Climber: An Incentive-based Resilient Peer-to-Peer System for Live Streaming Services

Kunwoo Park*, Sangheon Pack†, and Taekyoung Kwon*

*School of Computer Science and Engineering, Seoul National University, Korea
Email: kwpark@mmlab.snu.ac.kr and tkkwon@snu.ac.kr

†School of Electrical Engineering, Korea University, Korea
Email: shpack@korea.ac.kr

***Abstract*— Due to the explosive popularity of peer-to-peer (P2P) file-sharing services, there have been remarkable research efforts on the P2P live streaming services. P2P systems are cost-effective and can be easily deployed only by leveraging the participating peer's resources (i.e., upload link bandwidth) to distribute contents. In this paper, we propose $Climber$, an incentive-based resilient P2P system for live streaming services. Climber encourages each peer to provide more upload link bandwidth to the system, and embodies an incentive mechanism that improves resilience under high churning rate. The structure of Climber is a hybrid of a tree and a mesh, which self-improves and adapts to network churning rate. Simulation results are given to evaluate the performance of the proposed protocol.**

I. Introduction

Recently, live streaming services over the Internet for entertainment (e.g., sports game, music concert) are becoming popular. For example, Akamai [1] provides an infrastructure support for these services, and AOL broadcast [2] and MSN [3] serve as streaming portals. These services leverage infrastructures (i.e., server-farm based solutions or content distribution networks) where contents are replicated on a number of servers to improve the speed of content delivery. However, infrastructure-based live streaming has some drawbacks in cost, scalability, and deployment. Live Earth concert in July 2007 on MSN broadcast service rated 15 million streams, with 237,000 simultaneous viewers at the peak [3]. Akamai also reports an aggregate traffic of 200Gbps during peak traffic periods through 20,000 servers in 71 countries [1]. Unlike these services, which requires costly infrastructures, peer-to-peer (P2P) live streaming is gaining much attention in the literature ([9] [10] [15] [6] [14]) because of its scalability, low cost, and tactical deployment. In P2P live streaming, all peers participate in distribution of contents by sharing their resources (i.e., upload link bandwidth).

P2P live streaming solutions can be classified into two categories: tree-based and mesh-based approaches. The tree-based approach implements a single or multiple distribution trees, rooted at the source of the stream. In a tree, each node always receives streaming packets from a parent and hands down to its child nodes. Due to the well-defined "parent-child" relationships, the signaling overhead is marginal. However, its performance can be severely degraded as the churning rate increases. Narada [9] is one of the first protocols targeting end-system streaming applications. Narada builds a source-rooted minimum-delay tree on top of a mesh topology that connects the participating peers. It is used in End System Multicast [13], which is a on-line service for P2P live streaming. SplitStream [10] operates by striping the stream into k stripes and forwards each stripe via one of separate k trees built using Scribe [11]. By maintaining multiple distribution trees, SplitStream reduces the impact of node failure on the rest of the system.

On the other hand, in the mesh-based approach, a peer has direct connectivity to neighbor peers. Each peer pulls a number of chunks from a subset of the neighbor. Control messages are exchanged among the peers in order to locate and pull the chunks throughout the mesh. Since each peer relies on a subset of the neighbor peers to receive streaming packets, this approach offers better resilience to membership dynamics than the tree-based approach. However timely delivery to all the peers and efficient management of large buffers are difficult tasks since the stream packets are generated by a live source at short intervals. Peers in Coolstreaming [15] and Chainsaw [14] maintain a list of neighbors, and periodically exchange data availability information with the neighbors. Each peer notifies neighbors of data arrivals and employs a pulling mechanism to receive chunks.

We focus on two challenging issues in P2P live streaming for both approaches. First of all, the way to encourage peers to contribute their resources for data distribution should be devised in order to improve live streaming services in terms of delay and resilience. The other issue is dynamic membership, i.e., each peer dynamically joins or leaves (peers easily fail or leave at will) a P2P network for live streaming. Therefore, resilience to churning rate should be considered.

