An Approximation Algorithm for Minimum-Delay Peer-to-Peer Streaming

Fei Huang*, Binoy Ravindran*, V.S. Anil Kumar†
*Dept. of Electrical & Computer Engineering
†Dept. of Computer Science and Virginia Bioinformatics Institute
Virginia Tech
Blacksburg, VA 24060, USA
huangf@vt.edu, binoy@vt.edu, akumar@vbi.vt.edu

Abstract

Peer-to-peer (P2P) technology provides a scalable solution in multimedia streaming. Many streaming applications, such as IPTV and video conferencing, have rigorous constraints on end-to-end delays. Obtaining assurances on meeting those delay constraints in dynamic and heterogenous network environments is a challenge. In this paper, we devise a streaming scheme which minimizes the maximum end-to-end streaming delay for a mesh-based overlay network paradigm. We first formulate the minimum-delay P2P streaming problem, called the MDPS problem, and prove its NP-completeness. We then present a polynomial-time approximation algorithm to this problem, and show that the performance of our algorithm is bounded by a ratio of $O(\sqrt{\log n})$. Our simulation study reveals the effectiveness of our algorithm, and shows a reasonable message overhead.

1. Introduction

In the recent decade, P2P networks have greatly enhanced content distribution on the Internet by enabling efficient cooperation among end users [1]. Benefiting from its significant scalability, there is an increasing demand on applications of P2P live streaming, such as IPTV, VOIP, and video conferencing [2]–[5]. In these classes of applications, the delivery of real-time video content imposes rigorous constraints on the end-to-end delay. Obtaining assurances on meeting such delay constraints in highly dynamic and heterogenous P2P network environments is a challenging and open problem. This has negatively affected the extensive commercial deployment of P2P systems. For example, IPTV deployment from commercial service providers is far below the industry expectation [1]. Motivated by this, in this paper, we focus on minimizing end-to-end streaming delay in P2P networks.

Minimizing streaming delays in P2P networks is an NP-complete problem. This is due to heterogeneous bandwidth requirements and network dynamics of P2P systems. Thus, obtaining optimal solutions to this problem for large-scale networks is intractable. In this paper, we propose an efficient approximation algorithm for this problem, which *provably* achieves a delay assurance by an approximation ratio of $O(\sqrt{\log n})$ where n represents the number of peers in the network. Based on an analytical model, we then design an adaptive distributed version of the algorithm, which can be easily deployed in a fully dynamic network environment.

Previous work on P2P streaming can be broadly classified into two classes: (1) multiple tree-based overlays, and (2) mesh-based overlays [6]–[11]. Recent studies have shown that the mesh-based approach consistently exhibits a superior performance over the tree-based approach [7], [12]. The tree-based P2P streaming approach organizes peers into multiple diverse trees. After obtaining the description of a Multiple Description Coded (MDC) content, it pushes each description through separate trees [6], [7]. In contrast, the mesh-based P2P streaming approach arranges peers into a randomly connected mesh and employs swarming content delivery [7]. The major advantages of mesh-based systems are easy maintenance and inherent robustness in high-churn P2P environments [10]. Motivated by these promising advantages, we study the minimum-delay problem under the mesh-based model. Although our algorithm is developed with a mesh paradigm, our study also indicates its readiness to fit the multiple tree-based model after simple modifications. To reduce the complexity of the problem, our paper focuses only on minimizing the communication delay. For packet scheduling, there exists a vast array of solutions, such as [11], [13], [14]. The mesh built from our algorithm can adopt any of these scheduling algorithms to yield low-delay streaming.

Existing heuristics on the problem of reducing P2P streaming delay either provides no theoretical bound on the worst-case performance or loosely estimate the bound without a robust theoretical analysis [8], [9], [15]. The estimated bound of previous algorithms [15], [16] is $O(\log n)$. In this paper, we not only present an approximation algorithm with a strong theoretical basis, but also reduce the approximation factor to a ratio of $O(\sqrt{\log n})$.

To the best of our knowledge, this paper represents the first approximation algorithm that optimizes P2P streaming delay and provides an upper bound. In summary, the paper makes three important contributions: (1) We present an efficient provable approximation algorithm for the minimum-

978-1-4244-5067-1/09/$26.00 ©2009 IEEE

delay P2P streaming problem with reasonable message overhead; (2) We analyze the algorithm's performance and derive the algorithm's approximation ratio, which is found to be the lowest ratio when compared with past results; and (3) We extend the algorithm to a practical distributed version that is robust to high user churn. Our simulation results indicate our algorithm can actively ensure the end-to-end streaming delay in the worst-case scenario.

The rest of this paper is organized as follows. Section 2 describes an overview of past and related works. In Section 3, we formulate the minimum-delay P2P streaming problem and prove its NP-completeness. Section 4 presents our proposed approximation algorithm and derive its performance. We compare our algorithm against past works through simulation-based experimental studies; these are reported in Section 5. Section 6 concludes the paper.

2. Related Work

Most of the previous protocols for the P2P streaming delay problem are based on tree-shaped overlays [5], [6], [8], [17]. Tran *et al.* propose Zigzag in [17], an approach to cluster peers into a hierarchy, called the administrative organization, for easy management, and build the multicast tree atop this hierarchy so as to reduce the delay. In [8], Noh *et al.* propose an overlay consisting of multiple trees with moderate out-degree to reduce end-to-end transmission delays in P2P media streaming systems. In [6], Venkataraman *et al.* present Chunkyspread, which splits a stream into distinct slices and transmits them over multiple trees by a P2P multicast algorithm. However, such multiple-tree overlay construction cannot be easily maintained [7]. In addition, it is difficult to obtain the globally optimal delay and coordinate among different trees [9]. Moreover, resource utilization of multiple-tree approaches is generally speaking, relatively low. For example, all leaf nodes do not contribute any bandwidth or CPU cycles to the multicast trees [18].

Recently, an increasing number of studies have focused on mesh-based P2P live streaming [9], [11], [16], [19]. In [9], Ren *et al.* propose a heuristic to reduce the delay on mesh topology, where peers select their parents based on the metric of link capacity divided by the communication delay. In this algorithm, peers located at the edge of mesh may only download the data without contributing its bandwidth resource, which may lead to low bandwidth utilization in the network. Thus, when the total uploading capacity is close to the downloading capacity among peer nodes, some peers may not be able to receive a live streaming. Besides, the heuristic does not provide performance guarantees on the end-to-end streaming delay, which is critical in delay-sensitive applications, such as video conferencing. In [19], Wu *et al.* present a distributed algorithm for obtaining the optimal average streaming delay. They apply several techniques in linear programming, such as Lagrangian relaxation and the subgradient algorithm. To reduce the computational complexity, they strictly limit the potential connections for each peer, which may restrict its practical applications. In their experimental results, we can observe significant time costs for achieving a near-optimal result. For a large-scale network, the convergence of the algorithm cannot be guaranteed, which may significantly increase the P2P start-up delay. Moreover, to exchange computational data between peers, considerable message overhead may be incurred in the network.

