Computer Networks 54 (2010) 1316–1327

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

An adaptive peer-to-peer live streaming system with incentives for resilience ☆

Kunwoo Park [a], Sangheon Pack [b], Ted "Taekyoung" Kwon [a,*]

[a] School of Computer Science and Engineering, Seoul National, Republic of Korea
[b] School of Electrical Engineering, Korea University, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Available online 11 November 2009

Keywords:
P2P live streaming
Resilience
Incentive
Self-improvement
Adaptation

ABSTRACT

Recently, there have been a lot of research efforts on peer-to-peer (P2P) live streaming services. P2P systems can be easily deployed since a participating peer's resources (i.e., upload link bandwidth) can be exploited to distribute contents. However, how to adapt to leaving peers and how to encourage peers to contribute resources voluntarily are still challenging issues. In this paper, we propose Climber, an adaptive P2P live streaming system with incentives for resilience. Climber is based on the hybrid structure of a tree and a mesh, so as to achieve self-improvement and adaptation to users' dynamic joining and leaving. Moreover, Climber substantiates an incentive mechanism that provides better resilience for peers with more upload bandwidth allocated. Simulation results reveal that Climber significantly reduces the topology maintenance cost compared to SplitStream and NICE-PRM. Also, simulation and analytical results verify that Climber can bound the level of disruption by dynamically adapting to the user churning rate.

1. Introduction

Recently, live streaming services over the Internet (e.g., sports game or music concert) are becoming popular. For example, Akamai [2] provides an infrastructure support for these services; AOL broadcast [3] and MSN broadcast [4] serve as streaming portals. Live Earth concerts in July 2007 on MSN broadcast service recorded 15 million streaming sessions, with 237,000 simultaneous viewers at the peak [4]. These services leverage infrastructures (i.e., server-farm based solutions or content delivery networks (CDNs)) where contents are replicated on a number of servers to improve the speed of content delivery. However, infrastructure-based live streaming systems have some drawbacks in terms of cost, scalability, and deployment. Akamai reports an aggregate traffic of 885 Gbps during peak traffic periods through 30,000 servers in 67 countries [5].

Unlike these infrastructure-based services, peer-to-peer (P2P) live streaming is gaining much attention in the literature [6–10] because of its scalability, low cost, and tactical deployment. In P2P live streaming, all peers participate in distributing contents by sharing their resources (i.e., upload link bandwidth). However, providing a resilient P2P streaming service is a key challenge since peers are prone to departures and failures. Especially, a peer acts as both a server and a client, and thus each failure of a peer causes streaming disruption for peers downstream until the time the relevant topology is reconstructed.

In this paper, we focus on two challenging issues for resilient P2P live streaming: (1) efficiency and (2)

☆ A preliminary version of this paper was presented at The International Workshop on Peer-to-Peer Systems (IPTPS' 08) and published as an electronic archive [1].

* Corresponding author. Address: Seoul National University, Building 301, Room 553-1,599 Gwanak-ro, Gwanak-gu, Seoul 151-742, Republic of Korea. Tel.: +82 2 880 1848.
E-mail addresses: kwpark@mmlab.snu.ac.kr (K. Park), shpack@korea.ac.kr (S. Pack), tkkwon@snu.ac.kr (T. "Taekyoung" Kwon).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.10.022

incentive. The first issue is how to efficiently support dynamic membership; i.e., each peer dynamically joins or leaves (peers easily fail or leave at will) the P2P network. Hence resilience to churning rate should be taken into account. In general, sending more signaling packets (e.g., keep-alive or heartbeat messages) between peers will increase the resilience level of the network, while consuming a considerable amount of network resources. There is a trade-off between the service resilience and the signaling overhead of the system; so the target level of resilience should be carefully decided. The control overhead of a system is also tightly coupled with the complexity of the system. The system should be designed to be as simple as possible for efficient operations.

The second issue, incentive, is more crucial. The performance of P2P systems strongly relies on how much resources are contributed by peers. Therefore, the way to encourage peers to contribute their resources for stream distribution should be devised in order to improve the quality of live streaming services in terms of delay and resilience. In P2P file-sharing applications (e.g. [11,12]) with an incentive mechanism, download rate of a peer is roughly proportional to the peer's contributions. Hence, peers with more contributions need a shorter time to receive a file than others. Even though a peer with low contribution (i.e., a peer with small upload capacity) is able to receive a complete file, it will take longer time than peers with high contributions. This incentive mechanism makes sense since the objective is just to share the file. However, there is a significant difference in live streaming applications and file-sharing applications. Especially in the P2P live streaming applications, the incentive mechanism should be designed by considering unique characteristics of P2P live streaming which contrast with those file-sharing applications.

There have been some P2P live streaming proposals that have similar incentive mechanisms to the file-sharing applications. Generally, peers contribute to a system by providing their upload bandwidth in order to gain reputation (e.g. [13,14]) or currency (e.g. [15]) as a reward. In these systems, reputation is used to compete for connections with better neighbors against other peers, or currency is consumed to buy chunks from other peers. However, streaming packets of a stream are generated by a source in live streaming applications, encoded and sent to other peers (suppose the stream is k kbps). Live streaming packets are periodically generated and streaming packets at the current moment are disseminated. Thus, if a peer with high contributions is receiving a stream of k kbps, there is no way to give more incentive to this contributor. Peers with low contributions, receiving lower than k kbps, cannot play a video due to the nature of live streaming. Therefore, we should devise a different incentive mechanism for live streaming applications.

We propose a novel P2P live streaming system, *Climber*. The salient characteristics of Climber are summarized as follows. (1) Climber is a hybrid of a tree and a mesh. That is, a single tree is a main structure for streaming; however, random (or redundant) edges between peers (not between parent and child) are added to the tree for resilient connectivity. (2) Climber is simple in the sense that the effect of the dynamic membership on the system topology is mitigated. Climber simply attaches a new joining node[1] to the tree as a leaf node. When a non-leaf node leaves or fails, Climber minimizes the effect of peer departure by efficiently relocating its descendants to other positions of the tree. (3) Despite the time-varying network conditions (e.g., churning rate, congestion, delay), Climber bounds the level of resilience of a system. Climber adaptively changes its mesh topology (i.e., the random edges) to maintain a target level of resilience which is an input QoS parameter. (4) Climber keeps improving the transfer delay of a tree using the mesh topology. When a peer finds an alternative path that is faster than the current data path from the root, the peer switches its link to the alternative path. This switching process continuously improves the overall root-to-peer (or end-to-end) delay in the system. (5) Climber embodies an incentive mechanism that enhances resilience despite churning rate. The number of potential incoming data paths from a source to a peer is approximately proportional to the number of child nodes of the peer. Hence, a peer will experience more resilient services as the peer maintains more child nodes (or allocate more upload link bandwidth). This incentive mechanism encourages nodes to assign more upload link bandwidth for streaming data distribution, which leads to more (system-wide) resilient services.

