2009

A Cooperative Scheme for Dynamic Window Resizing in P2P Live Streaming

Zhipeng Ouyang
University of Nebraska - Lincoln

Lisong Xu
University of Nebraska - Lincoln, xu@cse.unl.edu

Byrav Ramamurthy
University of Nebraska - Lincoln, bramamurthy2@unl.edu

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2009 proceedings
IEEE International Conference on Communications, 2009. ICC '09.
Digital Object Identifier: 10.1109/ICC.2009.5198585
Publication Year: 2009 , Page(s): 1 - 5

A Cooperative Scheme for Dynamic Window Resizing in P2P Live Streaming

Zhipeng Ouyang, Lisong Xu and Byrav Ramamurthy
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588-0115, U.S.A.
Email: {zouyang, xu, byrav}@cse.unl.edu

***Abstract*—Due to their widespread popularity, Peer-to-Peer (P2P) live streaming systems have become a great challenge for Internet Service Providers (ISPs) as they consume huge amount of Internet bandwidth. By observing that different users may watch a channel with different window sizes, we propose a cooperative scheme called Partial Participation Scheme (PPS) in which different peers request a video stream at different rates based on their window sizes, and a subset of peers viewing the video stream using a small window work as helpers to forward extra data to help other peers using a large window. By reducing streaming rate received by small-window peers, the total amount of consumed bandwidth decreases without sacrificing users' satisfaction. PPS includes peer cooperative bandwidth allocation algorithms and neighbor maintenance mechanisms to achieve short resizing delay when a peer changes its window between different sizes. We evaluate the performance of PPS via a comprehensive set of metrics generated from extensive simulations. Our simulation results show that PPS greatly reduces the bandwidth consumption, achieves short resizing delay, and maintains high and stable streaming quality.**

I. Introduction

Several commercial P2P live streaming systems have been deployed, and some of them can support high-quality multimedia streaming to over 500,000 consumers simultaneously [6][10][12]. These widely deployed large-scale P2P live streaming systems also consume hung amount of Internet bandwidth. It is challenging and costly to upgrade the network to satisfy the increasing needs of a large number of users. So it is critical to find solutions to decrease bandwidth consumption of current P2P live streaming systems.

Some existing P2P live streaming systems already allow users to customize the window sizes when they are watching a program in those windows using a P2P live streaming software. The users also dynamically switch live streaming windows between full-screen mode and partial-screen mode based on their personal choices. Another case of users switching window sizes is observed in multi-channel applications. For example, PPStream [11] offers multi-channel function in which users can watch two channels with one in a large window and another in a small window simultaneously, like picture-in-picture in traditional TV. Users can arbitrarily choose which one is in the large/small window and change them at any time. We refer to these as $Dynamic\ Window\ Resizing$.

This work is partially supported by UNL Layman Award.

Compared to a full-screen window, a small window has a short width and height in terms of pixels, and thus a relatively low streaming rate is sufficient to provide the user with a satisfactory streaming quality. Furthermore, the channel shown in a small window is usually not as important for the user as the one shown in a large window, so a low streaming rate and coarse quality does not notably affect the user's satisfaction. Therefore, differentiating the required streaming rates at different peers according to their window sizes can greatly reduce the bandwidth consumption, while maintaining satisfactory streaming quality.

Motivated by the above observations, in this paper we propose a scheme called PPS: Partial Participation Scheme with which a P2P live streaming system consumes less bandwidth, while still being able to maintain high and stable streaming quality in case of dynamic window resizing. In contrast to most of the current P2P live streaming systems, small-window peers receive a lower streaming rate than large-window peers in PPS. To simplify the description, we choose layered coding [3] with two layers, the base layer and the enhancement layer, as the source coding technology, but our scheme can be easily extended to other coding methods such as layered coding with more than two layers and MDC [4]. In PPS, large-window peers receive both the base layer and the enhancement layer, while most small-window peers receive only the base layer and some small-window peers help in forwarding the enhancement layer data. Thus PPS consumes much less bandwidth and achieves both stable streaming quality and short resizing delay.

The remainder of this paper is organized as follows: Section II discusses related work. We present the problem in Section III. In Section IV, we describe the framework of PPS and an oracle-based centralized algorithm. Distributed PPS is described in Section V. Section VI includes experiment results, and Section VII provides the conclusion.

