

On Providing Reliability Guarantees in Live Video Streaming with Collaborative Clients

Aravindan Raghuv[†] and Yingfei Dong* and David Du[†]

[†]Dept. of Computer Science and Engineering, Univ. of Minnesota, Minneapolis, MN 55455, USA

*Dept. of Electrical and Computer Engineering, Univ. of Hawaii, Honolulu, HI 96822, USA

ABSTRACT

Although many overlay and P2P approaches have been proposed to assist large-scale live video streaming, how to ensure service quality and reliability still remains a challenging issue. Peer dynamics, especially unscheduled node departures, affect the perceived video quality at a peer node in two ways. In particular, the *amplitude* of quality fluctuations and the *duration* for which stable quality video is available at a node heavily depend on the nature of peer departures in the system. In this paper, we first propose a service quality model to quantify the quality and stability of a video stream in a P2P streaming environment. Based on this model, we further develop tree construction algorithms that ensure that every peer in the collaborative network receives a video stream with a statistical reliability guarantee on quality. A key advantage of the proposed approach is that we can now *explicitly* control the quality and stability of the video stream supplied to every node. This is the fundamental difference of our approach from existing approaches that provide stream stability by over-provisioning resources allocated to every peer. Also, the proposed tree construction schemes decide the position of a node in the delivery tree based on both its estimated reliability and upstream bandwidth contribution while striving to minimize the overall load on the server. Our simulations show that our algorithms use the server resources very efficiently while significantly improving the video stability at peers.

1. INTRODUCTION

Multimedia streaming has become one of the most popular applications on the Internet, e.g., services like google video and YouTube receive more than 100 million hits per day. With such high demand and broad scope for more growth (in markets like IP-TV), it is essential to have scalable streaming solutions. P2P streaming fits this niche nicely to provide scalable streaming services and has been a very active research area in recent years [5, 7, 8, 15, 16, 18, 21, 23, 25]. In P2P streaming techniques for live video delivery, all peers form a *delivery tree* rooted at a server. Every non-leaf node in the tree relays the video frames it receives from its parent nodes to its *children*. In other words, the server sends out a number of streams that are then delivered to all peers via the delivery tree. A key advantages of P2P streaming is that the system capacity scales with the number of peers since every peer contributes bandwidth resources. When a peer joins a session, peers with spare upstream bandwidth are assigned as parents to the incoming client using a centralized or a distributed scheduling mechanism.

However, one of the key challenges in P2P streaming is handling unscheduled peer departures. When a peer leaves, due to reasons like preemption of the relay process by a higher priority network process, the video stream to its children is disconnected. This disruption cascades down the tree and causes a ripple effect of stream disruptions through the delivery tree. Such node failures have two important implications on the streaming system:

- **Video Quality Fluctuation:** The video quality (in terms of bit-rate) experienced by a peer, depends on the total upstream bandwidth contributed by its parents. Due to peer dynamics, the parent set of a peer may keep changing with time thus leading to fluctuating received bit-rate. So, with unscheduled peer departures, it is very hard to control the *amplitude* of quality fluctuations experienced by each client.
- **Unpredictable Stream Disruption Frequency:** Stream disruption frequency (SDF) at a node is the reciprocal of the time elapsed between two successive failures among its parents such that the quality experienced by the node falls below a threshold during each failure. In other words, SDF is reciprocal of the average time duration for which the video quality at a node is equal to or above a threshold. This frequency depends on the reliability of each parent in its parent set. When the parents are reliable, the SDF tends to be small. However, having an unreliable parent set leads to frequent stream disruptions or quality fluctuations at a peer. Since the SDF cannot be explicitly controlled, the

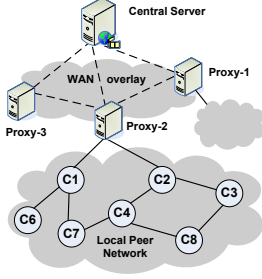


Figure 1. A Hybrid Collaborative Streaming Architecture.

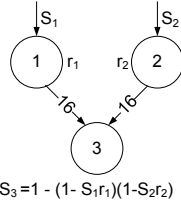


Figure 2. Parent set with 16 unit contribution

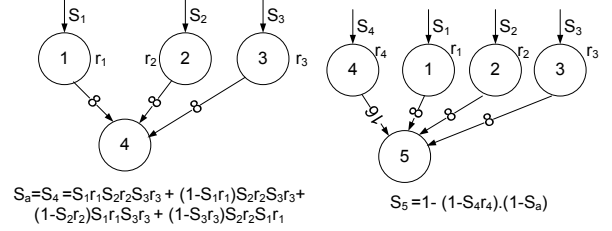
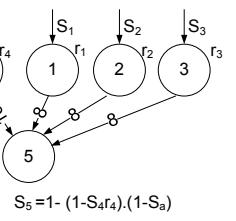


Figure 3. Parent set with 8 unit contribution

Figure 4. Parent set with 16 and 8 unit contribution



duration for which a peer can play uninterrupted, minimum quality video presents another dimension of instability in the video.

Video quality and stream disruption greatly affect the perceived quality. Several results in classical video streaming show that fluctuating quality leads to poorer perceived quality than constant low quality [22]. Frequent playback disruption further downgrades the perceived quality. To make P2P streaming techniques feasible in real applications, we must have control over video quality and the probability of stream disruption. To our best knowledge, most P2P streaming systems do not explicitly control these parameters. To this end, this paper makes two important contributions:

1. Define a QoS model for P2P streaming, which takes both the stream quality and the disruption frequency into account.
2. Develop a family of tree construction heuristics which take both the reliability and the upstream bandwidth of a peer into account while constructing a delivery tree that can conform to the proposed QoS model.

In this paper, we focus on the construction of delivery trees that can provide a statistical quality and stability guarantee. The guarantee is determined by the streaming service provider and could serve as a basis for billing. The rest of the paper is organized as follows. In Section 2, we describe the streaming system model and the proposed QoS model. In Section 3, we formulate the tree construction problem as a variation of a non-linear, multi-dimensional knapsack problem. In Section 4, we discuss the design and tradeoffs of the proposed tree construction algorithms. In Section 5, we evaluate the proposed tree construction techniques in detail. In Section 6, we present related work and discuss how the proposed methods can be integrated into other popular P2P streaming techniques. We conclude the paper in Section 7.