The contribution of this paper is three-fold. First, we proposes a simple and adaptive system for resilient P2P streaming. Second, an analytical model, which demonstrates the effect of the disruption rate, is developed and validated by simulations. Third, extensive simulation results are given to evaluate the performance of Climber against other schemes. To the best of our knowledge, Climber is the first proposal that provides the required level of resilience probabilistically, and the incentives to contribute more resources for better resilience that suits P2P live streaming services.

The rest of this paper is organized as follows. Design concepts and the operations of Climber are introduced in Section 2, and we analyze the level of disruption of Climber in Section 3. Section 4 gives simulation results, followed by related work in Section 5. Finally, we conclude this paper in Section 6.

2. Climber

2.1. System model

The structure of Climber is based on a single tree, rooted at the source, but more than one data path from the source may exist for each peer. We focus on P2P live streaming applications where 10 or 20 s of delay is tolerable, which is normally assumed in the literature [9,16]. We focus on the packet forwarding aspect of Climber, and how to achieve reliability is out of the scope of this paper. Reliable transmission can be achieved by error control techniques (e.g., forward error correction (FEC)) or retransmission techniques.

A peer i has O_i ($O_i \geqslant 0$) outdegrees, which means the peer i currently forwards packets to O_i child nodes. We as-

[1] Hereafter, we use the term "node" when a peer joins a system. However, the terms "node" and "peer" may be used interchangeably.

sume 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 (or upload) link bandwidth in P2P streaming. O_i^{max} ($O_i^{max} \geqslant 1$) is the maximum number of outgoing edges permitted to use by the peer. Peer i can adjust the maximum amount of possible contributions 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 parent node i. In Climber, parent and child nodes exchange heartbeat messages at relatively long intervals to detect a path failure. A heartbeat message contains the number of descendants and therefore a root node can estimate the current total number of nodes in the system by these messages.

2.2. Topology construction

A peer first sends a Join message to the root to join Climber. If the root cannot accept the new peer as its child, the Join message is randomly forwarded to one of its child nodes. If node i in the system (including the root) receives a Join message and the node has a remaining outdegree (i.e., $O_i < O_i^{max}$), it takes the new peer as a child node with the probability $1 - (O_i/O_i^{max})^l$, where l is a positive real number determined by the system. If the new node is not accepted, the Join message is randomly forwarded to one of its child nodes. When a leaf node receives the Join message, it should always take the node as a child node since $O_i = 0$ and $O_i^{max} \geqslant 1$. Note that the tree-depth parameter l is a configurable parameter that tunes the average tree-depth of Climber. If $l = 1$, the probability of accepting a new node as a child node linearly decreases as O_i increases. If $l > 1$, Climber tends to build a short and wide tree, while if $l < 1$, Climber tends to make a tall and narrow tree. Overall, the value of l will trade-off the end-to-end delay for system resilience, which will be investigated in Section 4.2.

In addition to the tree topology, we have to build a mesh topology for resilience. After node i joins the system, it makes a decision whether to establish a random edge or not with probability λ_t at time t for each available outdegree ($O_i^{max} - O_i$). λ_t is a random edge generation probability, which is sent from the root node piggybacked in the streaming packets. The terminal node ("prospective child nodes") of a random edge is randomly selected from the set of all the peers. Afterwards, node i forwards only the sequence numbers of the streaming packets to its prospective child nodes. A root node keeps a list of recently joined nodes and each node obtains the list from the root node when it joins. In this way, the joining node is aware of the peers. 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 2.4.

2.3. Handling churn

When a node i has a tree edge from a parent node j and a random incoming edge from a 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 that from the prospective parent by a certain margin (i.e., $Seq_j + \Delta < Seq_r$, where Δ is a predefined threshold to avoid oscillations), it indicates that the current data path is slower than the path via the random edge. Then node i changes its parent from node j to node r, and hence the 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 network topology changes, each peer reestablishes the random edge(s) periodically to other nodes.

Fig. 1 illustrates how a node "climbs" a tree using 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 arrow) 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 data path (1–3–6–7) is slower than the path (1–2–7), node 7 changes its parent from node 6 to node 2. Supposing $O_7 < O_7^{max}$, 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 to 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 [9]) or a failure recovery (proactive [17] or reactive [7]) method additionally. Instead, parent and child nodes need to exchange heartbeat messages at relatively long intervals, which means low signaling overhead. Actually, Climber does not distinguish node departure or failure from node congestion. That is, Fig. 1 describes the recovery procedure of Climber not only from congestion, but also from node failure: (a) when node 3 fails, its descendent nodes 6, 7, 8 (node 3's subtree) notice the current data path has a problem since no streaming packets are received; (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 old parent node 6 (through a newly established random edge r_7) and child nodes; (c) finally, the failed node 3 is placed at the end of data path (1–3) and node 1 concludes that node 3 has left the tree due to missing heartbeat messages. This continual *climbing* effort also keeps improving the performance of the distribution structure even if the system experiences the high churning rate.

If a node has no incoming random edges and its parent fails, then the node detects that there is no parent alive by missing heartbeat messages. We say the node is *disrupted*, and the node sends a Rejoin message to the root node, which is the same as the joining process. Note that disrupted nodes are the direct child nodes of a failed node. Therefore, the number of Rejoin messages is bounded to the number of current outdegrees of the failed node.

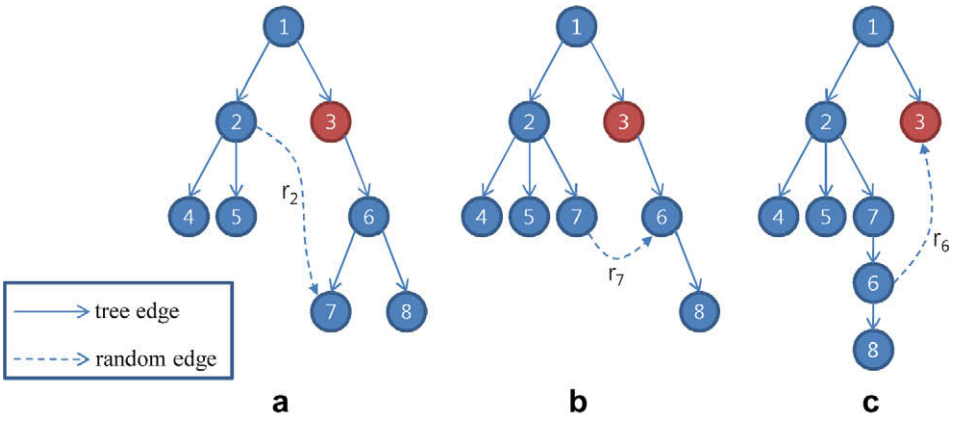

Fig. 1. Tree reconstruction steps are illustrated using random edges. When node 3 suffers from congestion, node 7 *climbs* the tree by being a child of node 2. Then node 6 becomes a child of node 7.