II. Related Work

Differentiating peer streaming rates has been discussed in some papers. Chakareski *et al.* [1] investigated the performance of implementations of MDC and of layered coding for video streaming. In [7] Liu *et al.* proposed an incentive mechanism based on layered coding, in which streaming rates received by different peers were determined by their

contribution. Padmanabhan *et al.* tried to make live streaming robust to peer transience through redundancy with MDC in [9]. In contrast to their work, we study how to reduce the total bandwidth consumption instead of encouraging peers to contribute more bandwidth. Cui *et al.* [2] studied the asynchrony of user requests and heterogeneity of peer bandwidth in P2P streaming solutions with layered coding. In [8] we proposed Partial Forwarding Scheme (PFS) to achieve low bandwidth consumption and short resizing delay in P2P live streaming systems. However, PFS blindly asks every small-window peer to forward the same amount of partial enhancement layer without considering their bandwidth heterogeneity.

III. Dynamic Window Resizing

As described in the previous sections, different users may have different window sizes when watching a channel, so we make peers require different streaming rates based on their window sizes to decrease the total bandwidth consumption. However, they may also change their window sizes from large to small or vice versa during the streaming. For example, some user reduces its window size and uses the remaining screen to surf the Internet when ads are inserted into the stream, and later recovers to a full-screen mode after ads are played. In multi-channel scenarios, a user may simultaneously watch multiple windows for multiple video streams, e.g. a user watches the live broadcast of a football game with multiple windows displaying different views of the game. He/she can dynamically enlarge one of them which is displaying the close-up of a view.

To simplify the description, we assume that there are two different viewing window sizes: large and small. For example, we can consider a full-screen window as a large window, and a partial screen window as a small window. Obviously, we can give more subtle definitions, such as 300*400 pixels, 600*800 pixels, 900*1600 pixels, and so on. Peers may arbitrarily resize their windows from large to small or vice versa, we call these events $Dynamic\ Window\ Resizing$, since a user still watches the same channel but with a different window size, and can dynamically change its window size at any time. $Dynamic\ Window\ Resizing$ has stringent requirements, as users are still watching the same channel and expect no or very little delay when resizing between different window sizes. Please note that we use the term peers to refer to both nodes and users at these nodes in the remainder of this paper.

In this paper, we study how to make P2P live streaming systems consume less bandwidth, while still achieving little resizing delay and stable high steam quality in case of $Dynamic\ Window\ Resizing$.

IV. PPS Framework

In this section, we propose a scheme called PPS to reduce total bandwidth consumed by P2P live streaming systems and fulfill stringent requirements on $Dynamic\ Window\ Resizing$. PPS achieves: 1) low total bandwidth consumption, 2) short resizing delay between different window sizes and 3) stable and high streaming quality.

The basic idea of PPS is shown in Figure 1. All peers request different streaming rates based on their window sizes, while some small-window peers called helper peers, forward the same stream like large-window peers. To differentiate helper peers from other ordinary small-window peers, we call these two types of small-window peers helper peers and non-helper peers, respectively. We use dashed lines to highlight the connections called heterogeneous connections which connect non-helper peers and large-window/helper peers, as the two end peers of these connections have different content. The other connections whose two end peers have the same content are called homogeneous connections. This dramatically reduces the total bandwidth consumed by the P2P live streaming systems without impairing users' satisfaction.

But due to dynamic window resizing events, reducing the streaming rate at small-window peers may cause long resizing delay in case of a small-window peer finding no available large-window neighbors when it resizes to a large window. Furthermore, the streaming quality of a large-window peer may become worse, if it cannot quickly find a good substitute for a former large-window neighbor that just resized to a small window. Hence we propose a set of algorithms which are used to relieve the impact of $Dynamic\ Window\ Resizing$.