Minimum-delay P2P streaming has some similarity with the minimum-delay multicast tree problem (MDMT problem) [20], [21] and degree-bounded minimum-diameter tree problem (DBMDT problem) [22], [23]. Our algorithm is inspired by the clustering method first proposed by Könemann *et al.* [22]. However, previous approximation algorithms on MDMT and DBMDT generally assume a constant and equivalent degree on all the nodes and consider a single-commodity flow for each receiver. These assumptions are not appropriate for P2P streaming, where peers have heterogeneous and dynamic bandwidth capacities, i.e., heterogeneous degrees on the nodes, and they need to aggregate the multiple flows for a smooth playback. Toward that end, the clustering in our algorithm will generate a subgraph which is not simply a tree. Besides, the proofs on DBMDT theorems in [22] highly depend on the equality of node degree, which will cause the major proofs in their work do not hold in our case. Moreover, the DBMDT problem in [22] is bounded by a bidirectional degree, which does not distinguish out-degree and in-degree. For example, when calculating the aggregated cluster degree, their method, if used in our paradigm, will depend on both in and out degrees; however, only the out-degree should be counted in this scenario. All those differences make our problem more challenging than MDMT and DBMDT. In particular, DBMDT is a special case of the MDPS problem, in which every node has uniform outlink capacity and an inlink capacity of 1.

3. Problem Formulation

In this section, we formally state the minimum-delay P2P streaming problem (MDPS problem) and show that the problem is NP-complete.

3.1. Minimum-Delay P2P Streaming Problem

We model an overlay network as a directed graph $G = (V, E)$, where V is the set of vertices representing peer nodes, and E is the set of overlay edges representing directed overlay links. Let n represent the number of peers in the network, i.e. $n = |V|$. Each overlay link $(i, j) \in E$ is associated with a communication delay $d_{(i,j)}$. In the rest of this paper, we define the length of edge (i, j) as $d_{(i,j)}$, $\forall (i, j) \in E$. We assume that G is symmetric, i.e.,

$d_{(i,j)} = d_{(j,i)}, \forall i, j \in V$, and the delays associated with G form a metric, i.e., G satisfies the triangle inequality. For every peer $i \in V$, we define an upload capacity of O_i units/second and a download capacity of I_i units/second. For ease of presentation, we define $unit$ as the minimum package size in P2P streaming, which varies in different applications [4], [24].

We consider a peer-to-peer streaming session to originate from a single source node S to a set of receivers R, where $V = \{S\} \cup R$. Peers may receive the streaming data from the source node directly or indirectly from multiple P2P paths. Suppose S streams data at a constant streaming rate of s units/second. We denote f_{ij} as the rate at which peer i streams to peer j. If peer j receives the aggregated stream at s units/second from its parents, we call peer j as *fully served* [9]. Mathematically, the fully served requirement of peer j can be expressed as $\sum_{i:i\in L_j} f_{ij} = s$, where L_j is the set of parents of peer j. We assume that a fully served peer can smoothly play back the streaming content at its original rate of s units/second [9].

We call the stream from the source to one receiver j as the *P2P unicast flow* to j. A P2P unicast flow U may consist of streams from multiple P2P paths, called *fractional flows* [19]. Each fractional flow $p \in U$ has the arrival latency t_p from the source to receiver, where $t_p = \sum_{(i,j)\in p} d_{(i,j)}$. The latency of the unicast flow U can be defined as the maximum latency among its fractional flows, i.e., $\max_{p\in U} t_p$. To stream multimedia content to multiple receivers, we can envision multiple unicast flows from the source to receivers. Thus, the maximum delay in P2P streaming is defined as the maximum latency of all unicast flows.

We now define the problem formally:

Definition 1: **Minimum-Delay P2P Streaming Problem (MDPS problem):** Given the constraints that are previously mentioned, the MDPS problem is to devise a streaming scheme which minimizes the maximum end-to-end streaming delay with each receiver fully served.

To help obtain greater insights about the MDPS problem, we formulate the problem in the integer linear programming framework, as follows:

$$\min\ t \tag{1}$$

subject to

$$\sum_{(i,j)} d_{(i,j)} x^r_{ijm} \le t, \forall (i,j) \in E, \forall r \in R, \forall m \tag{2}$$

$$x^r_{ijm} \in \{0,1\}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{3}$$

$$x^r_{ijm} \ge f^r_{ijm}/s, \forall (i,j) \in E, \forall r \in R, \forall m \tag{4}$$

$$s \ge f^r_{ijm} \ge 0, \forall (i,j) \in E, \forall r \in R, \forall m \tag{5}$$

$$f^r_{ijm} - f^r_{jim} = b^r_{im}, \forall (i,j) \in E, \forall r \in R, \forall m \tag{6}$$

$$\sum_{m=1}^{s} \sum_{r:r\in R} f^r_{ijm} \le y_{ij}, \forall (i,j) \in E \tag{7}$$

$$\sum_{j:(i,j)\in E} y_{ij} \le O_i, \forall i \in V \tag{8}$$

$$\sum_{j:(j,i)\in E} y_{ji} \le I_i, \forall i \in V \tag{9}$$

$$0 < m \le s, \tag{10}$$

where

$$b^r_{im} \begin{cases} \ge 0 & \text{if } i = S, \\ \le 0 & \text{if } i = r, \\ = 0 & \text{otherwise,} \end{cases} \tag{11}$$

$$\sum_m b^r_{im} = \begin{cases} s & \text{if } i = S, \\ -s & \text{if } i = r, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

In this integer programming (IP) expression, t denotes the streaming delay of the fractional flow; x^r_{ijm} is set to 1 only if there is a connection between peer i and j on the m^{th} fractional flow to receiver r; f^r_{ijm} represents the m^{th} fractional flow rate on link (i,j) to receiver r; and y_{ij} is the aggregated flow rate on link (i,j).