2. SYSTEM MODEL

In this section, we first introduce our streaming model and related assumptions. We then discuss the proposed QoS model in detail.

2.1. Streaming Model

We use a two-level hybrid streaming architecture [6], as shown in Figure 1. At the upper level, we use an overlay network to deliver videos from a central server to proxy servers [2, 4]. The overlay network supplies a stable stream to a local network and provides the base for service reliability. At the lower level, a proxy server transmits video data to clients with the assistance of a *collaborative peer network*. This peer network consists of clients who commit storage, computing and bandwidth resources to assist the proxy server. For example, a collaborative peer network is shown as a multicast tree rooted at proxy server *Proxy-2* with clients C1 through C8 as nodes in the tree in Figure 1. The proxy server provides a directory service for clients in a local network and schedules streaming sessions to deliver video data to clients. The two-level model provides scalability at the WAN level through the use of proxy servers and scalability at the local area level through the use of P2P streaming. We point out that the two-level hybrid streaming model is used for its scalability and does not affect the tree construction algorithms proposed in this paper. We now discuss key assumptions in the two-level streaming model below.

Node Upstream Bandwidth: A node i contributes a limited upstream bandwidth, denoted as u_i , to P2P streaming [15,23]. Meanwhile we assume that sufficient downstream bandwidth is available between peers in a collaborative network.

Node Reliability: We define the reliability of node i as the probability that it is able to deliver a stream to others (denoted as r_i). We assume that the proxy has a mechanism to estimate the node reliability in its local community. An effective technique would be to learn the reliability of a peer based on history [19]. Intuitively, when a delivery tree is constructed based on only node reliabilities, we would have the following a property: on a path from the root to a leaf, the reliabilities of nodes decrease. This property helps us reduce the chances of large scale service disruption in the tree.

Video Coding: In our design, we use Multiple Description Coding (MDC) [16] for video coding. MDC uses redundant encoding to provide service robustness and hence is naturally fit for dynamic and unreliable P2P streaming environments. A video is encoded into multiple *descriptions*, and each description can be decoded independently at a destination. Consequently playback quality is proportional to the number of descriptions received. For ease of illustration, we assume that the bandwidth of each description is one unit. We measure the upstream bandwidth u_i contributed by node i as the number of descriptions that it can relay to other peers.

2.2. A QoS Model For P2P Streaming

Since the perceived QoS of a video is determined by both the quality and stability of streaming, we define an integrated QoS model as follows:

(i) **Quality Measure:** The *playback quality* Q_i at a node i is defined as the number of descriptions received by the node.

(ii) **Reliability Measure:** This QoS dimension provides a statistical, quantitative measure for the stability of a video stream. The stability of an input video stream, called *stream reliability*, is the probability that quality measure Q_i of the stream will *not* drop below a threshold q , due to node failures. In other words, it is the probability that node i can receive at least q descriptions from its parents. It is denoted as S_i^q . (For convenience, we drop the suffix q whenever it is obvious.) For example, the stream reliability at a direct child of a proxy server is 1. The stream reliability at a node is intimately tied to the SDF experienced by the node. Intuitively if the stream reliability at a node is high, then the SDF should be low because the probability that the stream quality drops to less than the threshold q is low.

We now discuss the details of calculating the stream reliability S_i^q at a node. Figures 2, 3 and 4 show the calculation of the stream reliability for $q = 16$, and peer upstream bandwidth contribution can either be 8 or 16 descriptions. Note that the stream reliability at a node is a function of the stream reliabilities of the input streams of all parents and the reliability of the parents. We now define the generalized expression of stream availability at node i . We divide the parent set of node i into two subsets, ϕ_1 and ϕ_2 , based on the bandwidth contribution of parents. Set ϕ_1 includes class-1 parents who are able to provide upstream bandwidth of q , $|\phi_1| = k_1$, and set ϕ_2 includes class-2 parents who are only able to provide a bandwidth of b , where $b < q$, and $|\phi_2| = k_2$. (For ease of illustration, we assume all these k_2 class-2 parents provide the same bandwidth b .) As a result, a stream with quality q is available at node i if one of its class-1 parents are available or at-least k'_2 of class-2 parents are available (where $b \cdot k'_2 \geq q$). We use ϕ_{21} to denote the set of class-2 parents who are available, and use ϕ_{22} to denote those who are unavailable. We know $k^* = |\phi_{21}| \geq k'_2$. So the stream reliability at i is calculated as:

$$S_i^q = 1 - \prod_{m \in \phi_1} (1 - S_m \cdot r_m) + \sum_{n=1}^{C_{k_2}^{k^*}} \prod_{j \in \phi_{21}} (S_j \cdot r_j) \cdot \prod_{t \in \phi_{22}} (1 - S_t \cdot r_t) \quad (1)$$

3. PROBLEM FORMULATION

In this section, we formulate the tree construction problem in two steps. In the first step, we describe the fundamental property that every solution (in our case, a delivery tree) should satisfy. In the second step, we use an objective function to identify the best solution from the set of possible solutions from the first step.

Step-1: In the previous section, we define a QoS model that quantifies the (*in*)*stability* of a video stream. The first goal of our tree construction algorithms is to provide statistical bounds on quality and instability of the video stream at a node. To this end, we place a lower bound on both the quality and stream reliability dimensions as follows:

a) The *quality constraint*, denoted as Q_{min} , is a lower bound on the number of descriptions which a client needs to receive for playback.

b) The *reliability constraint*, denoted as R_{min} , is a lower bound on the stream reliability of the input stream received by a peer, while satisfying the quality constraint.

In other words, each peer should receive at least Q_{min} descriptions with a minimum probability of R_{min} . By bounding the stream reliability to at least R_{min} , we indirectly bound the SDF at every node. Another important consequence of the reliability constraint is that, some nodes may need an input stream with bandwidth Q_i strictly greater than Q_{min} . This is because, multiple backup paths will be needed to achieve a given R_{min} , if the parent nodes are not reliable enough. We name the extra bandwidth used ($Q_i - Q_{min}$) as the *backup bandwidth* used by node i . Backup bandwidth provides redundancy in data delivery to achieve the required reliability. By this way, we can trade upstream bandwidth to provide reliability. To ensure stable video quality, a client always uses only Q_{min} descriptions at any point even if it receives more than Q_{min} descriptions. Q_{min} and R_{min} are parameters decided by the video service provider and may be used as a basis for billing.