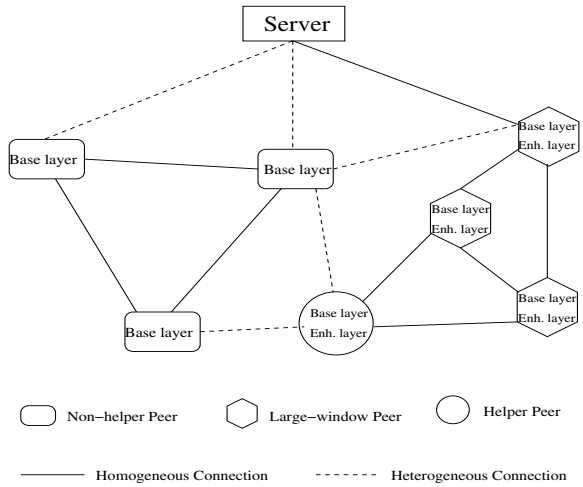

Fig. 1. In PPS, different peers with different window sizes receive different streaming rates, and helper peers are used to relieve the impact of $Dynamic\ Window\ Resizing$.

A. Formulation

To simplify our analysis, we consider an extreme case of a P2P live streaming system, where all peers are connected to the others . We model this extreme case with a complete graph graph $G = (V, E)$ where V is the set of vertices representing online peers and $E = V \times V$ is the set of connections. There is a server S in G, which generates a video stream at constant rate R including the base layer and the enhancement layer with streaming rates at R_B and R_E, respectively. Large-window peers require both the base layer and the enhancement layer

to display a high definition video, while only the base layer is used by small-window peers to display the video in their small windows.

Peers normally have asymmetric bandwidth - usually the download bandwidth is much larger than the upload bandwidth. They receive a stream at most at rate R which is relatively low compared with their download bandwidth, so here we only consider the impact of upload bandwidth. A peer $i \in V$, has an upload bandwidth of U_i. As aforementioned, peers are divided into three groups: large-window group, helper group and non-helper group and we denote them with $V_L(t)$, $V_H(t)$ and $V_S(t)$, respectively. Remember that the group membership is dynamic, and the groups are specified with respect to time t. The streaming rate required by peer i is denoted with $R_i(t)$, $R_i(t) = R_B$ or R. Let $N_i(t)$ be the set of neighbors of peer i. We define $r_{i,j}(t)$ as the rate that neighbor peer j streams to i. If peer i receives an aggregate incoming stream of $R_i(t)$ from its neighbors, i.e. $\sum_{j \in N_i(t)} r_{i,j}(t) = R_i(t)$, we say that peer i is fully served. Only fully served peers can playback the stream smoothly.

So in a P2P live streaming system which achieves high performance,

$$\sum_{i \in V(t)} U_i \geq (|V_L(t)| + |V_H(t)|) \times R + |V_S(t)| \times R_B \quad (1)$$

and

$$\sum_{j \in N_i(t)} r_{i,j}(t) = R_i(t), \quad \forall i \in V(t) \quad (2)$$

To focus on dynamic window resizing problem, we assume that Inequality (1) holds for any $V_H(t)$. If Inequality (1) does not always hold, this means the system does not support the worst case when all peers have large windows. If this happens, we need to take some actions, such as admission control, and this topic is beyond the scope of our paper. Furthermore, as G is a complete graph and every peer has connections to all other peers, we only need to make sure that the system has sufficient bandwidth for the base layer and the enhancement layer respectively compared to the demands, then we can ensure that Equation (2) holds for each peer.

As only the enhancement layer downloaded by helper peers is useless to the helper peers themselves, the consumed bandwidth which cannot directly benefit the stream quality can be measured by $|V_H(t)| \times R_E$. R_E is a constant value here, so if we can minimize the number of helper peers, the total bandwidth consumption is minimized. The other two metrics - high stream quality and low resizing delay, can be ensured by enough bandwidth for the enhancement layer in the system, because G is a complete graph and we assume that Inequality (1) always holds, peers can always get sufficient stream rate from neighbors - all the other peers. Thus our goal can be simplified as:

Minimize $|V_H(t)|$, subject to:

$$\sum_{i \in V_L(t) \cup V_H(t)} U_i \geq (|V_L(t)| + |V_H(t)|) \times R_E, \quad \forall t \quad (3)$$

B. An Oracle-based Centralized Algorithm

Based on the complete graph structure, we propose a centralized algorithm assuming the system can forecast dynamic window resizing activities. This algorithm also serves as a benchmark for the evaluation of our proposed distributed algorithm.

