ evolve, and as peers churn (on a much larger time scale than channel churn), we will need to adapt the assignments and allocations accordingly. A secondary goal is that, when adapting to peer churn and long-term channel popularity evolution, we more often adapt by modifying the allocations rather than re-assigning peers to groups.

To this end, order the substream groups in ascending order according to their resource indexes ρ_k. Let G be the maximum number of groups that a peer can initially be assigned to. (In our simulations, we use $G = 5$.) When a new peer i with upload capacity u_i joins the system, we need to assign it to substream distribution groups, and then allocate its upload bandwidth to the assigned substreams. We consider assigning i to the first G substreams in the ordered list (that is, to the G most resource-poor substreams). To this end, we use a water-filling policy and distribute u_i among the G groups to balance the resource indexes of the G groups. In doing the water leveling, we always make sure that the allocated bandwidth u_i^k to any group is never less than u_{min}^k.

Because peers can leave the system and the average demand for viewing channels can change, we may also need to adapt assignments and allocations inbetween peer arrivals. In particular, if the resource index of a substream drops below a threshold ρ_{min}, we adjust the peer bandwidth allocations without (if possible) modifying the peer assignments to distribution groups. With fixed peer assignment, the optimization problem defined in (3) $\sim$ (4) is simplified into a linear max-min programming problem. Using the Lagrangian relaxation method [15], it can be solved by distributed algorithms implemented on individual peers. In our current experiments, all peering connections are TCP connections. The bandwidth allocation among all neighbors of a peer is regulated by the TCP congestion control scheme. We will investigate the optimal distributed bandwidth allocation algorithm in future work.

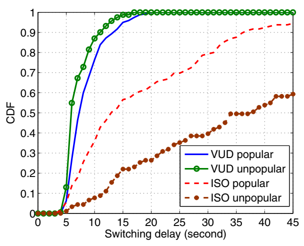
After bandwidth allocation adjustment, if some distribution groups still have resource indexes below ρ_{min}, we will have to adjust peer assignments. The basic idea is to shift nodes from resource rich groups to the groups whose resource indexes are below the threshold. First, we select the distribution group k with the highest resource index, and continuously move the distribution peers with the lowest upload bandwidth utilization from that group to a set P until the resource index of group k falls below ρ_{avg}. Then, we sort all the peers in P into a descending-ordered list based on their available upload bandwidth. After sorting, for each peer v in P, from the first to the last, we assign v to a distribution group j with the minimum updated resource index $\rho'_j = \frac{u_s^j + \sum u_i^j + u_v}{m_j r_j}$. Let H be the set of distribution groups with $\rho \leq \rho_{min}$. If the updated resource index $\rho'_j > \rho_{min}$, remove distribution group j from the set H. If there are still some resource-index poor distribution groups in H after all the peers in P have been assigned, we will choose the distribution group with the second highest resource index and continue the same process as above. The process will be executed until all the resource-index poor distribution groups have been removed from H.

IV. Simulation Experiments

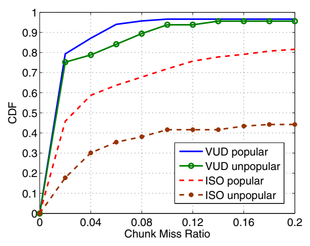
To evaluate the performance of VUD-based P2P streaming system, we implemented an event-driven P2P streaming simulator based on the source code provided by [16]. The simulator can simulate packet-level transmission, end-to-end latency among end nodes, node heterogeneity, peer churn and channel churn.

In the default simulation settings, there are totally 50 streaming channels and $2,000$ peers in the system. For each channel, there is only one server whose upload capacity is configured as 1 Mbps. All the channels have the same streaming rate of 400 Kbps. The video stream is further divided into 5 substreams for substream swarming. Within each distribution group, the distribution peers are organized into a mesh and use *Push-Pull* method for chunk scheduling. Viewers of a channel also use Push-Pull method to fetch chunks from all substream distribution groups of the channel.

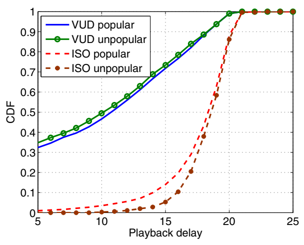
For comparison, we also implemented a Push-Pull isolated-channel P2P streaming system similar to GridMedia [13]. It serves as a baseline to evaluate the performance of VUD-



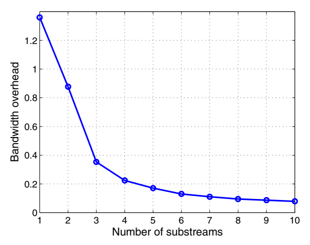
(a) CDF of Switching Delay



(b) CDF of Chunk Miss Ratio



(c) CDF of Playback Delay



(d) VUD Bandwidth Overhead

Fig. 2. Performance comparison between ISO and VUD design through simulations: a) the switching delays to both popular and unpopular channels under VUD design are much smaller than those under ISO design. b) VUD design achieves much lower chunk miss ratio than ISO design for both popular and unpopular channels. c) VUD design has smaller playback delay compared with ISO design. d) the bandwidth overhead of VUD design decreases as the number of substreams increases.

based system. In the following, the baseline system is referred as *ISO*, and our VUD-based system is referred as *VUD*.