In this paper, we propose a novel P2P live streaming system, $Climber$, whose structure is a hybrid of a tree and a mesh. That is, a single tree is constructed for streaming; however, random edges between peers (not between parent and child) are added to the tree for redundant connectivity. In addition, the level of resilience provided to a peer is proportional to its contribution in data distribution. The contribution of a peer is defined as the number of outdegree of the peer, which is typically proportional to the link bandwidth allowed. This incentive mechanism encourages nodes[1] to assign more bandwidth for data distribution, which leads to better (system-wide) performance. To tackle the membership dynamics, Climber simply attaches a new joining node to the tree as a leaf node. When a non-leaf node leaves or fails, Climber minimizes the effect of peer departure by gracefully relocating its descendants to other positions of the tree.

Our contributions in this paper are summarized as follows: 1) we propose a simple and resilient P2P live streaming protocol, which is self-improvable and adaptive to network conditions with reasonably low protocol overhead; 2) the proposed protocol embodies an incentive mechanism that enhances resilience despite churning rate. To the best of our knowledge, this is the first work considering incentives to construct resilient P2P networks.

The rest of this paper is organized as follows. Design concepts and the operations of Climber are introduced in Section II. Section III gives simulation results, followed by concluding remarks in Section IV.

II. CLIMBER

Climber has the following design concepts.

- **Incentive**: Climber gives more incentive to a peer of more contribution. Approximately the number of potential data delivery paths from a source to a peer is determined in proportion to the outdegree of the peer. Hence, a peer will experience more resilient service as the peer forwards packets to more other peers.
- **Resilience**: Since each peer may potentially fail, Climber utilizes a randomized forwarding technique motivated from [4] to augment the tree structure. Climber not only uses the randomized forwarding for seamless streaming, but also uses a randomized pulling technique to improve the system resilience.
- **Self improvement, adaptation**: When a peer finds an alternative path that is faster than the current delivery path from the root, the peer switches its link to the parent to the alternative path. Each peer keeps trying to find a better path. This switching process incrementally improves the overall root-to-peer delay of the system. Also, Climber adaptively changes its topology according to the current network condition to maintain a predefined level of resilience.

A. System Model

The structure of Climber is based on a single-tree, rooted at the source, but more than one data delivery path for each peer may exist. We focus on P2P live internet broadcasting applications where 10 or 20 seconds of delay is tolerable, which is normally assumed in the literature ([5], [6]). Climber does not give any solution for packet losses. Redundancy (e.g., Forward Error Correction) or retransmission technique can be added to the stream in order to recover the missing packets.

Peer i has O_i ($O_i \geq 0$) outdegree, which means peer i currently forwards packets to O_i peers. We assume that a peer's incoming link bandwidth (used by its parent to send packets to the peer) is always enough and the bottleneck resource is the outgoing link bandwidth in P2P streaming. O_i^{max} ($O_i^{max} \geq 1$) is the maximum outgoing edges permitted to use by the peer. A peer can adjust the maximum possible amount of contribution to the system by setting/updating O_i^{max}.

A root node generates a series of streaming packets and tags each packet with a sequence number in an incremental order. Therefore, we can assume a packet with a higher sequence number is the packet generated more recently. Seq_i is the sequence number of an arriving streaming packet from a node's parent node i. In Climber, parent and child nodes exchange heartbeat messages at relatively long intervals to detect a path failure. The heartbeat message contains the number of descendants and therefore a root node can roughly estimate the total number of nodes in the system by the message.

[1]We use the term "node" when a peer joins a system and has parent/child relationship. However, the term "node" and "peer" may be used interchangeably.

B. Topology Construction

To join Climber, a peer sends a Join message to the root. If node i (including the root) receives a Join message and the node has a remaining outdegree ($O_i < O_i^{max}$), it takes the new peer as a child node by the probability $1 - O_i/O_i^{max}$. Otherwise, the Join message is randomly forwarded to one of its child nodes. If a leaf node receives the Join message, it should take the node as a child node since $O_i = 0$ and $O_i^{max} \geq 1$.

After receiving streaming packets, a node randomly selects a number of other participating peers (selected peers are called "prospective child nodes") to establish random edges. The random edges are constructed as follows. For each available outdegree ($O_i^{max} - O_i$), node i makes a decision whether to make a random edge or not with probability λ_t at time t. The target node of the random edge is randomly selected from all the participants. As node i just joined, O_i is 0. λ_t is sent from the root node piggybacked in the streaming packets. After then, the node forwards only the sequence numbers of the streaming packets to its selected prospective child nodes as soon as it receives stream from its parent. A root node keeps a list of recently joining nodes and each node obtains the list from the root node when it joins. When a random edge is established, the lists are exchanged so as to maintain the up-to-date list. How the root node determines λ_t will be elaborated in Section II-D.