We can consider the server S as a central controller. At time t, S forecasts the enhancement layer bandwidth demand change in the next short period Δt, and checks if existing large-window and helper peers can still provide enough upload bandwidth to satisfy the demand at time $t + \Delta t$, or in other words Inequality (3) still holds. If Inequality (3) no longer exists at $t + \Delta t$, S asks some small-window peers to become helper peers, thus Inequality (3) also holds in the next Δt.

The resizing activities in the next Δt will cause one of the following three cases:

a) The demand becomes larger than the supply. In this case, server S needs to change some non-helper peers to be helpers to make Inequality (3) hold. For a chosen peer i, the left side increase of Inequality (3) is U_i, and the right side increase of Inequality (3) is a constant R_E. Obviously choosing those peers with highest upload bandwidth will minimize the amount of newly chosen helper peers.

b) The demand and supply are balanced. Nothing needs to be done.

c) The demand is less than the supply. Server S considers if it can reduce the number of helper peers while Inequality (3) still holds. Opposite to case a), helper peers with the lowest upload bandwidth are more likely to be chosen, so the number of helper peers is minimized.

V. Cooperation Based Distributed Algorithm

However, the complete graph as the overlay topology and accurate forecasting are impossible in practice. The centralized algorithm only serves as a benchmark to evaluate our practical distributed algorithm. In this section, we propose a distributed algorithm, in which peers form the mesh structure, exchange buffer maps with their neighbors and request packets from the appropriate neighbors based on their received buffer maps, like in many real P2P live streaming systems. This algorithm is scalable and follows the same principles as the centralized algorithm. The algorithm consists of three components:

A. Neighbor Selection

In order to receive the stream at its desired rate, a newly arriving peer has to look for appropriate neighbors. As described in Section IV, peers have both homogeneous connections and heterogeneous connections. Ideally peers can benefit from all these connections for conveying streaming data from neighbors in a timely manner, and heterogeneous connections ensure that the enhancement layer can be quickly fetched when non-helper peers resize to large windows.

In practice, a newly joining peer i contacts the rendezvous point S which keeps track of the online peers including their types. S returns a few of them to the newcomer as the neighbor candidates. As a new entrant, i will be served by its neighbors with their residual bandwidth, and even if i is a small-window peer, it will also try to find neighbors which can provide the enhancement layer to prepare for resizing activities. Different from Equation (2) and Inequality (3) used in the centralized method, we have the local forms as follows.

$$\sum_{j \in N_i(t)} u_j \geq R \tag{4}$$

$$\sum_{j \in V_l(t) \cup V_h(t)} u_j \geq R_E \tag{5}$$

$V_l(t)$, $V_h(t)$ denote the large-window peers and helper peers in i's neighbor set-$N_i(t)$ respectively, and u_j denotes peer j's residual bandwidth. To speed up the selection, peer i prefers peers with large residual bandwidth.

If the returned peers are not able to fully satisfy peer i, peer i requests them to provide their neighbor lists, and implement neighbor selection in these newly returned peers. After getting enough neighbors, peer i prefers to request the base layer from non-helper neighbors.

B. Resizing Activities

Because of neighbor selection, the resizing peer i itself can quickly resize to the desired window size no matter from small to large or vice versa. However, the resizing activity affects i's neighbors. Peer i needs to inform its neighbors, and this can be implemented by exchanging buffer maps or by explicitly sending messages. In order to make the rendezvous point have exact system information, it also needs to send a message to S indicating this resizing event.

After receiving a resizing message, S updates its records. Peer i's neighbors check to see if their streaming quality will be affected by this resizing event. If so, they immediately request the stream from other neighbors instead of i, and remove i from their neighbor lists. If the resizing activity does not affect neighbor j immediately, for example peer i forwards the stream to j at a low rate or just forwards the base layer to it. In these cases, peer j does not need to remove i right away, and this can be left to the neighbor maintenance procedure described next.

C. Neighbor Maintenance

Neighbor maintenance is critical to this algorithm, as short resizing delay and good playback continuity of peers is based on a good neighbor list. Peer i constructs its neighbor list and makes connections based on the residual bandwidth and the type of other peers. By maintaining a good neighbor list, peer i is able to gain required content at a satisfactory rate, and to recover fast from a resizing activity.