2.4. Adaptation

Climber provides fine-grained control over the mesh topology, i.e., λ_t indicates the level of structural resilience at time t in 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 disruption rate is defined as the ratio of the number of disrupted nodes to the total number of nodes in the system. In Climber, a source node compares a predefined disruption rate Q and Q_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. As the churning rate increases, Q_t will exceed Q. Then, the source node increases λ_{t+1} to enhance the resilience of the distribution structure. Otherwise, the source node decreases λ_{t+1} to save network bandwidth by reducing the number of random edges. In Climber, a source node smoothes Q_t by a moving average method and λ_{t+1} is derived from

$$\lambda_{t+1} = \begin{cases} max(\lambda_t - 0.01, 0.01) & \text{if } Q_t \leqslant Q, \\ min(\lambda_t + 0.01, 1) & \text{otherwise.} \end{cases}$$

The above formula is one example of how to adjust λ_{t+1}. Other solutions (e.g., additive increase and multiple decrease (AIMD) as in a TCP window) may be applied depending on the characteristics of the service or streaming contents.

2.5. Incentives

Climber gives more incentives to a highly contributing peer in the sense that a peer that makes higher contributions will have more descendants, which implies more incoming random edges to its subtree probabilistically. The level of contribution of a peer is defined as the current number of outgoing links (child nodes). Note that the 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 outdegrees and the participating time. This is because more attempts have been by the node and its descendants to set up outgoing random edges to prospective child nodes. Since Climber is designed to recover from failures by using random edges, a node with more descendants will experience less streaming disruptions. 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 in P2P streaming services.

Instead of providing more resilient service as an incentive, there is another way to give incentives namely by adjusting streaming quality (i.e., higher bit rate). Multi-bit rate encoding schemes (e.g., MDC [18], scalable video coding [19], or stream switching [20]) can provide differentiated high or low quality streams to each nodes. Climber can be extended using multiple Climber trees with a multi-bit rate encoding scheme to give a better quality for a user according to the user's contribution, which is our future work.

3. Disruption rate analysis

Climber allows resilient streaming services by adding random edges. The system-wide number of random edges is adjusted based on λ_t sent by a root node. There is a trade-off between the resilience of a system and the signaling overhead over random edges. In this section, we analyze Climber with a focus on the effect of random edges on the disruption rate.

We have the following notation and assumptions. m nodes form a complete g-ary tree with height h. Let $\mathbf{G}$ denote the set of all nodes of the tree. All non-leaf nodes are assumed to have the same number, g, of child nodes. For the sake of simplicity, m is set to the number of nodes of complete g-ary trees. In addition to g child nodes, every node is assumed to have r random edges whose destinations are uniformly distributed over other nodes. Note that r is a positive real number. v_d refers to a node v at depth d (e.g., $0 \leqslant d \leqslant h, d = 0$ for a root node). The set of descendant nodes of v_d is denoted by $\mathbf{A_d}$. When v_d leaves (we assume the root node does not leave), v_d's descendants are partitioned into g subtrees and each subtree is denoted

by $\mathbf{a_d}$. The number of nodes of a tree $\mathbf{T}$ is denoted by $|\mathbf{T}|$ (e.g., $|\mathbf{G}| = m$). Hence, $|\mathbf{A_d}|$ and $|\mathbf{a_d}|$ are respectively calculated as

$$|\mathbf{A_d}| = \sum_{i=0}^{h-d} g^i - 1 \tag{1}$$

and

$$|\mathbf{a_d}| = \frac{|\mathbf{A_d}|}{g}. \tag{2}$$

Let $f(p,q)$ be the number of possible permutations that q are chosen out of p elements with repetitions on the condition that each of p different elements exists in the permutations. For example, if $p = 2$ and the elements are 1 and 2, $f(2,3) = 6$, since among eight possible permutations (111, 112,121,122,211,212,221,222), only six permutations satisfy the constraint (112,121,122,211,212,221). Then $f(p,q)$ can be calculated as

$$f(p,q) = \begin{cases} 0 & \text{if } p > q, \\ 1 & \text{else if } p = 1, \\ p^q - \sum_{i=1}^{p-1} \binom{n}{i} \cdot f(i,q) & \text{otherwise.} \end{cases}$$

Since Climber recovers disrupted nodes by random edges, it is necessary to estimate how many random edges are set up from non-disrupted nodes. For the sake of analysis, if v_d fails, nodes in $\mathbf{A_d}$ are assumed to be partitioned from $\mathbf{G}$, and then the expected number of disrupted nodes in $\mathbf{A_d}$ will be determined by the expected number of random edges from $\mathbf{G} - (\mathbf{A_d} \cup \{v_d\})$. For a node v_d, $\beta(k,d)$ is the probability that k random edges fall into $\mathbf{A_d}$ from $\mathbf{G} - (\mathbf{A_d} \cup \{v_d\})$. Since $|\mathbf{G} - (\mathbf{A_d} \cup \{v_d\})| = m - |\mathbf{A_d}| - 1$ and every node has r random edges, $\beta(k,d)$ is given by

$$\beta(k,d) = \binom{(m-|\mathbf{A_d}|-1)r}{k} \cdot \left(\frac{|\mathbf{A_d}|}{m-1}\right)^k \cdot \left(1 - \frac{|\mathbf{A_d}|}{m-1}\right)^{(m-|\mathbf{A_d}|-1)r-k}. \tag{3}$$

We next derive $n(k,d)$, the expected number of disrupted nodes if v_d fails when $\mathbf{A_d}$ has k incoming random edges from outside. When v_d fails, $\mathbf{A_d}$ is partitioned into g subtrees (each subtree is $\mathbf{a_d}$). Every node in $\mathbf{a_d}$ is not disrupted if one or more random edges fall into the subtree $\mathbf{a_d}$. Out of g subtrees, suppose i subtrees do not have any incoming random edges. As a result, $i \cdot |\mathbf{a_d}|$ nodes will be disrupted. Then, $n(k,d)$ is given by

$$n(k,d) = \sum_{i=0}^{g} \frac{\binom{g}{i} \cdot f(g-i,k)}{g^k} \cdot i \cdot |\mathbf{a_d}|. \tag{4}$$

By multiplying Eqs. (3) and (4), the expected number of disrupted nodes when v_d fails, $n(d)$, can be obtained from

$$n(d) = \sum_{k=0}^{m-1} \beta(k,d) \cdot n(k,d). \tag{5}$$

The probability of a node to be located at depth d is $\frac{g^d}{m}$. Then, the expected number of disrupted nodes when a node v fails, n, is given by

$$n = \sum_{d=1}^{h} \frac{g^d}{m} \cdot n(d). \tag{6}$$

Fig. 2 shows the expected number of disrupted nodes (n) as a function of the number of nodes (m). In Fig. 2, a 4-ary complete tree and three different r values (0.01, 0.05, and 0.1) are used, and the expected number of random edges in each experiment is mr. It can be found that the analytical and simulation results are consistent, which validates the analytical model. From Fig. 2, when r is 0.01, the number of disrupted nodes converges to 2.9, whereas the number converges to 1.4 when r is 0.1. Consequently, it can be concluded that even an arbitrary small r (e.g., 0.01) value can bound the level of disruption of Climber even if the number of nodes increases exponentially.