C. Handling Churn

When a node i has a parent node j and a random incoming edge from node r (r is called a "prospective parent"), it compares Seq_j and Seq_r. If the sequence number from its parent is lower than the one through the random edge ($Seq_j < Seq_r$) [2], it indicates that the current delivery path is slower than the path via the random edge. Then node i changes its parent from node j into node r, and hence old parent-child link is broken. Furthermore, if $O_i < O_i^{max}$, node i immediately sets up an outgoing random edge to its old parent j. If the sequence number from the parent is higher than the one from the prospective parent, no actions are taken. To adapt to membership dynamics and internet topology changes, each peer reestablishes the random edge(s) periodically to other nodes.

Figure 1 shows how a node "climbs" a tree by a random edge. Let us assume node 1 is the source and the network around node 3 is congested. (a) When node 2 establishes a random edge r_2 (shown as a dotted line) to node 7, node 7 compares Seq_6 from its parent node 6 and Seq_2 from node 2. (b) If $Seq_6 < Seq_2$, which means the current delivery path ($1-3-6-7$) is slower than the path ($1-2-7$), node 7 changes its parent from node 6 into node 2. Suppose $O_7 < O_7^{max}$, node 7 sets up a random edge r_7 to node 6. (c) Node 6 also detects $Seq_3 < Seq_7$ and switches its parent into node 7 and establishes r_6 to node 3. If $Seq_1 > Seq_6$, no further action is taken. We call our system "climber" since node 7 $climbs$ the tree by being a child of node 2.

The random forwarding technique greatly simplifies the recovery procedure caused by node departures or failures. We do not need a fast failure detection mechanism (usually done with heavy signaling overhead [6]) or a failure recovery (proactive [16] or reactive [10]) method additionally. Instead, parent and child nodes need to exchange heartbeat messages at relatively long intervals. Actually, Climber does not distinguish node departure or failure from node congestion. Again, Figure 1 successfully describes the recovery procedure of Climber. (a) When node 3 fails, its descendent nodes $6, 7, 8$ (node 3's subtree) notice the current delivery path is congested. (b) At that moment, if there exists at least one random edge (r_2) established from the outside of the subtree to the inside of the subtree, node 7 with the random incoming edge switches its parent to node 2. Then node 7 forwards the stream to all the remaining edges including parent 6 (through a newly established random edge r_7) and child nodes as it has available outdegree. (c) Finally, the failed node 3 is placed at the end of delivery path ($1-3$) and deemed as it left the tree eventually.

If a node has no incoming random edges and its parent fails, then the node detects there is no parent alive by heartbeat messages. We say the node is $interrupted$, and the node sends a Rejoin message to the root node, which is the same as the joining process. The number of Rejoin messages is limited by the number of current outdegree of the failed node since only the direct child nodes send Rejoin messages to the root node.

This periodical $climbing$ technique also keeps improving the performance of the structure even the system experiences the high churning rate. For the nodes who do not have random incoming edges, Climber provides randomized pulling to request a randomly selected node to setup a random edge to the requesting node. Then the selected node establishes a random edge to the requesting node by the same procedure described in Section II-B.

[2] In our simulations, we use $Seq_j + \alpha < Seq_r$, where α is a predefined threshold to avoid oscillation, instead of $Seq_j < Seq_r$.

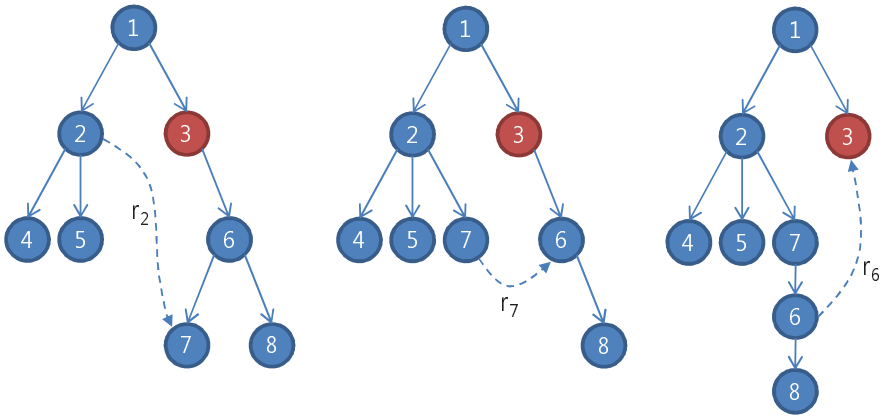

Fig. 1. Tree reconstruction steps using random edge. Node 7 *climbs* the tree by being a child of node 2.