Step-2: For a given set of nodes, we can have multiple solutions that satisfy the quality and reliability constraints. One way of differentiating multiple solutions is based on the upstream bandwidth contributed by the proxy server in a local network. The proxy bandwidth contribution is heavily dependent on how the delivery tree is constructed. For instance, if high bandwidth contributors are present as leaves, then the number of proxy server streams required to support all clients will consequently tend to be higher than a solution where such high contributors are present as internal nodes. At the same time, there is a complex interaction with the reliability constraint. By placing reliable nodes in the interior of the tree, we may be able to reduce the peer bandwidth wasted as backup and indirectly reducing the proxy bandwidth usage. We choose a solution that uses the least proxy bandwidth resources. This objective function can be justified by the following arguments: 1) The total upstream bandwidth available at the proxy is fixed and it needs to be shared across multiple live video streams served by the proxy. To maximize the number of clients that can be supported by the system, it is necessary to utilize the proxy bandwidth as efficiently as possible. 2) An overload situation occurs during a flash-crowd like scenario where the proxy bandwidth is exhausted and admission control needs to kick in. Minimizing the proxy bandwidth usage per stream, at every time instant, postpones an overload event by utilizing the peer bandwidth as efficiently as possible.

The above two-step goals can now be summarized as an optimization problem as follows:

$$\text{Minimize } \int_0^T B(t)dt \quad (2)$$

subject to: $Q_i \geq Q_{min}, S_i^{Q_{min}} \geq R_{min}, \forall i \in (1, 2, \dots, n)$,

where $\int_0^T B(t)dt$ represents the area under the *proxy bandwidth curve* in a plot of proxy bandwidth utilization over time, T is the duration of the video, $B(t)$ is the proxy bandwidth used at time instance t , R_{min} is the reliability guarantee parameter, Q_{min} is the quality guarantee parameter, and n is the number of clients in the system.

Optimal Solution: To minimize the area under the curve $B(t)$, we should use the minimum required proxy bandwidth at any given instance. On every node arrival/departure (termed as *events*), we should adjust the tree to ensure optimality. The “optimal” position of a node is a function of the reliability and upstream bandwidth of all the nodes in the system, including itself. We observe that the solution to this problem does not exhibit the *optimal substructure* property and hence the tree has to be rebuilt on every *event*. Finding an optimal solution is further complicated by the heterogeneity of peer upstream bandwidth contribution and peer reliability [14]. Also note that the reliability constraint is non-linear as the stream reliability is a non-linear function. The above problem can be mapped on to a non-linear minimization knapsack problem with multidimensional constraints, which is known to be NP hard [13].

Heuristics: Since the optimal solution is computationally infeasible to find for large problem sets, we resort to multiple levels of heuristics. The first class of heuristics, called *Static Heuristics*, are aimed at creating a near optimal tree and have high tree reconstruction cost. The second class of heuristics, *Online Heuristics*, focus on constructing an efficient tree incrementally with low reconstruction cost. This categorization is based on a fundamental tradeoff between tree reconstruction cost and solution quality. We explain the heuristics in further detail in the next section.

4. TREE CONSTRUCTION ALGORITHMS

As briefly discussed in the previous section, we propose two classes of heuristic algorithms for the tree construction problem. The first class of algorithms, that use static heuristics, aim at finding near-optimal solutions in polynomial time. We further design a second class of heuristics, namely *online heuristics*, because static heuristics cannot be used in practice due to their high tree reconstruction cost. We use near-optimal solutions obtained from static heuristics to guide online heuristics to improve the quality of delivery trees. We first introduce the key terminologies used in the remainder of this paper, and then discuss the proposed static algorithms in detail in Section 4.1. Using the static algorithms as a foundation, we then develop online algorithms in Section 4.2.

Definition-1: A *Full Stream* is a combination of a set of stream(s) contributed by one or more nodes, which can provide a quality of at least Q_{min} with stream reliability of at least R_{min} . For example, in Figure 1, the streams from nodes 3 and 4 to node 8 can form a full stream. We use F_i to represent the full stream to the i^{th} peer in the system. We use $Q(F_i)$ to denote the quality of the full stream, and it is the sum of upstream bandwidths provided by each node contributing to the full stream. $R(F_i)$ represents the stream reliability of the full stream and is obtained as discussed in Section 2.2. Based on the definition of a full stream, it is clear that $R(F_i) \geq R_{min}$ and $Q(F_i) \geq Q_{min}$. The streams originating from a proxy server are full streams with quality Q_{min} and reliability 1. The *stream reliability* at a node is defined as the stream reliability of the full stream supplying a video to the node. Therefore, the full stream abstraction provides a convenient method to represent the parent set (denoted as $\{P\}_i$) of node i , the upstream bandwidth contributed by each parent to node i , the stream reliability at node i , and the video quality at node i .

Definition-2: The *level* $L(F_i)$ of a full stream F_i is the maximum of the levels of its contributing nodes. The level of node i is the maximum number of hops from i to reach the root. For example, in Figure 1, the level of node 8 is 3.

4.1. Static Algorithms

The static algorithms are intended to provide efficient, polynomial algorithms to efficiently search the solution space for a near-optimal solution. We use static algorithm for two main purposes: (i) provide tree construction hints for the online algorithms and guide them towards a high quality solution; (ii) provide a baseline to evaluate the online counterparts for large problem sizes where finding an optimal solution is infeasible. The main idea behind static algorithms is to construct a delivery tree from scratch on every *event*, i.e., for every node arrival/departure. Therefore, the tree structure depends only on the current set of nodes, denoted as $\{N\}$, and is independent of the arrival times of the nodes in $\{N\}$. This gives static algorithms the complete freedom to change the tree structure to adapt to the membership changes of set $\{N\}$.

We use observations from the optimal tree structure to design the static algorithms. First, the static algorithms construct the tree, node by node, level by level. The position of node $k - 1$ is fixed before the position of the next node k is decided. On the other hand, the optimal algorithm searches all possible combinations of positions for all nodes to decide the best choice. This difference makes the static heuristic a polynomial time algorithm. Secondly, since the position of a node in the optimal tree is a function of its reliability and upstream bandwidth, the static algorithms should also exhibit this property. Finally, the upstream bandwidth contributed by nodes higher in the tree should be *consolidated* to form full streams that support nodes inserted later. Based on the above discussion, the following generalized algorithm template can be formalized for a static heuristic. Let $\{N\}$ be the set of nodes to be inserted into the tree and let T represent the tree being constructed.