The following metrics are used for the comparison: (1) *Switching delay*, the interval from one channel is selected until the initial buffer has been filled. (2) *Chunk miss ratio*, the number of chunks that miss the playback deadline over the total number of chunks that peer should receive. (3) *Playback delay*, the interval from a chunk is generated at the source node to the moment it is played at the peer. (4) *VUD bandwidth overhead*, the ratio between the total upload bandwidth utilized to upload video to distribution peers and the total upload bandwidth used to upload video to viewing peers.

In Figure 2(a), we plot the distribution of switching delay under VUD and ISO design. We use "*popular*" (or "*unpopular*") to indicate the delay when switching to a popular (or an unpopular) channel [1]. From the figure, we observe that, for both popular and unpopular cases, VUD design spends much less time in accumulating enough continuous chunks and have much shorter channel switching delay (mostly less than 10 seconds) than ISO design. Figure 2(b) compares the CDF of chunk miss ratio under two designs. Chunk miss ratio directly determines the playback continuity on peers. It is observed that, channel churn causes very high chunk miss ratio under ISO design. The situation becomes even worse for unpopular channels. In Figure 2(c), we present the CDF of playback delay for both VUD and ISO designs. Under VUD design, the playback delays of about 30% viewers are less than 5 seconds, and that of 70% viewers are less than 15 seconds. Under ISO design, the playback delays are much longer for both popular and unpopular channels. In Figure 2(d), we measure the bandwidth overhead of VUD design. It is important for the feasibility and efficiency of VUD design in real systems. By varying the number of substreams, we can reduce the bandwidth overhead to a rather low level.

[1]in the following, we use popular (or unpopular) similarly to indicate the performance in a popular (or unpopular) channel.

We also developed a prototype of VUD P2P streaming system and conducted an experiments on PlanetLab. The results are presented in our technical report [14].

V. Conclusion and Future Work

In this paper, we presented VUD, a novel streaming framework for multi-channel P2P video systems. By radically decoupling peer video uploading from viewing, VUD solves two fundamental performance problems of the traditional isolated channel P2P streaming, namely, excessively long channel switching delays and poor quality for small channels. We demonstrated through simulations and experiments on PlanetLab that VUD is immune to high channel churn and can efficiently achieve the multiplexing gain between channels with diverse popularity. In addition, we showed analytically and experimentally that VUD overhead can be well managed through balanced substreaming and peer assignment.

References

[1] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming," in *Proc. of IEEE INFOCOM*, Mar. 2005.
[2] "PPLive Homepage," http://www.pplive.com.
[3] "Sopcast Homepage," http://www.sopcast.com/.
[4] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, December 2007.
[5] X. Hei, Y. Liu, and K. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE Journal on Selected Areas in Communications*, December 2007.
[6] M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft, "On Next-Generation Telco-Managed P2P TV Architectures," in *Proc. of IPTPS*, 2008.
[7] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *Proc. of ACM SIGCOMM*, 2001.
[8] R. Kumar, Y. Liu, and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proc. of IEEE INFOCOM*, 2007.
[9] C. Wu, B. Li, and S. Zhao, "Multi-channel Live P2P Streaming: Refocusing on Servers," in *IEEE INFOCOM 2008*, Apr. 2008.
[10] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "AnySee: Peer-to-Peer Live Streaming," in *IEEE INFOCOM 06*, 2006.
[11] G. Tan and S. Jarvis, "Inter-Overlay Cooperation in High-Bandwidth Overlay Multicast," in *ICPP-06*, Aug. 2006.
[12] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proc. of ACM SOSP*, 2003.
[13] M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, and S. Yang, "A peer-to-peer network for streaming multicast through the internet," in *Proc. of ACM Multimedia*, 2005.
[14] D. Wu, C. Liang, Y. Liu, and K. W. Ross, "View-Upload Decoupling: A Redesign of Multi-Channel P2P Video Systems," *Technical Report, NYU-Poly*, 2008, http://eeweb.poly.edu/faculty/yongliu/docs/VUD1.pdf.
[15] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.
[16] M. Zhang, "Peer-to-Peer Streaming Simulator," http://media.cs.tsinghua.edu.cn/~zhangm/download/.