D. Adaptation

Climber provides fine-grained control over the tree, i.e., λ_t indicates the level of structural resilience at time t of the system operation. To make a protocol more agile to live streaming, the service provider should have a way to adjust the level of QoS, e.g., the interruption rate (the proportion of interrupted nodes in the system) γ of the stream. In Climber, a source node compares a predefined interruption rate γ and γ_t, the latter of which is the number of received Rejoin messages out of the total number of nodes in the system at time t. If the churning rate increases γ_t will exceed γ. Then, the source node increases λ_{t+1} to enhance the resilience of the tree. Otherwise, the source node decreases λ_{t+1} to save network bandwidth by reducing the number of random edges. In Climber, a source node measures γ_t every 10 seconds and λ_{t+1} is derived as follows.

$$\begin{aligned} &\text{If } \gamma \geq \gamma_t, \text{ then } \lambda_{t+1} = max(\lambda_t - 0.01, 0.01) \\ &\text{Else if } \gamma < \gamma_t, \text{ then } \lambda_{t+1} = min(\lambda_t + 0.01, 1) \end{aligned} \quad (1)$$

Equation (1) is an example to adjust λ_{t+1}. Other solutions (e.g., linear increase and multiple decrease of TCP window) may be used depending on the characteristics of the service or streaming content.

E. Giving Incentives

Climber gives more incentive to a highly contributing peer in the sense that a peer that makes higher contribution will have more descendants, which implies more incoming random edges probabilistically. The level of contribution of a peer is defined as the current number of outgoing links. Note that time during which the node has been participating in the tree also affects the level of contribution. The number of descendants of a node tends to increase with the node's outdegree and participating time since more attempts to make prospective child nodes have been made via outgoing random edges by the node and its descendants. Since Climber is designed to recover from failures by using random edges, a node with a larger number of descendants will experience less streaming interruptions. Through this mechanism, Climber encourages nodes to use more upload link bandwidth to its child nodes and prospective child nodes, which leads to better performance of the P2P streaming service.

III. Simulation Results

To evaluate the performance of Climber, we have developed an event-driven simulator. The network topology is generated by the GT-ITM [7] topology generator that includes 2000 peers and 600 routers, using the Transit-Stub graph model. The topology consists of 3 transit domains with 8 transit routers each. There are an average of 3 stub domains per transit router, and an average of 8 stub routers per stub domain (3x8x(1+3x8)=600). Peers are randomly connected to the 576 stub routers. Link delays for the simulation is derived from [8]. We use link delays ranging from 1 to 55 ms for transit-transit or transit-stub links and 1 to 10 ms delay for a link within a stub. Each link is a symmetric link without packet loss. Only peers join and leave the system and every peer departure is regarded as a node failure. When a peer leaves the system, the peer joins the system (at a different position in the tree) immediately again so the number of peers in the system remains unchanged. The maximum outdegree O_i^{max} of peer i is randomly chosen between 1 to 10 ($1 \leq O_i^{max} \leq 10$). Peers reestablish their random edges in every 10 seconds.