4. Simulation results

4.1. Methodology

We use the GT-ITM topology generator [21] to generate the network topology, which consists of 3000 peers and 600 routers, using the Transit-Stub graph model. The topology consists of three transit domains with 8 transit routers each. There are an average of three stub domains per transit router, and an average of eight stub routers per stub domain (hence, there are total $3 \times 8 \times (1 + 3 \times 8) = 600$ routers). Peers are randomly connected to one of 576 stub routers. Link delays for the simulation are derived from [22]. Link delays are uniformly distributed ranging from 1 to 55 ms for transit-transit or transit-stub links, and link delays within a stub are uniformly distributed from 1 to 10 ms. Each link is a symmetric link without packet loss as assumed in [23,24]. Only peers join and leave the system and every peer departure is regarded as a node failure. When a peer leaves the system, there is no explicit notification and the peer rejoins the system (at a different position in the tree) immediately again so that the number of peers in the system remains unchanged. The maximum outdegree O_i^{max} of peer i is randomly chosen between 1 and 10 ($1 \leqslant O_i^{max} \leqslant 10$).

For performance comparison, we consider the following relevant systems.

- *SplitStream* [7] is a multitree-based system which partitions an original stream into multiple substreams. Each substream is distributed over its corresponding tree. Note that the number of trees is equal to the number of substreams. Each tree is structured according to Scribe [25] built on Pastry [26]. We use 16 multicast trees, which is a default setting in [27]. Each tree is interior-node-disjoint, where each node is an interior node in at most one tree, and a leaf node in the other trees. Since a node failure causes the loss of a single substream, SplitStream can mitigate the effect of node failures.
- *NICE-PRM* is Probabilistic Resilient Multicast (PRM) [28] added on the NICE application-layer multicast protocol [29]. NICE creates a hierarchical tree, where an interior node of the tree is a cluster of multiple peers in proximity. Its cluster size is maintained between k and $3k - 1$, where $k = 3$ in our simulations as in [29]. Streaming packets are delivered through cluster leaders of each

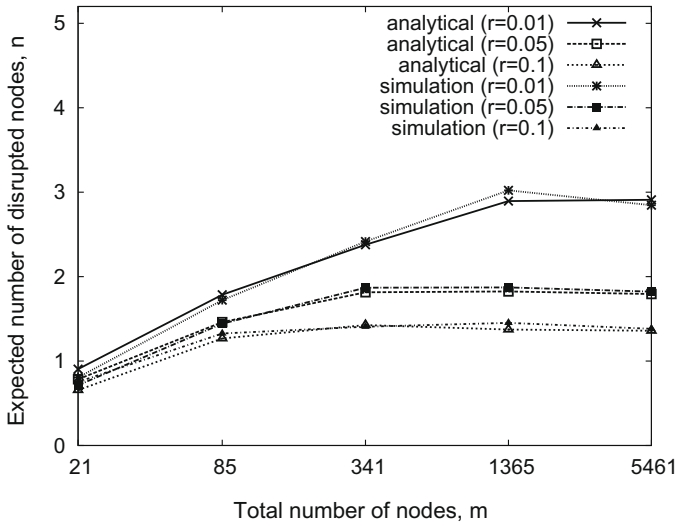

Fig. 2. Expected number of disrupted nodes vs. total number of nodes. We plot both analytical and simulation results as r changes.

cluster. PRM uses randomized forwarding among peers[2] and we use PRM-(3,0.01), where three random nodes are chosen and packets are forwarded to each of them with probability 0.01, as configured in [28]. We focus on the resilience and efficiency of NICE-PRM, compared to those of Climber.

- *BT* is a Basic-Tree that is a primitive graph-theoretical tree. BT consists of m nodes and $m-1$ edges, without any additional functionalities for performance improvement. Node i always receives a new joining node until O_i reaches O_i^{max}, and randomly forwards a joining request to one of its child nodes if O_i equals O_i^{max}. A node failure is detected by a missing Heartbeat message and disrupted nodes simply rejoin the tree. Although the performance of BT is poor, the complexity of BT is minimum.
- *Climber-Q* is our proposed scheme, where Q is the target delivery rate of QoS. The delivery rate is the proportion of nodes which receive the streaming packet among the alive nodes. Note that the disruption rate is $100-Q$ (%). For instance, Climber-95 refers to a Climber system that adjusts the number of random edges to achieve 95% delivery rate, depending on the current network churning rate.

In Climber and NICE-PRM, peers reestablish their random edges in every 10 s.

4.2. Tree depth parameter l

As mentioned in Section 2.2, node i takes a new joining peer as a child node with the probability $1-(O_i/O_i^{max})^l$. As l becomes greater, Climber tends to build a short and wide tree, while if l gets smaller, Climber tends to make a tall and narrow tree. Fig. 3 shows the average delay from a root node to other nodes, and the average number of disruptions a node experiences during simulation time. The experiment is carried out for 1000 s in Climber-95. Network churning rate is fixed at 5% (i.e., out of 3000 peers, 150 peers fail for each second) to avoid side effects from time-varying number of random edges. As l increases, the average delay from a root node tends to decrease because Climber builds a shorter tree. There is a trade-off between the root-peer delay and system resilience. In a short tree, non-leaf nodes fill their remaining outdegrees with new joining nodes as child nodes. This leads to the insufficient number of random edges, since random edges are probabilistically generated for each remaining outdegrees of a node. As shown in Fig. 3, the average number of disruptions per node increases with the increase of l. Since $l=2$ is the knee point, l is set to two for the rest of our simulations.

4.3. Adaptation

To support a wide range of live streaming services, Climber provides a way for the service provider to adjust the level of QoS (e.g., the delivery rate, Q) of the stream. Climber periodically checks Q_t, the currently measured delivery rate at time t, and adaptively changes the random edge probability λ_t as described in Section 2.4. Fig. 4 shows the adaptation behavior of Climber. The simulation is run over 10,000 s and the network churning rate varies over time. Churning rate is defined as the rate of the number of leaving nodes in a second to the total number of nodes in the system. We intentionally vary the churning rate between 0% and 20% (600 nodes fail in a second) as the simulation time goes by. In this scenario, we excercised two versions of Climber, Climber-95 and Climber-80, which indicates the delivery rate of each version will target 95% and 80%, respectively.

As shown in Fig. 4, Climber successfully adapts to the network churning rate. Since SplitStream and NICE-PRM

[2] This is similar to random edges in Climber. However, Climber forwards only the sequence number over random edges, while PRM forwards streaming packets. Also, PRM maintains a fixed number of random edges probabilistically system-wide.