The exact solution to this problem is optimal, but is computationally intractable to determine, and not practical in real applications. We will now show the NP-completeness of this problem, which motivates us to develop a near-optimal approximation algorithm.

Lemma 1: If the instance of MDPS problem has a solution, then the sum of the upload capacities, including source and receivers, must be no less than the sum of fully served streaming rates at all receivers, i.e.,

$$\sum_{i\in V} O_i \ge (|V| - 1) \times s. \tag{13}$$

Proof: Suppose we have a feasible streaming scheme described by the graph $A = (V, E_f)$, where $E_f \subset E$ represents the P2P connections among V. We can envision that each peer $v \in V$ consists of two conceptual nodes v_{in} and v_{out}, where v_{in} represents the download behavior, and v_{out} represents the upload behavior. Thus, A can be envisioned as a bipartite graph $A' = (V_{in}, V_{out}, E_f)$, where V_{in} is the set of all v_{in} and V_{out} is the set of all v_{out}. Now, the flow out of V_{out} should be equal to the flow into V_{in}, i.e., $(|V| - 1) \times s$. Since the flow out of V_{out} cannot exceed its total upload capacities, we have $\sum_{i\in V} O_i \ge (|V| - 1) \times s$. The lemma follows. □

According to Lemma 1, it is reasonable to assume that the preliminary condition in Equation (13) holds. In addition, we presume that the download capacity $I_i \ge s, \forall i \in V$ for a smooth playback at receivers.

3.2. Hardness

Theorem 1: The minimum-delay P2P streaming problem is NP-complete.

Proof: We first show that the MDPS problem can be reduced from the Freeze-Tag Problem (FTP), which is well known to be an NP-hard problem [25], [26]. Given a complete graph $G' = (V', E')$ in metric space, FTP can be described by a set of robots V', among which one robot S' is initially awake and all the others R' are asleep. The robot which is awake will select B' sleeping robots and awaken them. Once awaken, each new robot is available to assist in rousing another set of B' robots. There exists a latency between each pair of robots. Thus, a solution to the FTP can be described by a wake-up tree T which is a directed B'-ary tree rooted at S', spanning all robots R'. The objective is to minimize the makespan of T, i.e., the time t' when the last robot awakens [25].

Let the formulation of G in MDPS be identical to G' in FTP, and let the streaming rate be $s = 1$ units/second. For all nodes V in G, we set their upload capacities to be B' units/second. In this case, every node will have exactly one parent and can stream up to B' children. As we can observe, the resulting topology becomes a B'-ary spanning tree T, and the maximum end-to-end streaming delay of t is the latency of the last peer in T. Therefore, the optimal solution to the MDPS problem can also solve the FTP problem optimally. In other words, the FTP will have a tree T in G' with minimum delay of $\min t'$ if and only if the same T in G provides the minimum-delay spanning tree and the resulting min-max delay of MDPS equals $\min t'$. Since FTP can be reduced to the MDPS problem, the MDPS problem is NP-hard. Also, we can clearly see MDPS is in NP since it is easy to check whether a streaming scheme has a delay of t and follows the streaming constraints listed in Equations (8), (9), (12). Thus, we complete the proof. □

4. Approximation Algorithm

4.1. Overview of Techniques

Given the NP-complete nature of the MDPS problem, our goal is to design an efficient polynomial-time approximate solution with a provable performance bound. Our work builds on by Könemann *et al.* [22] and we extend a number of their concepts, including *clustering* and *filtering*. First, we partition the peers V into different clusters according to the regional aggregated streaming capacities. Then, we filter the peers by keeping one representative peer for each cluster, whose streaming capacity is virtualized by the aggregated streaming capacities of the entire cluster. As a result, we form a backbone using the representative nodes. Since the backbone nodes are virtual representatives of clusters, in the next step, we expand the mesh connections from the representative nodes into clusters, which constitutes a final streaming mesh for the overlay network.

To simplify the complexity of the problem, we assume that at least half of the nodes have upload capacities $O_i \geq 2s$ units/second. Besides, we assume there exists no free-riders, i.e., $\min(O_i) > 1$ unit/second, $\forall i \in V$. For the case of $O_i = 1$ unit/second, this reduces the problem to the traditional traveling salesman path problem (TSP), which has been extensively studied in the past [27], [28]. Therefore, we will not focus on this scenario in the paper.

In the rest of this section, we discuss the details of our algorithm and derive its performance bound.

4.2. Centralized Approximation Algorithm

4.2.1. Streaming Backbone Construction. The first step of our algorithm is to construct the virtual streaming backbone. We call it "virtual", because the links on the backbone represent the aggregated inter-cluster streams instead of the actual flows to the representative nodes. Towards that end, we define a metric $C(V')$, called the *residual streaming capacity* for a group of peers $V' \subseteq V$ as:

$$C(V') = \sum_{i \in V'} O_i - (|V'| - 1) \times s. \tag{14}$$

The residual streaming capacity represents the contributable bandwidth that a group of peers can supply other groups after satisfying its own streaming demands. Next, we define a threshold γ. The set of peers that are enclosed within the γ radius from peer $v_i \in V$ is denoted as $E_\gamma(v_i, V)$. In addition, we say that v_i γ-covers the peers in $E_\gamma(v_i, V)$ [22].

A parameter t' is chosen, which can be viewed as a "guess" on the optimal P2P streaming delay. A reasonable value of t' should be initialized in the range of $[\max_{v \in V} d_{(S,v)}, |R| \cdot \max_{v \in V} d_{(S,v)}]$. For the given value of t', we set $\gamma = t'/\sqrt{\log n}$. We begin the clustering process from the source node by defining the first cluster $U_1 = E_{3\gamma}(S, W_1^{3\gamma})$ and the first representative node as $u_1 = S$, where $W_1^{3\gamma} = V$, represents the initially unclustered nodes. For ease of notation, we call the set of peers that are at least γ distance away from existing $i-1$ representatives $u_1, \cdots, u_{i-1}$ as W_i^γ, where $W_i^\gamma = V \setminus \bigcup_{1 \leq j \leq i-1} E_\gamma(u_j, V)$. We then select a representative node $u_i \in W_i^\gamma$ that γ-covers the peers with the highest residual streaming capacity in $W_i^{3\gamma}$, i.e.:

$$u_i = \text{argmax}_{v \in W_i^\gamma} C(E_\gamma(v, W_i^{3\gamma})). \tag{15}$$

Now, we have $U_i = E_{3\gamma}(u_i, W_i^{3\gamma}) \cup E_\gamma(u_i, W_i^\gamma)$. The iteration stops once all the peers in V are 3γ-covered by the existing representatives. Suppose we have k clusters. Then, $W_{k+1}^{3\gamma} = \emptyset$ and $W_i^{3\gamma} \neq \emptyset, \forall 1 \leq i \leq k$.