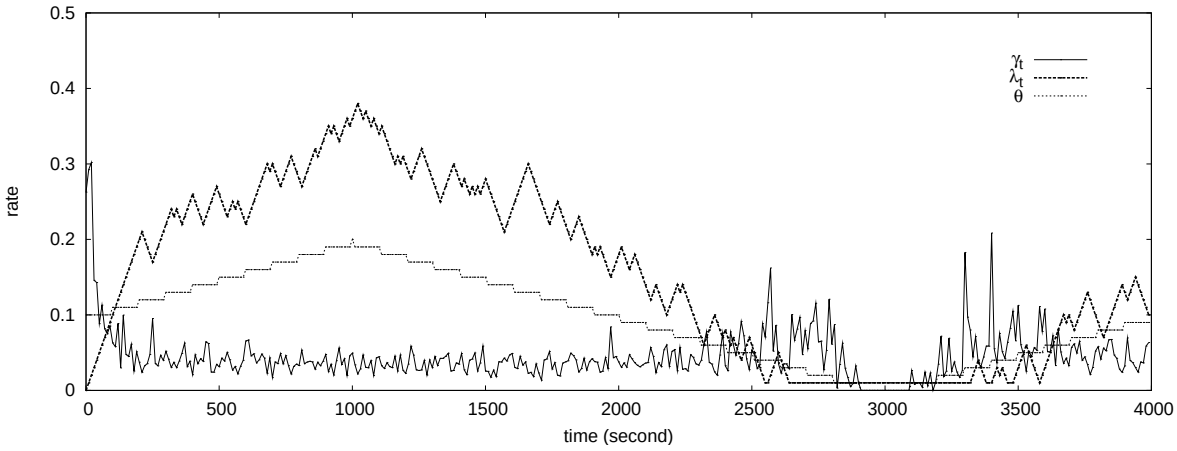

Fig. 2. Climber successfully adapts network condition by λ_t

A. Adaptation

To make a protocol more agile to live streaming, Climber provides a way for the service provider to adjust the level of QoS, e.g., the interruption rate γ of the stream. Climber periodically checks γ_t, the currently measured interruption rate at time t, and adaptively changes the random edge probability λ_t by Equation (1) in Section II-D. Figure 2 shows the adaptation behavior of Climber. The simulation is run over 4000 seconds and λ_t is recalculated every 10 seconds. In this scenario, γ is set to 0.03 which indicates less than 60 interrupted nodes out of 2000 nodes within 10 seconds would be acceptable. Churning rate θ is defined as the proportion of failed nodes out of the total number of nodes in the system. We intentionally vary the churning rate θ, i.e., the proportion of failed nodes in the system within the last 10 seconds between 0 and 0.2 (400 nodes fail in 10 seconds) as the simulation time goes by. To meet the constraint $\gamma = 0.03$, Climber enforces nodes to establish more random edges until λ_t becomes almost 0.4 when network condition gets worse (around 1000sec in the figure). γ_t spikes more sharply as λ_t decreases because the number of generated random edges declines as λ_t decreases.

B. Incentive

Climber provides more resilient streaming service to the nodes of higher contribution, i.e., a node that maintains more descendant nodes. Figure 3 shows the impact of our incentive mechanism. The figure is derived from the same experiment of Section III-A. Let $Inter(O = i)$ denote the number of interruption a node experiences when its current outdegree is i ($0 \leq i \leq 10$). Then relative interruption rate of a node of outdegree i is defined as $\frac{Inter(O=i)}{\sum_{j=0}^{10} Inter(O=j)}$. Leaf nodes (without a child)

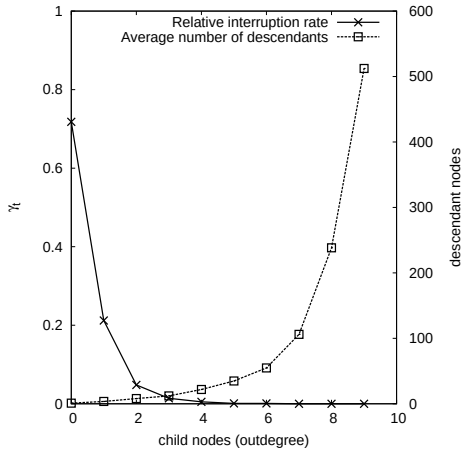

Fig. 3. Relative interruption rate and average number of descendants according to the number of outdegree(child)

experience 70% of the total interruption events while nodes with more than 3 child nodes rarely experience interruption. This is because the average number of descendants of a node increases exponentially as outdegree increases and hence the node is likely to get over the failure/departure of its ancestor more seamlessly as the number of descendants increases. Thus Climber strongly stimulates nodes to dedicate more upload link bandwidth to the system.