Algorithm 1 staticHeuristicTemplate($\{N\}$)

- 1: **while** all nodes in $\{N\}$ are not present in T **do**
 - 2: Start f proxy streams and initialize T to be empty
 - 3: **while** more full streams available **do**
 - 4: Choose next node i from $\{N\}$
 - 5: Insert i into T by assigning a full stream to i .
 - 6: Consolidate unused upstream bandwidths including u_i to form full streams
 - 7: **end while**
 - 8: $f = f + 1$
 - 9: **end while**
-

Steps 4 and 6 form the basic components of the static algorithms described in this section. Step 4 governs the order in which the nodes must be inserted into the tree to maximize the utilization of peer bandwidth. Step 6 decides how this peer

bandwidth must be efficiently converted to full streams to support nodes inserted later. We now present three methods to implement Step 3 in Section-4.1.1 and one method to implement Step 1 in Section-4.1.2.

4.1.1. Measuring Node Contribution

The contribution of a node, in a P2P system, can be measured by how effectively it can support other peers. In our context, the contribution of a node is a function of both its upstream bandwidth contribution and reliability. Due to heterogeneous upstream bandwidths and node reliabilities, each node contributes differently to the system. A rule of thumb that we use in designing the static heuristics is that high contribution nodes should be placed higher in the tree. This decision can be justified by the fact that when inserted higher in the tree, high contributors can be used to support more peers and hence alleviating the proxy load. Therefore, we need to quantify the contribution of a node. However, the contribution of a node is not a directly measurable quantity and is function of both its reliability and upstream bandwidth. To address this issue, we design three heuristics to quantify node contributions based on both node reliability and upstream bandwidth.

Node Amplification Capacity (NAC) The amplification capacity (α_i) of a node i is defined as the expected bandwidth contribution of the node to the system, and is calculated as:

$$\alpha_i = u_i \cdot r_i \quad (3)$$

If we use NAC to construct the tree, then α_i should decrease as we move from the root to leaves. Although this heuristic captures the combined effect of upstream bandwidth and reliability in general, it may lead to poor decisions in certain special cases where node reliability r_i is very low and node upstream bandwidth u_i is very high. The multiplication masks such a situation such that we may place an unstable node high in the tree.

System Bandwidth Gain (SBG) For node i to produce one or more full streams, its upstream bandwidth u_i may require b_i units of *support bandwidth* from other peers. We define support bandwidth b_i as the bandwidth required by node i from other peers to produce a full stream, which is required either when node i does not have enough upstream bandwidth to produce a full stream and/or cannot produce a stream with sufficient reliability. Therefore, b_i depends on both the upstream capacity u_i and reliability r_i of node i . It is clear that node i effectively contributes only $\beta_i = u_i - b_i$ units to the system to support other nodes. Therefore, we use β_i to summarize the contribution of node i here.

Support bandwidth b_i also depends on other full streams and nodes that are available in the system. We now present a method to calculate b_i based on the current tree configuration. We then use this estimate to calculate β_i . Let $\{SP\}_i$ be the smallest set of parents that can contribute b_i and let b_{ik} be the bandwidth provided by node k ($k \in \{SP\}_i$) to i . The set $\{SP\}_i$ is found by constructing a full stream in which node i is a member. We will discuss the details of full stream construction in Section 4.1.2.

$$\beta_i = u_i - \sum_{k \in \{SP\}_i} b_{ik} \quad (4)$$

Note that we do not actually assign the nodes in $\{SP\}_i$ as parents to any node but instead just use them to estimate b_i .

System Amplification Gain (SAG) The *System Amplification Gain* (γ_i) heuristic is an enhancement to the NAC heuristic to account for the loss of information caused by the scalar multiplication operation. This heuristic measures the change in the amplification capacity of the system after node i has been inserted. It is calculated as:

$$\gamma_i = u_i \cdot r_i - \sum_{k \in \{SP\}_i} b_{ik} \cdot r_k \cdot R(F_k) \quad (5)$$

So it essentially captures the combined effect of NAC and SBG.

4.1.2. Constructing Full Streams

Let $\{P\}$ represent the set of nodes in the tree, which have spare upstream bandwidth not assigned to any full streams. The *full stream construction* problem deals with how to form (not necessarily disjoint) subsets from $\{P\}$ such that the nodes in each subset can provide a full stream to support another node. If ξ_k represents the k^{th} subset in $\{P\}$ then:

$$Q(\xi_k) \geq Q_{min}, \quad R(\xi_k) \geq R_{min}, \quad \bigcup_{k=1}^p \xi_k \subseteq P$$

where p is the number of subsets formed. $Q(\xi_k)$ and $R(\xi_k)$ denote the quality and reliability of the full stream, respectively. To minimize the load on a proxy, we should produce as many full streams as possible from a given set. We propose to use a *best-fit* algorithm wherein each full stream that we produce just satisfies the reliability and quality constraints with minimum over-provisioning.

The main idea behind the best-fit algorithm is that we group high contributors with low contributors to form a full stream. First, we sort set $\{P\}$ based on one of the node contribution metrics discussed in Section 4.1.1. Now, we add one node at a time, alternatively from either end of the sorted $\{P\}$, into a set ξ_k until the nodes in ξ_k can form a full stream. We always start by adding a low contributor followed by a high contributor. Next, in order to attain the least overprovisioning within the set ξ_k , we ensure that every element in the set is indispensable. We do this by removing each element $x \in \xi_k$ and checking if $\xi_k - x$ can form a full stream. If so, we add x back to $\{P\}$. After all elements in the set ξ_k have been tested for exclusion, we start constructing the next full stream set ξ_{k+1} with remaining elements in $\{P\}$. This process is repeated until no more full streams can be formed from $\{P\}$. We use the above algorithm in the SAG and SBG heuristics to calculate support bandwidth and use NAC as the node contribution measure.

4.2. On-line Tree Construction Algorithms

Rebuilding the tree for every node join/leave is not practical because: (i) management costs will become prohibitive; (ii) the stability of the tree is affected by constant node shuffling. In an online algorithms, we build a delivery tree incrementally with minimal disturbance to the existing tree. However, by only concentrating on reducing reconstruction costs the quality of the solution may quickly degrade. Hence the proposed online algorithms use hints provided by the static algorithms to explore the tradeoffs between adjustment cost and solution quality.