Without loss of generality, we reorder the clusters $U_1, \cdots, U_k$ so that the residual streaming capacities of the clusters except that of U_1 are sorted in a non-increasing

order, i.e., $C(U_i) \geq C(U_j), \forall 1 < i < j \leq k$. By virtualizing the upload capacity of the representative u_i as $C(U_i)$, we can construct the global streaming mesh of low latency for the backbone representatives. Algorithm 1 describes this procedure. It will be rerun with different values of t' chosen by binary search to achieve the approximated minimum delay.

Corollary 1: Any peer that is 2γ-covered by u_i must be in the cluster of U_j with $1 \leq j \leq i \leq k$.

Algorithm 1 APX-MDPS($G, n, s, t', \{O_i\}, \{I_i\}$): Centralized approximation algorithm APX-MDPS for the MDPS problem

$\gamma = t'/\sqrt{\log n}$
$W_1^{3\gamma} = V$
$U_1 = E_{3\gamma}(S, W_1^{3\gamma})$
$u_1 = S$
$i = 2$
while $W_i^{3\gamma} \neq 0$ **do**
  $u_i = \text{argmax}_{v \in W_i^{\gamma}} C(E_{\gamma}(v, W_i^{3\gamma}))$
  $U_i = E_{3\gamma}(u_i, W_i^{3\gamma}) \cup E_{\gamma}(u_i, W_i^{\gamma})$
  $i = i + 1$
end while
Reorder $U_2, \cdots, U_k$ so that $C(U_i) \geq C(U_j), \forall 2 \leq i < j \leq k$
Construct the backbone mesh by Algorithm 2
Construct the regional mesh by Algorithm 3
Construct the final mesh by Algorithm 4

Algorithm 2 APX-BACKBONE($\{U_i\}, \{u_i\}$): Backbone mesh construction algorithm for the MDPS problem

for $i = 2$ to k **do**
  repeat
    $j = \min_{1 \leq n \leq k}\{n : U_n$ has remaining bandwidth$\}$
    Connect u_i to u_j
  until U_i is fully served
end for

4.2.2. Regional Streaming Mesh Construction. In this section, we show the steps of creating the regional streaming topology for each cluster. Given a set of clusters, we can identify two types among them by measuring $C(U_i) \geq 0$ or $C(U_i) < 0$. For the cluster with non-negative residual streaming capacities, a mesh spanning the peers of U_i can be constructed by Algorithm 3. For the other type of cluster with negative residual streaming capacities, we can deduce from Lemma 1 that the upload capacities $\sum_{v \in U_i} O_v$ inside the cluster cannot satisfy its internal streaming requirement $(|U_i| - 1) \times s$ and thus need extra streaming connections from external clusters. In such clusters, we will first satisfy the internal peers with the highest upload capacities so that they can timely serve other internal peers. In that way, only the peers with the lowest upload capacities will be left for external connections. Algorithm 3 also describes the method to construct the streaming topology for such clusters.

Algorithm 3 APX-CLUSTER($\{U_i\}, \{u_i\}$): Regional mesh construction algorithm for the MDPS problem

for $i = 1$ to k **do**
  for $j = 1$ to $|U_i|$ **do**
    repeat
      if j is the peer with largest uploading capacity **then**
        Hold j for an uplink connection from other cluster
      else
        $x = \text{argmax}_{v \in U_i \ \& \ v \text{ has remaining bandwidth}} C(\{v\})$
        if x is fully served **then**
          Connect j to peer x
        else
          Hold j for external connection
        end if
      end if
    until peer j is fully served or on hold
  end for
end for

4.2.3. Complete Streaming Mesh Construction. To complete the final mesh construction, we replace virtual links between representatives to real inter-cluster connections by Algorithm 4. We then do a binary search over t' to obtain the minimum-delay streaming topology. An outline of the algorithm is described in Algorithm 4. Figure 1 illustrates a cluster-based streaming mesh from the source to receivers.

Algorithm 4 APX-COMPLETE($G, \{U_i\}, \{u_i\}$): Complete mesh construction algorithm for the MDPS problem

for $i = 1$ to k **do**
  Connect each peer that is held for external connections to the closest available peer in the virtually linked parent cluster.
end for

4.2.4. Performance Bound. Assume t' is the optimum value. After binary search, it can be approximated within a factor of 2. We now analyze the performance bound of the approximation algorithm, i.e., the approximation factor.

We start from a simple scenario with a streaming rate of $s = 1$ unit/second. In this case, the resulting streaming topology can be expected as a tree structure denoted as T, because each peer will only receive stream from one parent. Let T^* be the optimal tree with the minimum streaming delay, denoted by $T^* = (V, E^*)$ and $E^* \subset E$. Now we partition T^* into clusters $\{U_1^*, \cdots, U_q^*\}$, which

are represented by $\{u_1^*, \cdots, u_q^*\}$, respectively. We define two functions: PATH(i, j), which returns true only if there exists a directed path from i to j in T^* denoted as $\langle i, j \rangle$, and HEIGHT(i), which returns the height of node i in tree T^* rooted at S. Let T_B^* be the backbone after partitioning T^*.

Our method to partition T^* is summarized as follows:

1) Let $u_1^* = S$ and U_1^* be the set of descendants that is γ-covered by S, i.e., $U_1^* = \{u : u \in V$, PATH$(S, u) =$ TRUE, and $d_{(S,u)} \leq \gamma\}$.
2) Let W^* be the uncovered peers where $W^* = V \setminus U_1^*$;
3) Select the lowest uncovered node as the next representative u_i^*, i.e., $u_i^* = \text{argmin}_{u \in W^*}$HEIGHT$(u)$ and $U_i^* = \{u : u \in W$ and PATH$(u_i^*, u) =$ TRUE and $d_{(S,u)} \leq \gamma\}$;
4) Remove U_i^* from W^*, i.e., $W^* = W^* \setminus U_i^*$;
5) Repeat from Step 3 until all the nodes are covered.

