Fast-Mesh: A Low-Delay High-Bandwidth Mesh for Peer-to-Peer Live Streaming

Dongni Ren, Yui-Tung Hillman Li, and S.-H. Gary Chan, *Senior Member, IEEE*

***Abstract*—Peer-to-peer (P2P) technology has emerged as a promising scalable solution for live streaming to a large group. In this paper, we address the design of an overlay mesh which achieves low source-to-peer delay, accommodates asymmetric and diverse uplink bandwidth, and continuously improves delay based on an existing pool of peers. By considering a streaming mesh as an aggregation of data flows along multiple spanning trees, the peer delay in the mesh is then its longest delay (including both propagation and scheduling delay) among all the trees. Clearly, such delay can be very high if the mesh is not designed well. In this paper, we propose and study a mesh protocol called *Fast-Mesh*, which optimizes such delay while meeting a certain streaming bandwidth requirement. *Fast-Mesh* is particularly suitable for a mildly dynamic network consisting of proxies, supernodes, or content distribution servers.**

We first formulate the minimum delay multiple trees (MDMT) problem and show that it is NP-hard. Then we propose a centralized heuristic based on complete knowledge, which may be used when the network is small or managed, and serves as an optimal benchmark for all the other schemes under comparison. We then propose a simple distributed algorithm, *Fast-Mesh*, where peers select their parents based on the concept of *power* in networks given by the ratio of throughput and delay. By maximizing the network power, our algorithm achieves low delay. The algorithm makes continuous improvement on delay until some minimum delay is reached. Simulation and PlanetLab experiments show that our distributed algorithm performs very well in terms of delay and source workload, and substantially outperforms traditional and state-of-the-art approaches.

***Index Terms*—Minimize mesh delay, multimedia communication, P2P live streaming.**

I. Introduction

In order to provide live streaming services (such as IPTV) to a group of peers in the absence of IP multicast support, traditionally client-server model is used where a server serves individual participants directly [1]. This model clearly is not scalable to a large group. Peer-to-peer (P2P) live streaming has recently been proposed to overcome this problem where the server only needs to stream to some peers, who in turn share their stream received with their neighbors by means of their uplink bandwidth. Such P2P systems have shown to be effective in serving quite a large group with very low server bandwidth required [2], [3].

In P2P live streaming, peers join an overlay in a distributed manner. In this work, our goal is to design an overlay whose peer traffic is mildly dynamic (e.g., consisting of supernodes, proxies, and content distribution servers).[1] The overlay achieves the following.

- *Low delay:* Our target is live streaming. Therefore, an overlay offering low source-to-peer delay is desirable. We would like to design such an overlay which minimizes the maximum delay of the peers.
- *Meeting streaming bandwidth requirement in the presence of asymmetric uplink bandwidths:* Peers in the network may have diverse uplink bandwidth depending on their access network (such as ADSL, broadband Ethernet, Wireless LAN, cable, etc.) The overlay should meet a certain streaming rate requirement for each peer in spite of this bandwidth heterogeneity or asymmetry.
- *Accommodation of peer churn:* Peer traffic in the network can be dynamic, i.e., a peer may join or leave at anytime. The overlay structure should accommodate this network dynamic and be adaptive to such peer churn to achieve high performance.
- *Distributed, simple, and self-improving:* The protocol should be distributed and its performance should be scalable to a large number of peers. The protocol should be self-improving in the sense that it continuously improves and adapts to the overlay based on the heterogeneous peer characteristics. It should also be simple so that it can be implemented.

Clearly, to meet the above objectives, streaming mesh should be used [4]. In mesh, each peer maintains a list of neighbors to exchange information. A peer obtains its stream by aggregating the flows from its many parents using either pull-based or push-based methods. Fig. 1(a) shows an overlay example of streaming using mesh. S is the streaming source and A, B, C, and D are four peers in the streaming session. S streams media to peers A and B, who in turn stream to peers C and D. Due to insufficient uplink bandwidth between A and D, B streams part of the stream to D. This is similar for C, which has two parents. Note that the ancestor-descendant relationship in the network may have loops; the loop formed between A and D simply means that A gets some part of the stream earlier and shares it

[1]In this paper, "peers" refer to proxies, supernodes, or content distribution servers.

Manuscript received September 11, 2008; revised June 03, 2009. First published September 22, 2009; current version published November 18, 2009. This work was supported in part by the Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation (SVCF08/09.EG01), and in part by the Hong Kong Innovation Technology Fund (ITS/013/08). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Gene Cheung.

The authors are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: tonyren@cse.ust.hk; hillmanl@cse.ust.hk; gchan@cse.ust.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2009.2032677