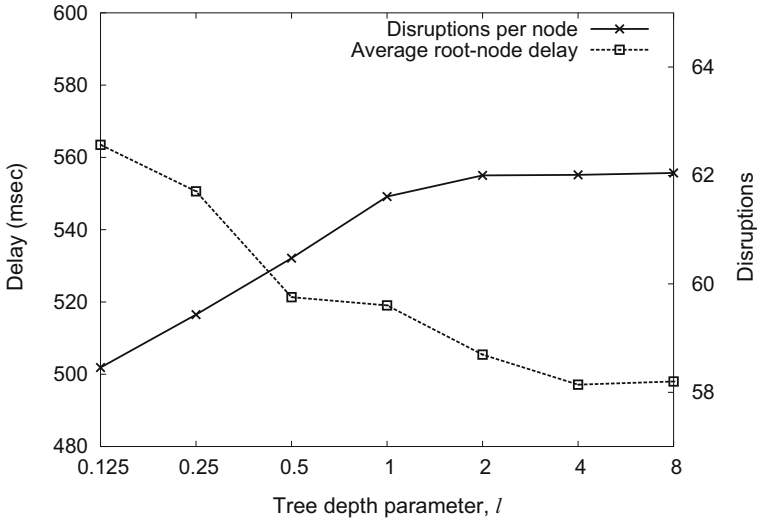

Fig. 3. Delay vs. tree-depth parameter l in Climber-95. As l increases, the average number of child nodes increases, which reduces end-to-end delay. Note that the actual probability of having more child nodes converges when l exceeds 2.

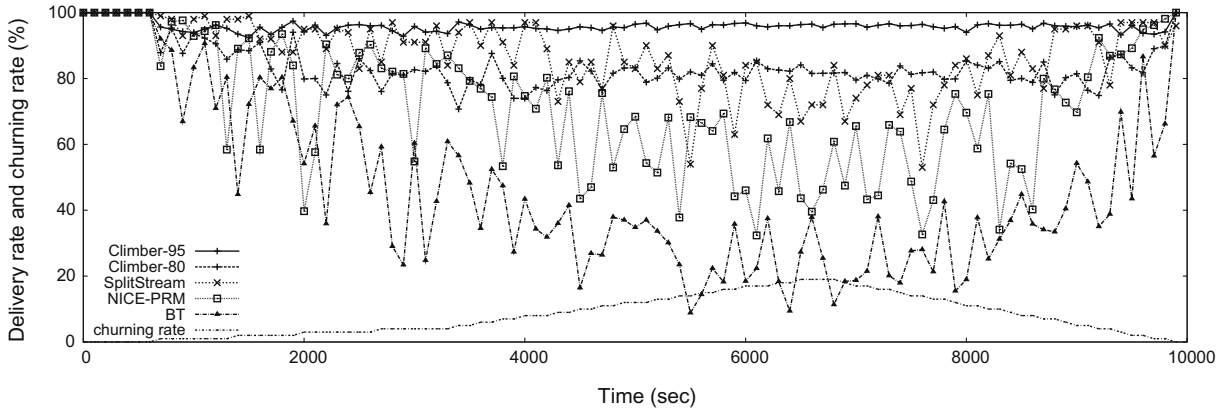

Fig. 4. Climber adaptively changes its topology depending on the current network conditions to maintain a target delivery rate. Notice that other schemes exhibit high fluctuation of delivery rates.

do not have any adaptation mechanisms, the performance degrades as the churning rate increases. NICE-PRM generates a constant number of random edges, 90 (we use PRM-(3,0.01)), despite the time-varying network condition. SplitStream performs better than NICE-PRM on average, due to its interior-node-disjoint structure. In the worst case of our simulation around 6500 s, when 20% of nodes fail, the delivery rates of SplitStream, NICE-PRM and BT exhibit spikes around 75%, 60%, and 30%, respectively, while Climber maintains a target level of delivery rate Q. To meet the constraint Q, Climber increases or decreases λ_t to adjust the number of random edges.

4.4. Incentive

Climber provides more resilient streaming services to the nodes of more contributions, i.e., a node that maintains more descendant nodes. Therefore, peer i may increase its O_i^{max} with the expectation that if it contributes more to the system, then it will receive more resilient service. Fig. 5 shows the number of disturbances that node i experiences with different O_i^{max} values. The number of disturbances of a node is incremented when the node is disconnected due to its parent's failure and hence it rejoins the system. In Climber, a node with a larger O_i^{max} has more descendants probabilistically, since establishing more outgoing random edges implies that prospective descendants may become "real" descendants over time. As a result, these nodes will have less chance to experience disturbances. Since SplitStream, NICE-PRM and BT do not have an incentive mechanism, there is no relationship between a node's maximum number of outdegrees and the expected number of disturbances.

Note that our definition of contribution is the actual upload bandwidth a node has served to the system, which is measured in terms of the number of transmitted packets.

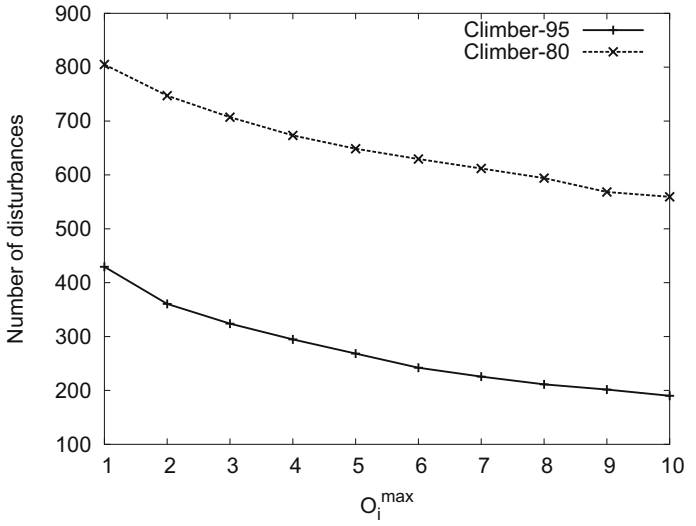

Fig. 5. Number of disturbances events vs. O_i^{max}. As O_i^{max} increases, a node has more random edges, which enhances the system resilience.

Fig. 6 demonstrates the effectiveness of our incentive mechanism. The number of disturbances of 3000 nodes are plotted for each scheme after 10,000 s, depending on a node's total upload bandwidth. In Climber-95, if a node contributes twice more than another node, the node roughly experiences half the number of disturbances than the other node. When we look at median values, a node with 10,000 packets uploaded experiences 300 disturbances, while a node with 20,000 packets uploaded experiences 150 disturbances. Similar trends can be also observed in NICE-PRM because PRM utilizes random edges as well. The reason why NICE-PRM has so many nodes with little contribution is that the NICE application forms a number of clusters, and a streaming data path is formed via cluster leaders only. In NICE, a node located at the center of other nearby nodes is likely to be a cluster leader while the others serve as leaf nodes with high probability, that is most of the nodes will make little contribution. Fig. 6 also shows that the forwarding load and the number of disturbances in SplitStream are spread across all participating nodes.