Suppose we have q clusters from T^* after the above steps. Without loss of generality, we then reorder the clusters $U_1^*, \cdots, U_q^*$ so that the residual streaming capacities of them except U_1 are sorted in a non-increasing order, i.e., $C(U_i^*) \geq C(U_j^*), \forall 1 < i < j \leq q$. Moreover, we denote T_B as the backbone constructed from G by Algorithm 1.

Lemma 2: Suppose U_i^* and U_j intersects, i.e., $U_i^* \cap U_j \neq \emptyset$. Then, there must exist a cluster U_m covering at least one $u \in U_i^* \setminus U_j$ and satisfying $C(U_m) \geq C(U_i^*)$, where $1 \leq m \leq k$.

Proof: If any peer w in $U_i^* \cap U_j$ is within the γ radius of u_j, i.e., $d_{(w,u_j)} \leq \gamma$, then every peer $u \in U_i^* \setminus U_j$ will be 2γ-covered by u_j. According to Corollary 1, there must exist some cluster U_m covering u , where $1 \leq m \leq j$. Since u_j γ-covers the peers in $U_i^* \cap U_j$, it is easy to see that $C(U_j) \geq C(U_i^*)$. Then we have $C(U_m) \geq C(U_j) \geq C(U_i^*)$.

If $\gamma \leq d_{(w,u_j)} \leq 2\gamma$, then there must exist a cluster U_m, which is the first cluster that covers at least one peer $\in U_i^*$ with $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* will form a cluster itself. Peer $u \in U_i^* \setminus U_j$ may have two possibilities: (1) $d_{(u,u_j)} \leq 2\gamma$, or (2) $2\gamma < d_{(u,u_j)} \leq 3\gamma$. In the case of $d_{(u,u_j)} \leq 2\gamma$, there must exist some cluster U_n covering u with $1 \leq n < j$ according to Corollary 1. Since U_m should have $C(U_m) \geq C(U_n) > C(U_j)$, we deduce $U_m \neq U_j$. In the other case of $2\gamma < d_{(u,u_j)} \leq 3\gamma$, the cluster U_m that covers u must be within the γ radius of u and $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* itself will be more qualified as a cluster than U_m.

If $d_{(w,u_j)} > 2\gamma$, then peer $u \in U_i^* \setminus U_j$ must be within the γ radius of some u_m and $C(U_m) \geq C(U_i^*)$; otherwise, U_i^* will form a cluster itself. Thus, in all cases, the lemma follows. □

Lemma 3: Compare the residual capacities of T_B and T_B^*.

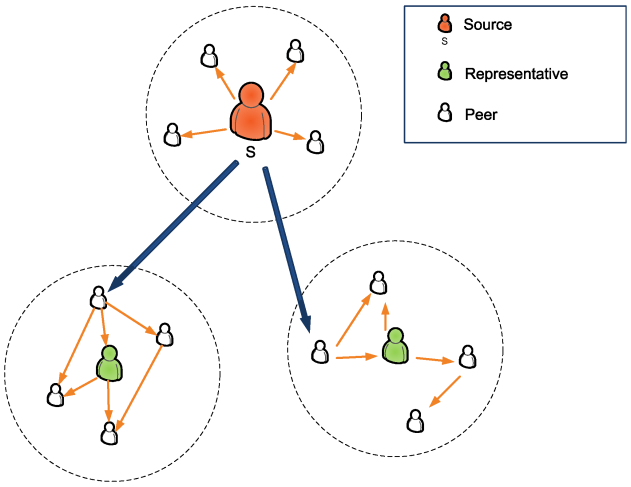

Figure 1: Cluster-based streaming mesh.

We have:

$$\sum_{i=1}^{j} C(U_i^*) \leq \sum_{i=1}^{j} C(U_i), \forall 1 \leq j \leq q. \tag{16}$$

Proof: We use induction to prove this claim. For $j = 1$, it is obvious that $C(U_1^*) \leq C(U_1)$. Now, we assume that the claim also holds for $j = x$. When $j = x+1$, we have a new cluster U_{x+1}^*. If $\bigcup_{1 \leq i \leq x} U_i$ contains no peer in U_{x+1}^*, i.e., $U_{x+1}^* \cap \bigcup_{1 \leq i \leq x} U_i = \emptyset$, then we should have $C(U_{x+1}) \geq C(U_{x+1}^*)$ because Algorithm 1 always selects U_{x+1} with the largest residual streaming capacity from the uncovered peers. If $\bigcup_{1 \leq i \leq x} U_i$ contains several peers in U_{x+1}^*, then we have $U_{x+1}^* \cap \bigcup_{1 \leq i \leq x} U_i \neq \emptyset$.

For the sake of contradiction, we assume that $\sum_{i=1}^{x+1} C(U_i^*) > \sum_{i=1}^{x+1} C(U_i)$. If there is any cluster U_n^* with $n \leq x$ satisfying $U_n^* \cap \bigcup_{1 \leq i \leq x} U_i = \emptyset$, then $C(U_{x+1}) \geq C(U_n^*) \geq C(U_{x+1}^*)$. Thus, to make the assumption correct, we have $U_n^* \cap \bigcup_{1 \leq i \leq x} U_i \neq \emptyset, \forall 1 \leq n \leq x$. Therefore, according to Lemma 2, we must have a U_m covering at least one peer in $U_n^* \setminus \bigcup_{1 \leq i \leq x} U_i$ and satisfying $C(U_m) \geq C(U_n^*)$ with $x < m \leq k$. Consequently, we should have $C(U_{x+1}) \geq C(U_m) \geq C(U_n^*) \geq C(U_{x+1}^*)$. Then, $\sum_{i=1}^{x+1} C(U_i^*) \leq \sum_{i=1}^{x+1} C(U_i)$, which contradicts the assumption. The lemma follows. □

Since the sum of the residual capacities is bounded by $\sum_{i \in V} O_i - (n-1) \times s$, we can easily deduce Corollary 2 from Lemma 3:

Corollary 2: The number of clusters in T is less than that in T^*, i.e., $k \leq q$.

Lemma 4: Let H_I denote the height of a tree I. We have $H_{T_B} \leq H_{T_B^*}$.

Proof: T_B is constructed as a balanced tree. Combining this fact with Lemma 3, which represents a higher out-degree in T_B than that in T_B^*, we can observe that $H_{T_B} \leq H_{T_B^*}$. □

We call the edge connecting different clusters as the *backbone edge* and the edge inside the cluster as the *cluster edge*.

Lemma 5: Any root-to-leaf path in T_B has at most $O(\sqrt{\log n})$ backbone edges.