Three operations are required for dynamically managing a delivery tree. First, we need to insert a peer into an existing tree when it joins the live multicast session. We also need delete a node to update the tree when a node leaves the system. Sections 4.2.1 and 4.2.2 will discuss the proposed insert and delete heuristics in further detail. In addition, we also need to construct full streams to fully utilize peer bandwidth.

4.2.1. The Insert Operation

The insert operation assigns a full stream to an incoming node. In Section 4.1, we observed that the node position in an optimal tree is a function of both its reliability and upstream bandwidth. To satisfy this property, the node should be inserted in a position such that its resources provide the maximum benefit the system. The novel feature of the online insert algorithm is that it uses hints from the *near-optimal* solution produced by the static algorithm (described in Section 4.1) to search for a suitable position for an incoming node. The online insert algorithm is aimed at producing a solution of good quality while not incurring too much reconstruction overhead. Two cases in an insert operation are **Case 1: A Full Stream is available in the system;** **Case 2: No Full Stream is available.**

Case 1. Full stream available : Let l_s be the level of the available full stream and let \hat{l}_i be the level suggested by the static algorithm for node i . We use \hat{l}_i as the hint provided by the static algorithm to guide the insert operation. The online algorithm inserts the node at a level l such that $|l - \hat{l}_i|$ is as small as possible. When multiple full streams are available, we choose the full stream which yields the least $|l - \hat{l}_i|$. A node x is said to *replace* another node y if node x acquires node y 's parents and children. Node y is said to be "replaceable" by node x if such a replacement does not cause violation of both reliability and quality constraints in any node in the subtree of node y . Based on the values of l_s and \hat{l}_i , there are two cases, *Case 1a: $l_s \leq \hat{l}_i$* and *Case 1b: $l_s > \hat{l}_i$* .

- *Case 1a. $l_s \leq \hat{l}_i$:* In this case, we linearly search levels l_s through \hat{l}_i to find a replaceable node for i . The replaceable node satisfies an additional constraint that it has higher contribution than node i . If we find a replaceable node, we insert node i in its position and assign the replaced node to the full stream in level l_s . If we cannot find a replaceable node, then node i is assigned to the full stream in level l_s .

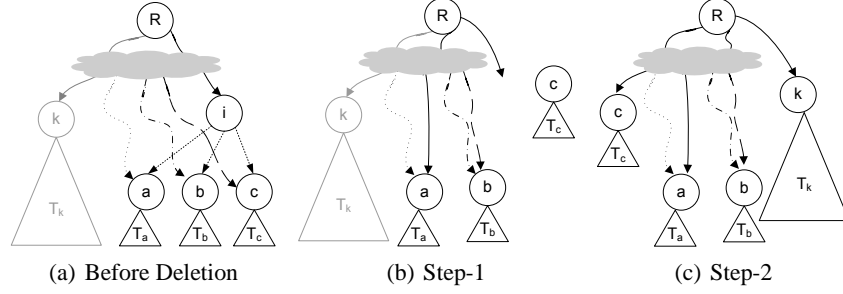


Figure 5. The Online Delete Algorithm

- *Case 1b. $l_s > \hat{l}_i$:* In this case, we use following procedure to find a suitable level for node i . Let l_i be the level in the tree where there exists at least one node j such that $r_j \leq r_i$ and $u_j \leq u_i$. In short, node j contributes lesser and is replaceable by node i . The online algorithm linearly searches levels \hat{l}_i through l_i to find a replaceable node. The replaceable node satisfies an additional constraint that it has lower contribution than node i . In the worst case scenario, node j is chosen as the replaceable node. After finding the replaceable node, i is inserted in its position. The replaced node is then *re-inserted* into the tree, as Case 1 again. This cascade of *replace* and *re-inserts* continue until a replaced node is re-inserted in level l_s and then the full stream will be assigned to that node.

Case 2. Full stream is not available: In this case, a new stream is started from the proxy. Since this new proxy stream has 100% reliability and is an expensive resource, we want to assign it to a node that can maximize system-wide benefit. To select the node that will receive the new stream from the proxy, we use the node contribution metrics defined in Section 4.1. We select the next highest contributor j which is not already connected to the root as the new level 1 node. Note that this would have been the choice of the static algorithm. If $i = j$, i.e., the next highest contributor is the incoming node itself, then the new full stream is assigned to the incoming node. If $i \neq j$, then node j is assigned the new proxy stream and thereby freeing up the full stream it was originally using. Since a full stream is now available in the system, node i is inserted as Case 1.

4.2.2. The Delete Operation

When node i is deleted, the full streams used by the children of i (denoted by the set $\{C\}_i$) become “incomplete”. So the crux of the delete operation is to find a valid parent set for all nodes in $\{C\}_i$ such that both the Q_{min} and R_{min} constraints are satisfied at these nodes and all nodes in their subtrees.

The online delete algorithm operates in two steps as in Figure 5. In the first step, we consolidate the upstream bandwidths from the full stream used by node i (F_i) and the “incomplete” full streams of nodes in the set $\{C\}_i$ with the existing peer upstream bandwidth available in the system. The goal of this consolidation step is to produce as many full streams as possible to support nodes in the set $\{C\}_i$. The best-fit algorithm is used for finding these new full streams. However, an additional constraint is imposed to this scenario. If s_a was the original stream reliability of the full stream to node $a \in \{C\}_i$, the new full stream should have stream reliability of at least s_a . This is to ensure that the R_{min} constraint is not violated in the subtree T_a .

The second step is necessary when sufficient peer bandwidth is not available in the system to support all nodes in the set $\{C\}_i$. In this case new proxy streams have to be started. Since every proxy stream is a valuable resource capable of benefiting the entire system, we use the algorithm described in Section 4.2.1 to find a level 1 node. Let c be a node $\in \{C\}_i$ that is yet to be inserted into the tree. We impose an additional constraint that the full stream currently used by the selected level 1 node should have stream reliability of at least S_c to ensure that the stream reliability constraint is not violated in subtree T_c .

5. PERFORMANCE EVALUATION

In this section, we present in-depth evaluation of the proposed tree construction heuristics using simulations. We first discuss the simulation setting and parameters, and we then evaluate the proposed schemes from the view of a proxy server and a client, respectively.