C. Overhead

Climber provides resilience at the cost of adding random edges. Figure 4 shows λ_t required to satisfy the level of QoS (γ) depending on different θ values. When the target resilience (γ) is 0.04 and 8% of nodes fail per time unit of 10 seconds, each node should dedicate 10% of its residual outdegree ($O_i^{max} - O_i$ in node i) to establish random edges. In high churning

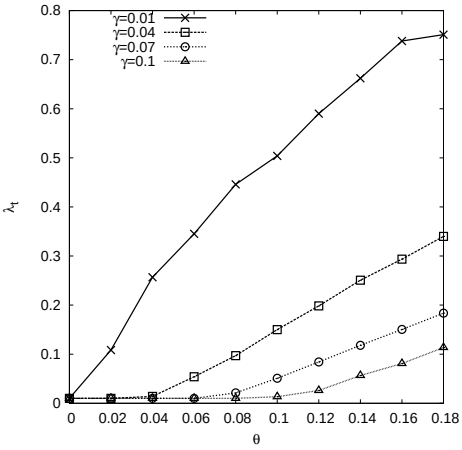

Fig. 4. λ_t required to satisfy the level of QoS (γ) in a certain network condition

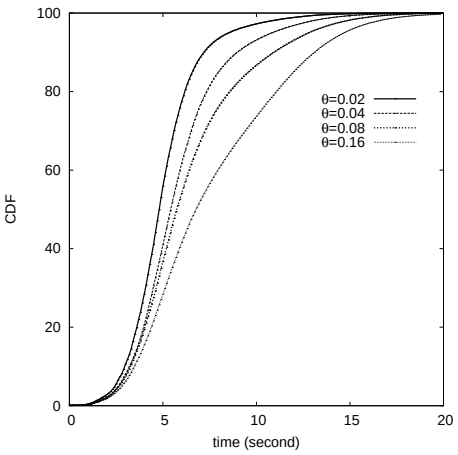

Fig. 5. CDF of delay from root node

state ($\theta = 0.18$), λ_t increases up to 75% to maintain interruption rate under 1%.

D. Delay

Figure 5 is a cumulative distribution function of the delay from a root node to a non-root node. We fixed the value of $\lambda_t = 0.04$ to see the delay characteristics of Climber. When $\theta = 0.02$, 97% of streaming packets are delivered within 10 seconds. As θ increases, i.e., churning rate increases, the average delay from the root becomes longer. Only 80% of the nodes receive packets within 11 seconds when $\theta = 0.16$. In this case, we should increase λ_t to enhance resilience, which in turn improves the delay.

IV. Conclusion and Future Work

This paper introduces Climber, an incentive-based resilient P2P system for live streaming services. It focuses on two challenging issues in P2P live streaming, namely, incentive and resilience. It is demonstrated that Climber gives more incentive to a highly contributing peer in the sense that a peer which makes higher contribution will experience more resilient streaming service. Moreover, Climber adapts successfully to a predefined level of resilience in the presence of high churn. Consequently, Climber is a viable solution for P2P streaming service providers who want to adjust the QoS level. As future work, we will investigate a relation between the proportion of received Rejoin messages out of the total number of nodes in the system (γ_t) and random edge probability (λ_t) in a systematic fashion. Furthermore, we are planning to implement Climber and carry out experiments on PlanetLab.

Acknowledgment

This work was supported by the Seoul R&BD Program funded by the Seoul Metropolitan Government, and the ICT at Seoul National University provided research facilities for this study.

References

[1] www.akamai.com
[2] http://press.aol.com/index.cfm
[3] http://liveearth.msn.com
[4] S. Banerjee, S. Lee, B. Bhattacharjee, A. Srinivasan, "Resilient Multicast Using Overlays," *IEEE/ACM Transactions on Networking*, vol. 14, no. 2, pp. 237-248, April 2006.
[5] W. Montgomery, "Techniques for Packet Voice Synchronization," *IEEE Journal on Selected Areas on Communications*, vol. 1, no.6, pp. , Dec. 1983.
[6] V. Venkataraman, P. Francis, and J. Calandrinoz, "Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast," in *Proc. IPTPS 2006*, February 2006.
[7] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proc. IEEE Infocom 1996*,
[8] Sonia Fahmy and Minseok Kwon, "Characterizing Overlay Multicast Networks and Their Costs," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 373-386, April 2007.
[9] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456-1471, Oct. 2002.
[10] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. ACM SOSP 2003*, 2003.
[11] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The Design of a Large-scale Event Notification Infrastructure," in *Proc. NGC 2001*, November 2001.
[12] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang, "The Impact of Heterogeneous Bandwidth Constraints on DHT-Based Multicast Protocols," in *IPTPS 2005*, 2005.
[13] http://esm.cs.cmu.edu/
[14] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *IPTPS 2005*, 2005.
[15] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming," in *Proc. IEEE Infocom 2005*, 2005.
[16] J. Byers, M. Luby, and M. Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1528-1540, Oct. 2002.