Proof: It follows from Lemma 4 that $H_{T_B} \leq H_{T_B^*}$. From the construction method of T_B^*, we know that any root-to-leaf path in T_B^* has a length, i.e., latency, which is at least $\gamma \cdot H_{T_B^*}$. Since T_B^* has a latency which is at most OPT, it follows that:

$$H_{T_B} \leq H_{T_B^*} \leq \frac{\text{OPT}}{\gamma} = \frac{\text{OPT}}{t'/\sqrt{\log n}} = O(\sqrt{\log n}). \quad (17)$$

□

Lemma 6: Any root-to-leaf path in T has at most $O(\log n)$ cluster edges.

Proof: Without loss of generality, we can envision T_B is constructed in a breadth-first and left-to-right order, which means representatives on the same height are arranged from left to right in a non-increasing order of their residual streaming capacities. In another word, if u_i is on the left of u_j on the same height in T_B where $0 \leq i, j \leq k$, we have $C(U_i) \geq C(U_j)$. Moreover, if u_i has a lower height in T_B than u_j where $0 \leq i, j \leq k$, we have $C(U_i) \geq C(U_j)$.

Denote $T(U)$ as the tree that is constructed inside the cluster U. Let Y be the rightmost root-to-leaf path in T_B, denoted as $Y = \langle u_1, u_{y_1}, \cdots, u_{y_b} \rangle$, i.e., Y is formed from root to leaf by representatives $u_1, u_{y_1}, \cdots, u_{y_b}$. For any root-to-leaf path X, where $X = \langle u_1, u_{x_1}, \cdots, u_{x_a} \rangle$, we denote the number of cluster edges in X as ζ_X. Since T_B is constructed as a balanced tree, we can deduct $b \leq a \leq b+1$.

Because Y is the rightmost root-to-leaf path in T_B, it follows that

$$\begin{aligned}\zeta_X &= H_{U_1} + \sum_{i=1}^{a} H_{T(U_{x_i})} \\ &= H_{U_1} + H_{U_{x_1}} + \sum_{i=2}^{a} H_{T(U_{x_i})} \\ &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} H_{T(U_{y_i})}.\end{aligned}$$

Recall the previous assumption that $\min(O_i) \geq 2, \forall i \in V$. For the case of $b \leq 1$, it is easy to deduct $\zeta_X \leq 3\log_2 n = O(\log n)$. Otherwise, we can carry out

$$\begin{aligned}\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) - 1\right) \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} \left(C\left(U_{y_i}\right) - 1\right) \right). \quad (18)\end{aligned}$$

In addition, the number of the clusters, i.e. k, is bounded by n. Thus, we can deduct

$$\begin{aligned}n \geq k \geq & 1 + C\left(U_1\right) + C\left(U_1\right) \cdot C\left(U_{y_1}\right) + \\ & \cdots + C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \\ & = 1 + C\left(U_1\right) \cdot \left(1 + \sum_{i=1}^{b-1} \prod_{j=1}^{i} C\left(U_{y_i}\right) \right).\end{aligned}$$

Thus, we have

$$C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \leq n. \quad (19)$$

Replacing Equation (19) into (18), we have

$$\begin{aligned}\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(C\left(U_1\right) \cdot \prod_{i=1}^{b-1} C\left(U_{y_i}\right) \right) \\ &\quad + \log_2 \left(C\left(U_{y_b}\right) / C\left(U_1\right) \right) \\ &\leq H_{U_1} + H_{U_{x_1}} + \log_2 n + \log_2 \left(C\left(U_{y_b}\right) / C\left(U_1\right) \right) \\ &\leq 4\log_2 n \\ &= O(\log n). \quad (20)\end{aligned}$$

Thus, the lemma follows. □

Theorem 2: Let OPT be the minimum P2P streaming delay from the source host S to receivers R in T. The streaming delay of the solution produced by Algorithm APX-MDPS is at most $O(\sqrt{\log n}) \cdot \text{OPT}$.

Proof: It follows from Lemma 5 that any root-to-leaf path has at most a latency of $O(\sqrt{\log n}) \cdot \text{OPT}$ arising from the backbone edges on the path. In addition, short edges in T have a latency that is no more than $6\gamma = 6t'/\sqrt{\log n}$. From Lemma 6, we know that any root-to-leaf path has at most a latency of $6\gamma \cdot O(\log n) = O(\sqrt{\log n}) \cdot \text{OPT}$ caused by cluster edges on the path. T is constructed from the backbone edges and the cluster edges. The theorem follows. □

Now, let us look at the problem when $s > 1$ units/second.

Theorem 3: Let OPT be the minimum P2P streaming delay from the source host S to receivers R in G. The streaming delay of the solution that Algorithm APX-MDPS returns is at most $O(\sqrt{\log n}) \cdot \text{OPT}$.

Proof: When $s > 1$ unit/second, the final streaming topology will be a mesh, which can be envisioned as a combination of multiple trees constructed by fractional streams. To prove the previous bound also holds here, we first normalize all flow rates and capacities by s. Then, the correctness of Lemmas 1-5 is obvious. For Lemma 6, we start justifying its correctness from Equation (18). Recall the assumption that at least half of the nodes have $O_i \geq 2s$. Because of the balanced topology in connection, we can

prove that any node whose $O_i < 2s$ will always lay on the bottom in its cluster. (Due to space limitations, we do not provide detailed proof of this claim.) As a result, we can carry out

$$\begin{aligned}\zeta_X &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} - 1 \right) \\ &\leq H_{U_1} + H_{U_{x_1}} + \sum_{i=1}^{b} \log_2 \left(C\left(U_{y_i}\right) \times \frac{2s}{\min(O_i)} \right) \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + b \cdot \log_2 \frac{2s}{\min(O_i)} \\ &= H_{U_1} + H_{U_{x_1}} + \log_2 \left(\prod_{i=1}^{b} C\left(U_{y_i}\right) \right) + \log_2 n \times \log_2 s. \end{aligned} \tag{21}$$

Similar to the deduction of Equation (20), we have

$$\begin{aligned}\zeta_X &\leq (4 + \log_2 s) \cdot \log_2 n \\ &= O(\log n). \end{aligned} \tag{22}$$

Thus, the theorem follows.