Distribution	Scenario
Uniform (UFM)	General environment
Heavy Tail High (HTH) : 1-Pareto($a=1, b=0.1$)	Majority of reliable nodes
Heavy Tail Low (HTL) : Pareto($a=1, b=0.2$)	Majority of unreliable nodes
Normal High (NoH) : Normal($\mu = 0.9, \sigma = 0.1$)	Reliable environment
Normal Low (NoL) : Normal($\mu = 0.3, \sigma = 0.1$)	Unreliable environment
biNormal (BiN) : ($\mu = 0.8/0.2, \sigma = 0.1$)	50% nodes reliable

Figure 6. Probability Distributions and their scenarios

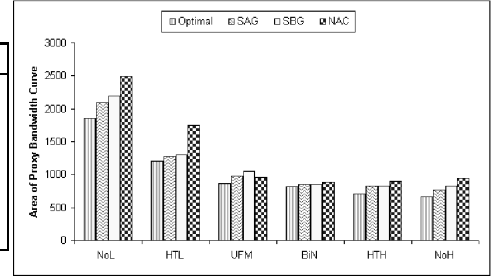


Figure 7. Comparison of Static and Optimal Algorithm

5.1. Simulation Setup

To evaluate the proposed tree construction algorithms under various client characteristics and problem sizes, we first generate client session traces. These traces contain arrival, departure timestamps for every client along with its upstream bandwidth contribution and estimated reliability. We choose the upstream bandwidth of clients based on a gaussian distribution. In particular, we assume that client upstream bandwidth, measured as the number descriptions that a node can relay, fall into 5 categories (8, 16, 24, 32, 40) with the probability distribution (0.1, 0.2, 0.4, 0.2, 0.1). To model client reliability, we follow an approach used in [19] to emulate real-life node reliability scenarios as summarized in Figure 6. Next, we use a poisson model to generate timestamps for node arrivals. Specifically, assume that the client inter-arrival time follows an exponential distribution with a mean rate of $\lambda = L/n$, where L is the length of the video, and n is the number of clients in the system. The departure time of node i is a random variable based on a normal distribution with mean $r_i \cdot L$. We then replay the trace, as generated above, to multiple tree construction algorithms and evaluate the performance as described in the following subsections. For all simulations, we set the reliability constraint (R_{min}) to 0.6, the quality constraint (Q_{min}) to 16 descriptions, and the video length L to 1000sec.

5.2. The Proxy Server Perspective

In the first set of simulations, we compare the solution produced by the static heuristics with the optimal solution produced by an exhaustive search algorithm. Figure 7 shows the proxy bandwidth usages of the three static algorithms and the optimal solution. First, we can see that the solution quality of static algorithms is close to that of the optimal algorithm. The static algorithms insert the nodes based on the node contribution measures and hence make locally optimal choices. On the other hand, the optimal solution selects an insertion order globally. The second observation from Figure 7 is that SAG heuristic outperforms the other two static heuristics except the case of UFM distribution. In that case, the scalar multiplication captures the contribution better than using a backup bandwidth-based heuristics like SAG or SBG. However, we see that the NAC heuristic performs worst for the distributions with low reliability such as HTL and NoL, where the scalar multiplication masks the extreme low reliability and causes sub-optimal decisions. The third observation is the relationship between the proxy bandwidth usage and the average peer reliability distribution. In Figure 7, the average reliability of the peers increases from left to right, i.e. NoL has the least average peer reliability and NoH has the maximum average peer reliability. The proxy bandwidth usage reduces when peers are more reliable because less peer bandwidth is wasted on backup leading to more efficient use of peer bandwidth. This in turn reduces the proxy bandwidth usage.

In the second set of simulations, we compare the performance of the static algorithms and online algorithms for a problem size of 1000 peers. The proposed online tree construction operations use hints from the static algorithm as feedback. In our simulations, we use the SAG heuristic to provide the feedback. To evaluate the effectiveness of the feedback mechanism, we compare the performance of the online algorithm with a simple insert and delete algorithm which does not take hints from the static algorithm. The *simple-insert* algorithm, for inserting a node i , operates as follows: When a full stream is available in the system, we start searching from level 1 until we find a node j (where $r_i \geq r_j$ and $u_i \geq u_j$) or the full stream. In the first case, node i replaces node j and node j is reinserted into the tree. In the second case, the full stream is assigned to node i . When a full stream is not available, we start a full stream from the proxy. To assign the new level 1 node, we search starting from level 1 to find a node j such that $r_j \geq r_i$ and $u_j \geq u_i$. We assign node j along with its subtree to the newly-started proxy stream, and assign node i to the full stream that was previously delivered to j . The *simple-delete* algorithm, for removing a node i from the tree, operates as follows: When node i is deleted, the first

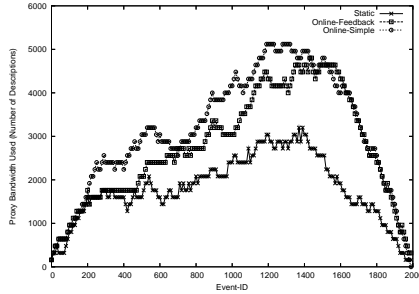


Figure 8. Proxy Bandwidth Usage Comparison (r_i : Heavy Tail Low)

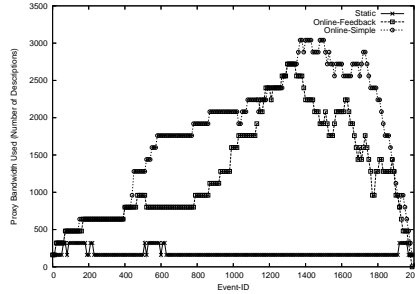


Figure 9. Proxy Bandwidth Usage Comparison (r_i : Heavy Tail High)

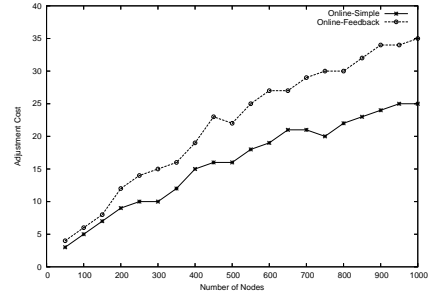


Figure 10. Tree Adjustment Cost (r_i : Heavy Tail High)

step of full stream consolidation is done as in the online-feedback delete algorithm explained in Section 4.2.2. Then, the second step of finding a level 1 node is done as explained in the *simple-insert* algorithm when no full stream is available. Figure 8 and 9 show the proxy bandwidth usage for the static (SAG), online-feedback and online-simple algorithms for two reliability distributions. First, we find that the online-feedback algorithm does improve the performance, compared with not taking feedbacks, This is primarily because of the more efficient use of the proxy streams and the better selection of replacement nodes, which are direct benefits of the hints from the static algorithm. Also, we see that the proxy bandwidth usage again is dependent on the reliability distribution of peers.