4.5. Simplicity

To evaluate the signaling cost of each scheme, we measured the control overhead that is defined as the necessary number of nodes to maintain a structure. For example, BT needs $O(\log N)$ nodes to keep the structure for a new node,

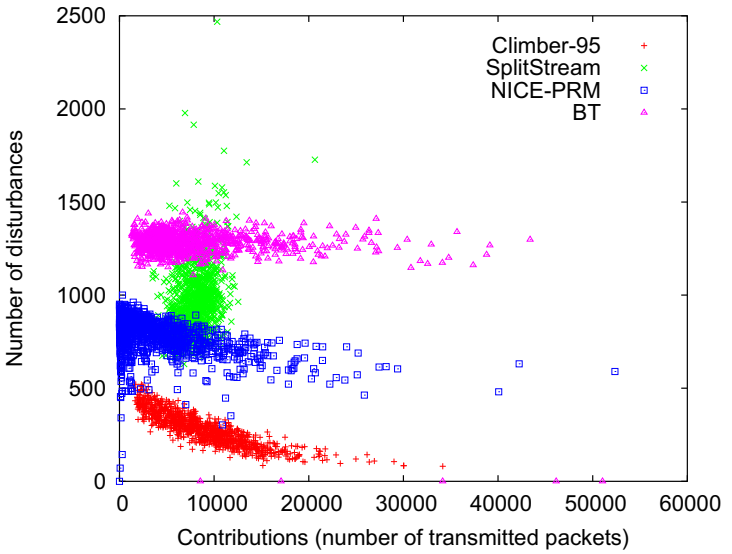

Fig. 6. Number of disturbance events vs. contributions. All peers in Climber and the cluster leaders in NICE-PRM show the similar trend that a peer with more contributions experiences less disturbances. However, non-cluster leaders in NICE-PRM cannot make more contribution by nature. SplitStream balances the load of contributions among peers.

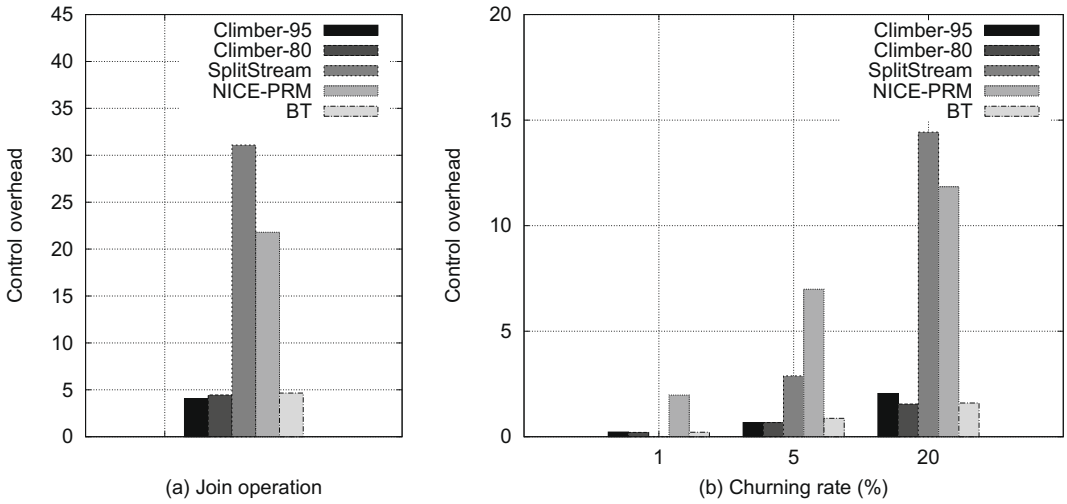

Fig. 7. Control overhead vs. membership change. The control overhead of Climber is much less than those of NICE-PRM and SplitStream.

because a join request is simply forwarded downstream until it reaches a node with a remaining outdegree for the new node. Fig. 7 illustrates the control overhead per node in each scheme. Since the main structure of Climber is a tree, its control overhead is similar to that of BT. NICE-PRM needs more nodes for maintaining the structure because of its cluster structure. Since SplitStream is a set of Scribe trees, built on Pastry, the maintenance cost is much higher than Climber.

Fig. 7a is the cumulated amount of control overhead when 3000 nodes join the overlay system one after another. SplitStream distributes 16 substreams over 16 Scribes, each of which is an interior-node-disjoint tree. Whenever a node joins or leaves the system, 16 trees should be updated. The NICE application forms a number of clusters and the size of each cluster is maintained between 3 and 8 nodes. A *split* operation is performed if more than 8 nodes join a cluster, which leads to high control overhead. When the network churning rate is 5%, Climber-95 needs less control overhead than Climber-80, and even less than BT. This is because when a node fails, nodes in Climber-95 are less likely to be disrupted by employing a larger number of random edges compared to Climber-80 and BT (no random edges). Therefore, climbing occurs more frequently in Climber-95, where climbing usually needs less control overhead than a rejoin operation. When the network churning rate is increased to 20%, Climber-80 and Climber-95 generate considerable numbers of random

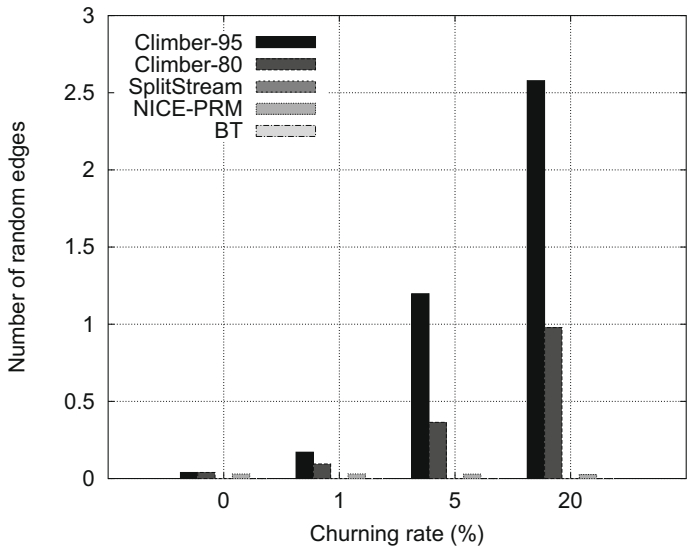

Fig. 8. Number of random edges vs. churning rate. As the churning rate increases, Climber increases the number of random edges per node to maintain the delivery rate.

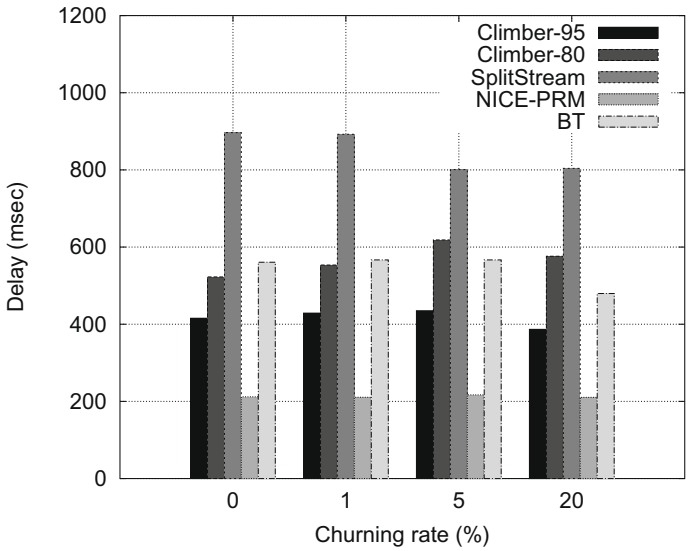

Fig. 9. Delay vs. churning rate. NICE-PRM achieves the lowest delay since it forms the tree considering proximity among peers. SplitStream shows the worst delay since all the 16 substreams should be delivered to restore the original stream. When the churning rate is changed from 5% to 20%, the delay in Climber is reduced due to self-improvement leveraging random edges.

edges and the control overhead of both schemes is greater than BT (each random edge generation is counted as 1 unit of control overhead).