To maintain the neighbor list, we need to add one item into the buffer map - the residual bandwidth of a peer which is calculated by periodically comparing its upload bandwidth with its upload rate. After receiving buffer maps, peers check to see if their large-window/helper neighbors satisfy Inequality (5), so non-helper peers can get the enhancement layer immediately after they resize to large windows and large-window peers can substitute some former large-window neighbor which resized to a small window.

If insufficient upload bandwidth of neighbors is detected, peer i refreshes the neighbor list by contacting its neighbors. The contacted neighbors return their neighbor lists to i. From the neighbor lists, peer i tries to substitute some existing neighbors with some new peers in these lists to make Inequality (5) satisfied. If this step cannot solve the problem, peer i acts like server S in case a) of Sec. IV-B. Peer i chooses one non-helper peer with the largest upload bandwidth in the neighbor list to become a helper peer. If the chosen peer agrees to forward more content, it becomes a helper peer and informs its neighbors and the rendezvous point S as well. Neighbors and S update this peer's record.

If the peer finds that the large-window/helper neighbors have lots of residual bandwidth, it acts like server S in case c) of Sec. IV-B by informing relatively low bandwidth helper peers that they can stop forwarding the enhancement layer for it. When helper peers receive this, they record this information and use it to periodically calculate if the extra forwarding can be terminated without breaking the neighborhood's balance. If so, they become non-helper peers.

VI. Simulation Results

We carried out simulation experiments to evaluate the performance of our schemes as well as former proposed schemes called QFS (Quality-first Scheme) and PFS (Partial-forwarding Scheme) [8]. Here, in QFS, all peers receive both the base layer and the enhancement layer. In PFS, large-window peers receive both the base layer and the enhancement layer which provide high definition quality, while small window peers receive the base layer and $1/X$ portion of the enhancement layer. We only present the results of $1/X$ =1/16 here, as with this value, small-window peers do not forward too much extra data and the performance is acceptable. The details of QFS and PFS can be found in [8]. The oracle-based centralized scheme is also included to serve as the ideal case for comparison purpose.

In our experiments, we simulate a single channel P2P system consisting of up to 1000 peers with various upload bandwidths: 23% 1 Mbps peers, 46% 384 Kbps peers, and 31% 128 Kbps peers. The stream consists of one base layer, and only one enhancement layer with 1:2 ratio, which is similar to MPEG-4 FGS structure [5]. We use three metrics mentioned in Section IV to evaluate our schemes: upload bandwidth consumption, resizing delay and overall streaming quality.

To simulate dynamic window resizing, we make peers behave independently and resize from small windows to large windows after an exponentially distributed time with mean $1/\lambda$ and resize from large windows to small windows after

an exponentially distributed time with mean $1/\mu$. We control the ratio between large-window peers and small-window peers and resizing interval through adjusting the values of λ and μ. Because of page limitation, we only present the results of the case in which there are about 20% large-window peers and resizing interval is around 500 seconds here.

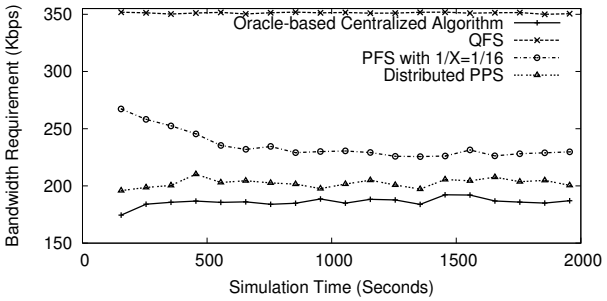

Fig. 2. Different schemes have different bandwidth demands.

Figure 2 shows the average bandwidth consumption for one peer. Compared with PFS and QFS, our PPS schemes consumes much less bandwidth because of smarter peer cooperation. QFS consumes most, and ideal centralized PPS consumes least. Different from PFS, in PPS small-window peers do not forward the enhancement layer until insufficiency of upload bandwidth for the enhancement layer is detected, and then some of them are chosen to become helper peers. This is why PPS consumes less bandwidth even compared to PFS with small $1/X$ value.