In the last set of simulations, we evaluate tree reconstruction costs of the proposed static and online algorithms. To compare these solutions, we define a metric called *Adjustment Cost*, which is the average number of changes made to a tree for an insert or delete operation. For the static algorithms, since we reconstruct the tree on every node arrival or departure, the adjustment cost is directly proportional to the number of peers in the system. Therefore, the cost increases linearly with the number of peers. Figure 10 shows how the tree adjustment cost of the online-feedback and online-simple algorithms varies with problem sizes. We see that both algorithms have low reconstruction cost and rise sub-linearly with problem size. The increase in adjustment cost can be explained by the increase of replace-reinsert steps for the insert operation and parent reallocation steps in the delete operation. We also see that the adjustment cost is higher for the online-feedback case. Figure 11 shows the average tree adjustment cost per node. Clearly, the adjustment cost of the online algorithms is very low, compared with that of the static algorithms; more importantly, it does not increase with the problem size and hence is more scalable than the static algorithms.

5.3. The Client Perspective

Here we focus on the effect of the proposed algorithms on peers. In the first simulation, we study the effect of the quality and reliability constraints on the perceived quality at a peer. We randomly select a peer and observe the quality fluctuations due to the failures of parent nodes. We compare the online-feedback tree construction algorithm with an algorithm that constructs the delivery tree only based on arrival times. This algorithm does not use the reliability information of nodes and just tries to provide $Q_{min} = 16$ descriptions to every node. Figure 12 shows that the perceived quality is more stable in our tree construction scheme than in the scheme that considers only upstream bandwidth. Note that a dip in quality is caused by a failure of parent node. The online delete algorithm is invoked to find new parent(s) when a parent fails. Since our schemes reserve backup bandwidth, the perceived quality drops below Q_{min} on only one occasion and hence significantly lowering the stream disruption frequency. Figure 13 quantifies the backup bandwidth for various reliability distributions. The backup bandwidth overhead was calculated as $\frac{B - Q_{min} \cdot n}{Q_{min} \cdot n}$ to reflect the excess bandwidth used, compared to the scheme that supplies only Q_{min} to every node, where B is the total bandwidth used. We see that the overhead varies based on the peer reliability distribution. For a uniform distribution case, the overhead is not very high. To provide the stability shown in Figure 12, we pay the costs shown in Figure 13. We observe that on an average, this cost is not prohibitively high, given that each peer receives a stable stream.

6. DISCUSSION AND RELATED WORK

In this section, we present related work in two key areas relevant to our work, namely, *multimedia streaming architectures* and *handling node failures in P2P streaming*. We also discuss how the proposed algorithms can be extended and incorporated with other popular streaming models.

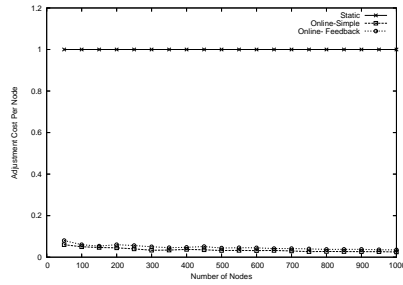


Figure 11. Average Adjustment Cost Per Node

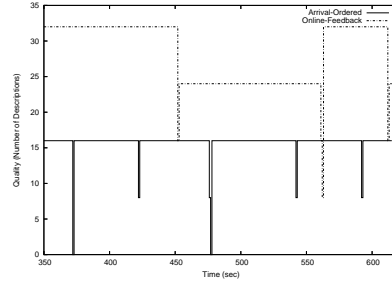


Figure 12. Perceived Quality at Client for online-feedback algorithm

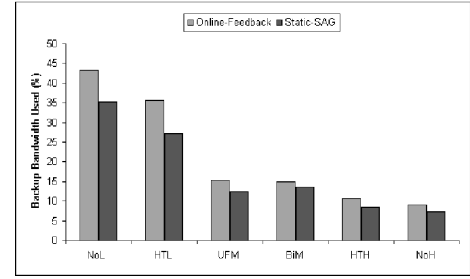


Figure 13. Backup Bandwidth Utilized for Online-Feedback and SAG Heuristics

Multimedia Streaming Architectures: Multimedia streaming is one of the most popular applications on the Internet. Many video streaming approaches have been proposed in the past two decades. Current video streaming systems can be classified into three types of architectures: client-server models on unicast/multicast networks, (e.g., batching [1], patching [11], periodic broadcasting [10]), CDNs with dedicated proxy servers [17], and P2P streaming approaches harvesting peer resources for scalability (e.g., SRMS [3], PROMISE [8], ZIGZAG [21]). Due to the lack of multicast capability in current IP networks, client-server approaches are not scalable to support large-scale streaming, while CDNs usually require extremely high investment on delivery infrastructures and demand a large amount of resources to achieve scalability. As a result, it often suffers the flash crowd problem [12, 20] due to its fixed resource allocation. On the other hand, P2P streaming approaches harvest peer resources for scalability and availability and thus service more clients with low costs. Many P2P streaming approaches have been proposed to support large scale real-time streaming such as, CoopNet [16], P2Cast [7], Dagster [15] PROMISE [8, 23, 24], SpreadIt [5], ZIGZAG [21], etc. These approaches exploit peer buffers and streaming capacities to assist servers for delivering videos to a large number of clients. Since peers may randomly leave and usually have *limited upstream bandwidth*, it is extremely challenging to sustain the streaming quality using common P2P approaches.