□

4.3. Distributed Algorithm

The centralized algorithm described in Section 4.2 approximately solves the minimum-delay P2P streaming problem. In a practical setting, however, we may not have a central server that can provide a global computation resource which is implicitly assumed by the algorithm. Thus, to increase the scalability and reliability, we extend our algorithm to a distributed version, which can be well adopted and improved as a practical P2P protocol. For the ease of presentation, we assume that all the representatives have sufficient resources to coordinate the peers. In the actual deployment, we define a *cluster leader* to take charge, which is the peer with the most computing and bandwidth resources in the cluster.

4.3.1. Peer Join. In order to join the clusters, a newly arrived peer i will first contact a *rendezvous point* (RP), which caches a list of existing representative peers. The rendezvous point then sends back a random list of representatives that are approximately nearby the newcomer and the updated parameter of γ. Peer i will then check the latency and the residual streaming capacity with each representative u_j. If there exists an u_j within γ latency of i, peer i will send a request to join the closest cluster; otherwise, peer i will tentatively join the closest cluster, but meanwhile measure the feasibility to build a new cluster.

In the attempt to organize a new cluster, peer i will contact a set of nearby representatives and will retrieve a list of peers that are between γ and 3γ-away from each representative u_j. Next, peer i will pick the peers within its γ radius and activate the new clustering process on these peers. In the process, they start exchanging the latency and capacity information with each other and find the center peer, i.e., the representative, which has the maximum residual streaming capacities for the attempted new cluster. If the residual streaming capacity of the candidate cluster is less than the existing cluster within 3γ radius of j, the new clustering process will be terminated; otherwise, the new representative will request peers within its γ latency to join the new cluster.

Finally, the new representative will request peers within its γ latency to join the new cluster. After a peer joins a cluster, its representative will allocate the parents and children for it. Once the new peer receives this information, it will initiate the stream with those parents and children directly.

4.3.2. Peer Departure. The departures and failures of peers may lead to interrupted playback at the remaining receivers. If any peer departs from the cluster, it will inform its representative and request re-allocating the bandwidth for its downstream peers. To handle this problem of failure, each peer will buffer for a short period when streaming and always keep an eye for the back-up peers during streaming. If failure does happen, the downstream peer will utilize those time of buffering to connect to the backup peers tentatively and then request a stream re-allocation to the representative. If it is the representative that fails or leaves, the affected peers will use the buffering time to activate a new round of clustering within γ radius.

4.3.3. Stream Coordination and Dynamic Adaptation. The topology maintenance and stream allocation are mostly coordinated by the representatives. They will assign a new peer to connect to the peer with the most capacity in its list. If the residual streaming capacity is non-negative, they will always maintain an intra-cluster streaming by utilizing the idle bandwidth. If the residual streaming capacity is negative, they will coordinate with the rendezvous point and inform the unserved peers to stream from the cluster with the most residual capacity. When a new cluster is established, the same steps are followed to import a complete stream to the cluster. If the γ needs to be tuned or optimized, the clustering process will be initiated by the rendezvous point and run in the background without interrupting the existing streams. All connections are updated until the background computation is completed.

In a distributed environment, peers may join and leave randomly. Thus, peers with high bandwidth resources may arrive late and as a result, connect far from the source. Cluster representative and rendezvous point will be responsible

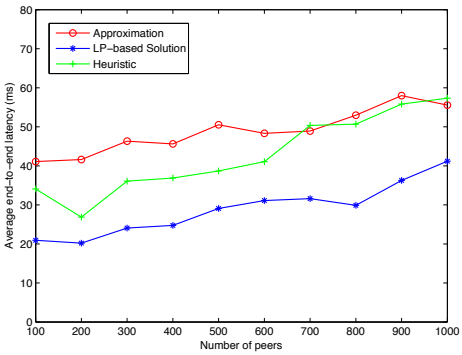

Figure 2: Average end-to-end latency

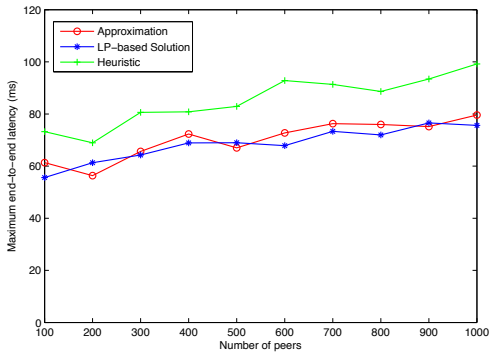

Figure 3: Maximum end-to-end latency

for monitoring this scenario at the cluster level and backbone level, respectively. Once they detect this scenario, they initiate new rounds of stream allocation in the background and update the connections until they reserve the bandwidth.

5. Simulation Study

In this section, we describe the results of our simulation study. We simulate a live streaming session of 300 Kbps from a source with 10 Mbps upload capacity. From previous studies, we know that network bandwidth exhibits rich diversity [9], [24]. Based on this, we set the upload capacity among peers as shown in Table 1. In this simulation study, we compare our algorithm with two other recently proposed algorithms: a heuristic approach [9] and a LP-based approach [19]. The heuristic in [9] is the first algorithm that focuses on reducing the maximum end-to-end delay on mesh streaming, where peers select their parents based on the metric of link capacity divided by the communication delay [9]. The LP-based approach in [19] applies several linear programming techniques to obtain an optimal average delay, such as Lagrangian relaxation and subgradient algorithm. Please note, to reduce computational costs, [19] restricts the potential connections for each peer. This actually lowers down the performance compared with real LP-based solution. However, for the easy of presentation, we still call it LP-based solution in the rest of this paper.

To evaluate the algorithm performance, we define four metrics, including *average end-to-end delay*, *maximum end-to-end delay*, and *message overhead*.

The average end-to-end delay is defined as the average

Upload Capacity	Percentage of Peers
200 Kbps	30%
1.0 Mbps	50%
2.0 Mbps	15%
10.0 Mbps	5%

Table 1: Upload Capacity Distribution

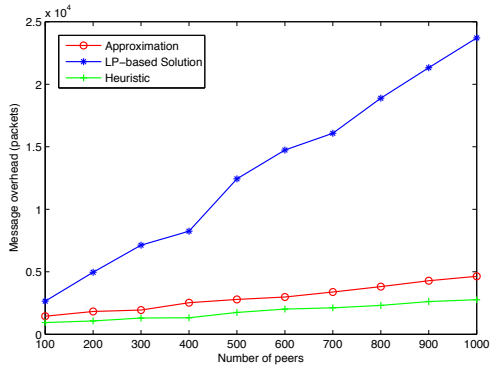

Figure 4: Message overhead

latency from the source to all receivers. Figure 2 illustrates the results from our simulation experiments. It shows that the LP-based approach generally achieves a lower average delay than the other two approaches. This is reasonable, since the LP-based approach is designed to minimize the average delay. It is also interesting to observe that the heuristic algorithm exhibits lower average delay than the approximation algorithm when the network size is relatively small. When the network size approaches 700 nodes, the approximation algorithm yields an average latency that is close to that of the heuristic approach. This indicates that the approximation algorithm has a smaller growth rate with respect to increase in network size, implying that the algorithm is scalable for large network sizes.