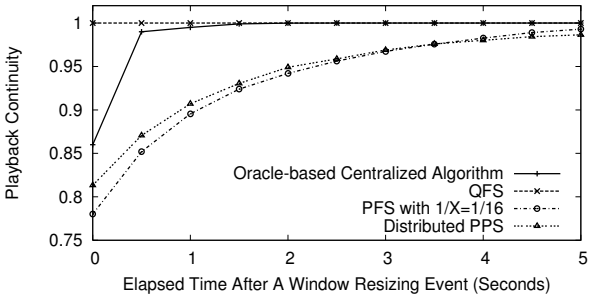

Fig. 3. In PPS, when a user changes its window size from small to large, there is a short delay for its playback continuity to reach a desired value.

Figure 3 shows resizing delay for peers to resize from small windows to large windows. From the former discussion, we can see that resizing delay exists only in cases of peers resizing from small windows to large windows. Obviously there is no resizing delay in QFS. With complete knowledge of the systems, small-window peers can directly obtain the enhancement layer from their existing neighbors after they resize to large windows. So the resizing delay for ideal cases is about 1 RTT (less than 0.5 second). Compared to PFS with a $1/X$ = 1/16, PPS with distributed algorithm achieves shorter resizing delay.

We present the overall system performance by measuring the average playback continuity in Figure 4. All of the schemes

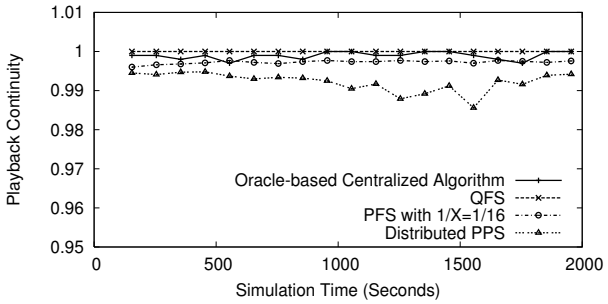

Fig. 4. PPS achieves high and stable playback continuity.

have high and stable playback continuity during the whole simulation. Although their playback continuities are a little different, we can ignore the differences as their values all stay above a high level (>99%). The results show that all the schemes look immune from dynamic window resizing at the whole system level. PFS and QFS use redundancy to make the enhancement layer be readily available all over the system, but in PPS, peers achieve this mostly by choosing appropriate neighbors.

VII. Conclusion

In this paper, we propose a cooperative scheme called PPS which greatly reduces the bandwidth demand of P2P live streaming systems but still achieves desired streaming quality in case of dynamic window resizing. Simulation results show that by by allowing peers to help each other, PPS performs well. However, we have not considered some problems here, such as the incentive mechanism, large-scale cooperation and so on. These will be explored in our future work.

References

[1] J. Chakareski, S. Han, and B. Girod. Layered coding vs. multiple descriptions for video streaming over multiple paths. In *ACM Multimedia 2003*, Berkeley, California, November 2003.

[2] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *Proceedings of the 13th ACM NOSSDAV*, Monterey, CA, June 2003.

[3] M. Ghanbari. Two-layer coding of video signals for VBR networks. *IEEE Journal on Selected Areas in Communications*, 7(5):771–781, June 1989.

[4] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, September 2001.

[5] T. Kim and M. H. Ammar. Optimal quality adaptation for scalable encoded video. *IEEE Journal on Selected Areas in Communications*, 23(2):344 – 356, 2005.

[6] J. Li. Peer-to-peer multimedia applications. In *Proceedings of the 14th annual ACM international conference on Multimedia*, Santa Barbara, CA, October 2006.

[7] Z. Liu, Y. Shen, S. S. Panwar, K. W. Ross, and Y. Wang. Using layered video to provide incentives in P2P live streaming. In *Proceedings of IPTV*, Kyoto, Japan, August 2007.

[8] Z. Ouyang, L. Xu, and B. Ramamurthy. A partial forwarding scheme for dynamic window resizing in live P2P streaming systems. In *Proceedings of IEEE Globecom*, New Orleans, Louisiana, November 2008.

[9] V. Padmanabhan, H. Wang, and P. Chou. Resilient peer-to-peer streaming. In *Proceedings of ICNP*, Atlanta, Georgia, November 2003.

[10] PPLive. http://www.pplive.com.

[11] PPStream. http://www.ppstream.com.

[12] X. Zhang, J. Liu, B. Li, and T. Yum. DONet/CoolStreaming: A data-driven overlay network for live media streaming. In *Proceedings of IEEE INFOCOM*, Miami, Florida, March 2005.