P2P streaming and server/proxy-based streaming techniques lie at the two ends of the solution spectrum. Current P2P streaming schemes provide scalability by trading quality and stability of streaming sessions. On the other hand, server/proxy-based schemes can support stable and high quality video sessions but do not scale well. Hybrid approaches have been proposed to combine the advantages of both server-based and P2P streaming models. In this paper we use a two-level hybrid streaming model as proposed in [6]. CoopNet [16] is another hybrid architecture that addresses the flash crowd issue of live streams by exploiting MDC with multiple dynamic distribution trees. It used one (or several) central server(s) that perform(s) a directory service for clients to obtain live streams from cooperative peers. CoopNet did not explicitly investigate issues for ensuring service quality. In this paper, we address this issue and propose efficient schemes for reducing server load and ensuring streaming service quality. Xu, et. al., also proposed a hybrid architecture [24] for on-demand service.

Handling node failures in P2P streaming: The existing P2P approaches like PROMISE [8], DagStream [14] address peer failures by switching to different peers to obtain missing data. They do not address the quality assurance issue during peer failures. PROMISE uses an elaborate parent selection process that takes into account an *inferred topology*, path loss rates and peer bandwidth. It detects peer failures based on either TCP connections or rate distortion, and then reselects new supporting peers. It uses forward error correction (FEC) to provide stream redundancy. A parent set is partitioned into active peers and standby peers. Parent nodes in the active peer set supply video data to a child node. Standby peers are used when one or more members in the active peer set fail. Although PROMISE provides a mechanism to prevent disruption of video sessions during peer failures, it does not explicitly control and guarantee video quality and stability.

We now discuss how we can extend PROMISE, using the techniques proposed in this paper, to further improve playback quality. In the proposed scheme, we do not explicitly separate active and standby peers. However, all the peers providing backup bandwidth can be considered as standby peers. PROMISE addresses the quality constraint by requiring that the aggregate bandwidth provided by all parents to be at least R_0 , the playback rate of the video. To enforce the reliability constraint, we can enhance the parent selection process in PROMISE with the proposed full stream construction algorithm. As a result, the parent selection is topology aware by virtue of Collectcast's [9] algorithms, and is also able to address both quality and reliability constraints by virtue of the full stream construction algorithm.

DagStream [14], on the other hand, ensures that each node has at least k parents to provide failure resilience and provable network connectivity. Thus both PROMISE and DagStream overprovision bandwidth (and paths), thereby providing redundancy to handle peer failures. The tree construction algorithms proposed in this paper also employ overprovisioning in the form of backup bandwidth. However, we use the knowledge of node reliability to limit the extent of overprovisioning and to also provide a statistical bound on stream failure probability. This contribution differentiates our work from existing schemes and is complementary to the contributions of solutions like PROMISE and DagStream.

7. CONCLUSIONS

In this work, we point out that ensuring playback quality and stability is a critical and challenging issue for P2P streaming. To address this issue, we propose a set of heuristic algorithms to build peer-based video delivery trees in local collaborative networks. The novel feature of our algorithms is that they use both upstream bandwidth contribution and peer reliability to build effective tree structures. Our simulations show that the proposed tree construction algorithms can efficiently deliver live video and significantly improve playback stability and quality.

REFERENCES

1. C. Aggarwal, J. Wolf, and P. Yu. On optimal batching policies for video-on-demand storage servers. In *Proc. of International Conference on Multimedia Systems'96*, 1996.
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, 2002.
3. S. Banerjee, S. Lee, R. Braud, S. Bhattacharjee, and A. Srinivasan. Scalable resilient media streaming. in *Proc. of NOSSDAV'04*, 2004.
4. Y. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. in *Proc. of ACM SIGMETRICS*, 2000.
5. H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peer-to-peer network. In *Technical Report*, Aug., 2001.
6. Y. Dong, E. Kusmierek, Z. Duan, and D. Du. A hybrid peer-assisted streaming architecture: Modeling and analysis. In *Proceedings of Internet Multimedia Systems and Applications*, 2004.
7. Y. Guo, Y. Suh, J. Kurose, and D. Towsley. P2cast: Peer-to-peer patching scheme for VoD service. in *Proc. of WWW 2003, May 20-24, Budapest, Hungary*, 2003.
8. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-peer media streaming using collectcast. *ACM Multimedia '03, Pages 45-54*, Nov. 2003.
9. M. Hefeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev. Collectcast: A peer-to-peer service for media streaming. In *ACM/Springer Multimedia Systems Journal*, October 2003.
10. A. Hu. Video-on-demand broadcasting protocols: a comprehensive study. In *Proc. of INFOCOM*, pages 508–517, 2001.
11. K. Hua, Y. Cai, and S. Sheu. Patching : A multicast technique for true video-on-demand services. In *Proc. of ACM Multimedia*, pages 191–200, 1998.
12. J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proc. of the International World Wide Web Conference*, pages 252–262, 2002.
13. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004.
14. J. Liang and K. Nahrstedt. DagStream: Locality Aware and Failure Resilient Peer-to-Peer Streaming. In *SPIE/ACM Multimedia Computing and Networking (MMCN'06)*, San Jose, CA, January 2006.
15. W. T. Ooi. Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment. In *Multimedia Computing And Networking*, 2005.
16. V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. in *Proc. of NOSSDAV'02, Miami, Florida*, 2002.
17. A. Raghuv eer, N. Kang, and D. H. Du. Techniques for efficient streaming of layered video in heterogeneous client environments. In *Proc. of IEEE Global Telecommunications Conference(GLOBECOM)*, 2005.
18. R. Rejaie and A. Ortega. Pals: Peer-to-peer adaptive layered streaming. in *Proc. of NOSSDAV'03*, 2003.
19. J. Sonnek, M. Nathan, A. Chandra, and J. Weissman. Reputation-based scheduling on unreliable distributed infrastructures. In *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2006.
20. T. Stading, P. Maniatis, and M. Baker. Peer-to-peer caching schemes to address flash crowds. In *Proc. of 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, USA, Mar. 2002.
21. D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. in *Proc. of IEEE INFOCOM*, Mar., 2003.
22. D. Wijesekera and J. Srivastava. Quality of service metrics for continuous media. *Multimedia Tools and Applications*, 3(2):127–166, 1996.
23. D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On peer-to-peer media streaming. In *Proc. of IEEE International Conference on Distributed Computing Systems*, 2002.
24. D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai. A CDN-P2P hybrid architecture for cost-effective streaming media distribution. *Computer Networks, Vol.44, Issue.3, pp.353-382*, 2004.
25. 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'05, Miami, FL, USA*, March 2005.