4.6. Redundancy

Climber provides resilience at the cost of adding random edges. Fig. 8 exhibits the number of random edges required to satisfy the given level of QoS Q. When the network churning rate is 5%, Climber-95 needs 1.2 random edges per node while Climber-80 needs 0.4 random edges on average. Climber generates a small number of random edges (the minimum value of λ_t is set to 0.01) even if there are no node failures, to cope with a sudden node failure in the future. Climber-95 has many more random edges than Climber-80, to provide more resilient services. NICE-PRM has a fixed number of random edges (0.03 on average); SplitStream and BT do not have any random edges. In Climber, only the sequence numbers of streaming packets are forwarded through random edges and therefore less network bandwidth is consumed.

4.7. Delay

Fig. 9 shows the average delay from a root node to a non-root node. Throughout our simulation, NICE-PRM keeps the average delay around 200 ms at the cost of large maintenance overhead. Although SplitStream exploits the proximity of nodes in Pastry, the actual delay of a packet is quite long. This is because a stream is partitioned into 16 substreams, and each node must wait for the last substream to restore the original packet. Climber self-improves the structure (or shortens its delay) by using random edges. Since Climber-95 has more random edges than Climber-80, the average delay of Climber-95 is smaller than Climber-80. It is interesting to observe that the average delay of Climber-95 is reduced when the network churning rate is changed from 5% to 20%, and a similar trend is observed in Climber-80. This can be explained from Fig. 8, where the number of random edges increases more than twice as the churning rate increases from 5% to 20%. Meanwhile, when the network churning rate is changed from 1% to 5%, the number of random edges is not enough to allow self-improvement and thus the average delay is not reduced.

5. Related work

P2P live streaming solutions can be broadly classified into two categories depending on the overlay structures: tree-based and mesh-based approaches. The tree-based approach implements a single or multiple distribution trees, where a root node is 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.

SplitStream [7] splits the original stream into k stripes and forwards each stripe via one of separate k trees built using Scribe [25] on the top of Pastry [26]. Scribe trees form a forest of interior-node-disjoint multicast trees that allow the forwarding load distribution among all participating peers. If SplitStream can leverage appropriate stream encoding techniques (such as Multiple Description Coding (MDC) [18]), SplitStream can become robust to the failures because MDC allows a peer to restore the original stream despite packet losses (clearly, the quality will be degraded as packet losses increase). Although SplitStream exploits nodes' proximity features of Pastry to reduce root-to-peer delay, a node should wait for all k stripes for the best quality, which increases the overall delay.

Maintaining k independent multicast trees also incurs significant signaling overhead. If a catastrophic failure happens (i.e., numerous nodes fail almost simultaneously), SplitStream will suffer from handling a lot of signaling messages to be stabilized.

NICE [29] is an application-layer multicast protocol which creates a hierarchical and cluster-based control topology. NICE creates a hierarchical topology with multiple levels, and nodes at each level are partitioned into a set of clusters. Nodes in each cluster elects a cluster leader that has the smallest maximum distance (or RTT) to all the other nodes in the cluster. The cluster leader performs a *merge* or *split* operation to maintain its cluster size between k and $3k-1$. Streaming packets are delivered through hierarchically-connected cluster leaders and therefore it can significantly reduce delay from a root node to peers along the overlay. However, due to the overhead for keeping clusters and their leaders, more frequent merge or split operations will be triggered as the churning rate increases. That is, the topology maintenance cost is relatively large especially for high churn. The authors of NICE also proposed PRM [28] to effectively recover node failures and to improve data delivery rate. PRM uses randomized forwarding among peers to effectively recover node failures and to improve data delivery rate. Each node in PRM chooses a constant number of nodes randomly (which are similar to prospective child nodes in Climber) and forwards data to each of them with a low probability. However, since PRM maintains a fixed number of random edges, disruption events occur more often as the churning rate increases. In contrast, Climber adaptively changes the number of random edges in order to satisfy a target level of resilience.

On the other hand, in the mesh-based approach, a peer has flat 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 issues since the streaming packets are generated by a live source at short intervals. Peers in Coolstreaming [8] and Chainsaw [10] 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. In [30], push algorithms are used to quickly disseminate chunks to a well-defined set of peers (a fraction of all peers). The remaining peers which have not received the chunks use the pull mechanism to distribute the chunks amongst themselves. Although the distribution structure is built on top of a pastry-like overlay structure, which ensures a logarithmic diameter and robustness, high maintenance cost is inevitable compared to Climber.

There have been a number of proposals that consider incentives for P2P live streaming. [15] uses an internal currency called points to quantify a peer's contribution. Peers compete for good parents in a first-price auction-like procedure using their points. In [13], a peer's contribution determines its score and its relative ranking in the system, which in turn determines its priority in selecting good peers for better streaming quality. [31] employs time-constrained tit-for-tat exchanges which seek to allocate an equal share of a node's upload bandwidth to each of its neighbors. Only nodes with sufficiently large contributions to the system are able to fully receive live streams. In contrast, uncooperative (or selfish) nodes cannot properly receive the streams as they are unable to fill their buffers in time. Consequently, incentivizing a fair sharing of resources can be achieved. In [14], every node maintains a record of the previous interactions with other peers. Peers choose their neighbors based on their history records, which consequently optimizes the system performance. Also, a taxation model such as [23] has been proposed, where resource-rich peers contribute more bandwidth to the system, and subsidize for the resource-poor peers. To sum up, incentives are given to users as the priority to receive more or faster data from other peers. Since the upload bandwidth of a peer is a scarce resource and usually a bottleneck in a system, it is reasonable for peers with more contribution to get more rewards. However, the above incentive mechanisms focus on how to retrieve more data steadily from good neighbors – as in file-sharing applications, and therefore these are less effective in live streaming.

6. Conclusion

This paper introduces Climber, an adaptive peer-to-peer live streaming system with incentives for resilience. We focus on efficiency and incentive in P2P live streaming. The main structure of Climber is a tree for simple and efficient stream distribution. To address the problem of dynamic membership, we add random edges to the tree; the number of random edges are dynamically adjusted to maintain service quality. Climber also embodies an incentive mechanism, which allows a peer with more contributions to experience less disruptions. Analytical modeling verifies the effectiveness of random edges; the number of disrupted nodes is bounded even if the number of total nodes in the system increases exponentially. It is shown that Climber successfully keeps the target resilience (or delivery rate) despite a time-varying churning rate, while SplitStream and NICE-PRM exhibit high fluctuation of the delivery rates and lack adaptiveness to the changing churning rate. The incentive mechanism of Climber achieves another desirable statistical property: the amount of contributions of a node is inversely proportional to the number of disturbances that the node experiences. Furthermore, Climber incurs significantly low control overhead to be stabilized in case of high churn, compared to SplitStream and NICE-PRM.