We also measure the maximum delay, which is the worst-case end-to-end delay observed in the simulation experiments. Note that this is our primary design objective. Figure 3 shows the maximum delay of the algorithms. It is apparent that the worst-case performance of our algorithm is close to that of the LP-based solution and outperforms the heuristic. This low worst-case delay indicates that our algorithm ensures good streaming performance with an approximation bound.

Figure 4 shows the message overhead of the algorithms,

measured in number of packets, during mesh construction and maintenance. Although a minimum delay is desirable, a large message overhead will challenge practical deployment of the underlying algorithm. As we can observe, the LP-based solution generates a huge number of overhead packets during mesh construction. In addition, its overhead significantly increases with the network size. This is mainly resulting from the computational message exchange. In contrast, the heuristic and the approximation algorithm both have much less message overhead and slow growth rate, as the network size increases. The major overhead of our algorithm occurs in clustering. From the simulation results, we observe that this overhead is slightly higher than that of the heuristic algorithm.

Thus, our simulation results reveal the effectiveness of our algorithm, in terms of ensuring a worse-case delay with high scalability.

6. Conclusion

In this paper, we focus on building delay-minimized overlay streaming mesh. We formulated the minimum-delay P2P streaming problem and presented two solutions for it: a centralized approximation algorithm and a distributed version. We show that our algorithms have a guaranteed performance bound. The distributed version has been extended to adopt to network churn and improve resource utilization.

Acknowledgment

The authors were supported partially by NSF Nets Grant CNS-0626964, NSF HSD Grant SES-0729441, CDC Center of Excellence in Public Health Informatics Grant 2506055-01, NIH-NIGMS MIDAS project 5 U01 GM070694-05, DTRA CNIMS Grant HDTRA1-07-C-0113, NSF NETS CNS-0831633, and CNS-0845700.

References

[1] A. Sentinelli, G. Marfia, M. Gerla, L. Kleinrock, and S. Tewari, "Will IPTV ride the peer-to-peer stream?" *Communications Magazine, IEEE*, June 2007.

[2] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen, "FLoD: A framework for peer-to-peer 3D streaming," in *INFOCOM 2008*, 2008.

[3] F. Picconi and L. Massoulie, "Is there a future for mesh-based live video streaming?" in *P2P '08*, 2008.

[4] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale p2p IPTV system," *Multimedia, IEEE Transactions on*, Dec. 2007.

[5] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *INFOCOM 2006*, 2006.

[6] V. Venkataraman, P. Francisy, and J. Calandrino, "Chunkyspread: Multitree unstructured peer-to-peer multicast," in *IPTPS06*, 2006.

[7] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," in *INFOCOM 2007*, 2007.

[8] J. Noh, A. Mavlankar, P. Baccichet, and B. Girod, "Reducing end-to-end transmission delay in p2p streaming systems using multiple trees with moderate outdegree," in *IEEE Multimedia and Expo '08*, 2008.

[9] D. Ren, Y.-T. Li, and S.-H. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *INFOCOM 2008*, 2008.

[10] X. Hei, Y. Liu, and K. Ross, "IPTV over p2p streaming networks: the mesh-pull approach," *Communications Magazine, IEEE*, Feb. 2008.

[11] Z. Chen, K. Xue, and P. Hong, "A study on reducing chunk scheduling delay for mesh-based p2p live streaming," in *GCC '08*, 2008.

[12] N. Magharei and R. Rejaie, "PRIME: Peer-to-peer receiver-driven mesh-based streaming," in *INFOCOM 2007*, 2007.

[13] A. da Silva, E. Leonardi, M. Mellia, and M. Meo, "A bandwidth-aware scheduling strategy for P2P-TV systems," in *P2P '08*, 2008.

[14] A. Caminero, O. Rana, B. Caminero, and C. Carrion, "Improving grid inter-domain scheduling with p2p techniques: A performance evaluation," in *GCC '08*, 2008.

[15] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs," in *SIGMETRICS '08*, 2008.

[16] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *ACM MULTIMEDIA '07*, 2007.

[17] D. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," *Selected Areas in Communications, IEEE Journal on*, Jan. 2004.

[18] X. Liao, H. Jin, Y. Liu, and L. M. Ni, "Scalable live streaming service based on interoverlay optimization," *Parallel and Distributed Systems, IEEE Transactions on*, Dec. 2007.

[19] C. Wu and B. Li, "rstream: Resilient and optimal peer-to-peer streaming with rateless codes," *Parallel and Distributed Systems, IEEE Transactions on*, Jan. 2008.

[20] E. Brosh, A. Levin, and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application-layer multicast trees," *IEEE/ACM Transaction on Networking*.

[21] S. Tayu, T. G. Al-Mutairi, and S. Ueno, "Cost-constrained minimum-delay multicasting," in *SOFSEM 2005*. Springer, Berlin Heidelberg, 2005, pp. 330–339.

[22] J. Könemann, A. Levin, and A. Sinha, "Approximating the degree-bounded minimum diameter spanning tree problem," *Algorithmica*, February 2005.

[23] B. Brinkman and M. Helmick, "Degree-constrained minimum latency trees are APX-Hard," Miami University, Tech. Rep., 2008.

[24] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *Networking, IEEE/ACM Transactions on*, Dec. 2008.

[25] E. M. Arkin, M. A. Bender, S. P. Fekete, J. S. B. Mitchell, and M. Skutella, "The freeze-tag problem: How to wake up a swarm of robots," *Algorithmica*, October 2006.

[26] M. Sztainberg, E. Arkin, M. Bender, and J. Mitchell, "Theoretical and experimental analysis of heuristics for the "freeze-tag" robot awakening problem," *Robotics, IEEE Transactions on*, Aug. 2004.

[27] F. Lam and A. Newman, "Traveling salesman path problems," *Mathematical Programming*, May 2008.

[28] C. H. Papadimitriou and S. Vempala, "On the approximability of the traveling salesman problem," *Combinatorica*, Feb. 2006.