Acknowledgements

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2007-331-D00267)

and the IT R&D program of MKE/IITA (2008-F-034-02). The ICT at Seoul National University provides research facilities for this study.

References

[1] K. Park, S. Pack, T. Kwon, Climber: an incentive-based resilient peer-to-peer system for live streaming services, in: IPTPS 2008, 2008.
[2] Akamai, 2009. <http://www.akamai.com>.
[3] AOL, 2007. <http://press.aol.com/index.cfm>.
[4] MSN, 2007. <http://liveearth.msn.com>.
[5] P. Gilmore, How Akamai works, in: 10th UK Network Operators' Forum, 2008.
[6] Y. Chu, S.G. Rao, S. Seshan, H. Zhang, A case for end SystemMulticast, in: Measurement and Modeling of Computer Systems, 2000, pp. 1–12.
[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Splitstream: high-bandwidth multicast in cooperative environments, in: ACM SOSP 2003, 2003.
[8] X. Zhang, J. Liu, B. Li, T. Yum, DONet/CoolStreaming: a data-driven overlay network for live media streaming, in: IEEE Infocom 2005, 2005.
[9] V. Venkataraman, P. Francis, J. Calandrinoz, Chunkyspread: Multi-tree unstructured peer-to-peer multicast, in: IPTPS 2006, 2006.
[10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. Mohr, Chainsaw: eliminating trees from overlay multicast, in: IPTPS 2005, 2005.
[11] B. Cohen, Incentives build robustness in bittorrent, in: 1st Workshop on Economics of Peer-to-Peer Systems, 2003.
[12] K.G. Anagnostakis, M.B. Greenwald, Exchange-based incentive mechanisms for peer-to-peer file sharing, in: ICDCS 2004, 2004.
[13] A. Habib, J. Chuang, Service differentiated peer selection: an incentive mechanism for peer-to-peer media streaming, in: IEEE Transactions on Multimedia, vol. 8, 2006, pp. 610–621.
[14] F. Pianese, D. Perino, J. Keller, E. Biersack, Pulse: an adaptative, incentive-based, unstructured P2P live streaming system, in: IEEE Transactions on Multimedia, vol. 9, 2007, pp. 1645–1660.
[15] G. Tan, S. Jarvis, A payment-based incentive and service differentiation scheme for peer-to-peer streaming broadcast, in: IEEE Transactions on Parallel and Distributed Systems, vol. 19, 2008, pp. 940–953.
[16] W. Montgomery, Techniques for packet voice synchronization, IEEE Journal on Selected Areas on Communications 1 (1983) 1022–1028.
[17] J. Byers, M. Luby, M. Mitzenmacher, A digital fountain approach to asynchronous reliable multicast, IEEE Journal on Selected Areas in Communications 20 (2002) 1528–1540.
[18] V.K. Goyal, Multiple description coding: compression meets the network, IEEE Signal Processing Magazine 18 (2001) 74–94.
[19] H. Schwarz, D. Marpe, T. Wiegand, Overview of the scalable video coding extension of the h.264/avc standard, in: IEEE Transactions on Circuits and Systems for Video Technology, vol. 17, 2007, pp. 1103–1120.
[20] M. Karczewicz, R. Kurceren, The sp- and si-frames design for h.264/avc, in: IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, 2003, pp. 637–644.
[21] E. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, in: IEEE Infocom 1996, 1996.
[22] S. Fahmy, M. Kwon, Characterizing overlay multicast networks and their costs, in: IEEE/ACM Transactions on Networking, vol. 15, 2007, pp. 373–386.
[23] Y. Chu, J. Chuang, H. Zhang, A case for taxation in peer-to-peer streaming broadcast, in: ACM PINS 2004, 2004.
[24] I. Keidar, R. Melamed, A. Orda, EquiCast: Scalable multicast with selfish users, in: ACM PODC 2006, 2006.
[25] A. Rowstron, A. Kermarrec, M. Castro, P. Druschel, SCRIBE: the design of a large-scale event notification infrastructure, in: NGC 2001, 2001.
[26] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: IFIP/ACM Middleware 2001, 2001.
[27] FreePastry, 2009. <http://freepastry.org/FreePastry/>.
[28] S. Banerjee, S. Lee, B. Bhattacharjee, A. Srinivasan, Resilient multicast using overlays, in: IEEE/ACM Transactions on Networking, vol. 14, 2006, pp. 237–248.
[29] S.Banerjee, B. Bhattacharjee, C. Kommareddy, Scalable application layer multicast, in: ACM SIGCOMM 2002, 2002.
[30] T. Locher, R. Meier, S. Schmid, R. Wattenhofer, Push-to-pull peer-to-peer live streaming, in: 21st International Symposium on Distributed Computing (DISC), 2007.
[31] T. Locher, R. Meier, R. Wattenhofer, S. Schmid, Robust live media streaming in swarms, in: 19th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), 2009.

Kunwoo Park received his B.S. in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2004. Currently, he is working towards a Ph.D. degree at the School of Computer Science and Engineering, Seoul National University. His research interests include peer-to-peer networks, overlay streaming, IPv6, and mobility management.

Sangheon Pack received the B.S. (magna cum laude) and Ph.D. degrees from Seoul National University, Seoul, Korea, in 2000 and 2005, respectively, both in computer engineering. Since March 2007, he has been an Assistant Professor with the School of Electrical Engineering, Korea University, Seoul. From 2005 to 2006, he was a Postdoctoral Fellow with the Broadband Communications Research Group, University of Waterloo, Waterloo, ON, Canada. He was the recipient of IEEE ComSoc APB Outstanding Young Research Award in 2009 and a Student Travel Grant Award at the 2003 IFIP Personal Wireless Conference (PWC). From 2002 to 2005, he was a recipient of the Korea Foundation for Advanced Studies Computer Science and Information Technology Scholarship. In 2003, he was a Visiting Researcher at Fraunhofer Institute for Open Communication Systems (FOKUS), Berlin, Germany. His research interests include mobility management, wireless multimedia, vehicular networks, and Future Internet.

Ted "Taekyoung" Kwon is an associative professor in the school of computer science and engineering, Seoul National University (SNU) since 2004. Before joining SNU, he was a post-doctoral research associate at UCLA and at City University New York (CUNY). He obtained B.S., M.S., and Ph.D. degree from the department of computer engineering, SNU, in 1993, 1995, and 2000, respectively. During his graduate program, he was a visiting student at IBM T. J. Watson research center and University of North Texas. His research interest lies in sensor networks, wireless networks, IP mobility, and ubiquitous computing.