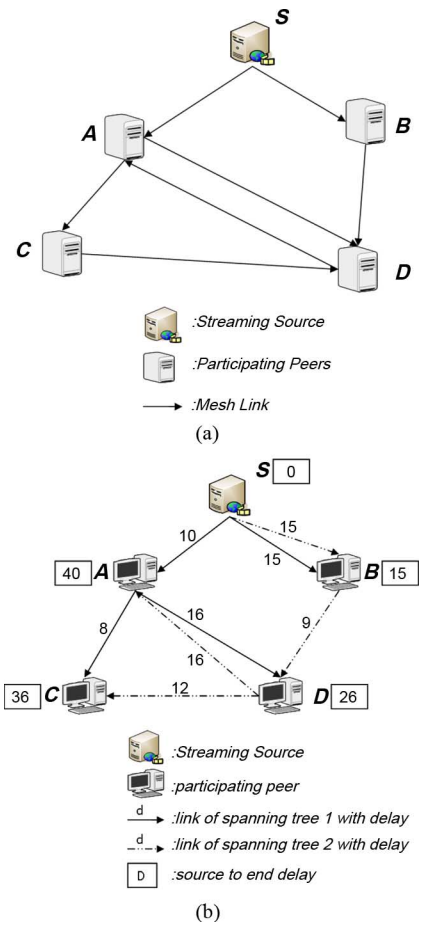


Fig. 1. Streaming with mesh and its constituent underlying spanning trees. (a) Overview. (b) Trees.

with D while D gets the other part earlier and shares it with A. By exchanging their contents with each other, both A and D can assemble a full stream. Since each spanning tree delivers a unique piece of stream, it is not possible to loop the same piece of data among two peers. The asymmetric bandwidth problem is overcome by aggregating the bandwidth of multiple parents to achieve the bandwidth requirement of a full stream.

However, this bandwidth guarantee comes with the cost of delay. Refer to Fig. 1(b) where we show how overlay delay is accumulated by representing the streaming mesh in Fig. 1(a) equivalently as two spanning trees of certain data bandwidth (labeled in solid and dashed lines). The number by the arrows is the overlay delay (in units), consisting of end-to-end propagation delay and scheduling delay due to the packet transmission time and scheduling policy. The square boxes indicate the source-to-end delay of the peer, which is given by the largest delay of the node in all the spanning trees. For example node B has a delay of 15 given the same overlay paths in both trees. Because node D has a delay of 26 in one tree and 24 in another, its delay is $\max(26,\ 24) = 26\,(\text{units})$. Similarly A's delay can be obtained as $\max(10,\ 15 + 9 + 16) = 40$ and C's delay as $\max(10 + 8,\ 15 + 9 + 12) = 36$. We see that the delay accumulates quite quickly as the number of peers increases.

From the above example, it is not difficult to see that in general, streaming mesh can be pictured as the superposition (or aggregation) of packet flows along multiple spanning trees. The mesh delay of a node is the number indicated in the square boxes in Fig. 1(b). Such delay is due to the longest path from the source to the node in all the spanning trees. Mesh delay captures the path length of packets from the source to the node. Minimizing such delay also leads to low packet scheduling delay arising from different packet arrival time at a node. In P2P streaming, there are other uncontrollable packet delay factors, such as packet transmission jitter, loss, etc. These are not particularly sensitive to the number of peers and can hardly be optimized, and hence will not be considered in this paper. Mesh delay can be high especially when the peer population is large and the mesh is not designed properly. Such mesh delay accumulates fast as the stream has to go through many hops and many constituent trees. In this work, we focus on mesh construction minimizing the mesh delay of the nodes. The design principle is to begin with a mesh with an "optimized" low delay using a mesh construction algorithm, then applying some buffering to mitigate jitters, we achieve an overall low delay for live streaming.

To the best of our knowledge, this represents the first body of work addressing the optimization of mesh delay for P2P streaming. We address the problem from the following three directions.

1) *Problem formulation and a centralized heuristic.* As previously discussed, low source-to-end delay is one of the most important requirements for live streaming service. In our work, we try to minimize this delay so that all peers can have low delay in real-time streaming. Therefore, we first formulate the minimum delay multiple trees (MDMT) problem, which is to form a low-delay mesh meeting a certain streaming rate requirement for all nodes. Note that the mesh may have loops and two peers are allowed to exchange contents between each other, e.g., nodes A and D in Fig. 1. We represent the mesh as the aggregation of packet flows along multiple spanning trees. We seek to minimize the maximum delay of the peers in the mesh. We show that the problem is NP-hard, and propose a centralized heuristic (based on complete knowledge) which achieves very good performance. This serves as a benchmark for our later comparison study.
2) *Fast-Mesh: A distributed protocol for low-delay high-bandwidth mesh.* Based on the approach of our centralized heuristic, we propose a novel distributed protocol called Fast-Mesh which builds a mesh with random peer joins and leaves. Here,"Fast" means low delay and high bandwidth (i.e., meeting a certain streaming rate requirement). Our protocol works especially well when frequent peer failures are unlikely, e.g., a proxy or super node network. In a proxy-based peer-to-peer streaming network, Fast-Mesh can build a robust, efficient, and high-bandwidth proxy backbone to provide better stream delivery to the peers. In Fast-Mesh, a new arrival initially settles for a good set of parents. The mesh continuously improves itself with the peers locating and connecting to better parents to further reduce their delay. Such adaptation mechanism is able to move the nodes to appropriate positions in the mesh with very few steps.

3) *Simulation and experimental studies on the algorithms.* We conduct simulation and Planetlab study on our centralized and distributed algorithms, and compare them with traditional and state-of- the-art approaches (closest-parents and Outreach [5]). Results show that our algorithms achieve substantially lower delay and server workload.

The rest of this paper is organized as follows. We first review related works in Section II. Then we present the formulation of the MDMT problem and the centralized heuristic in Section III. In Section IV, we discuss Fast-Mesh, a distributed algorithm to build a low-delay high-bandwidth mesh. Illustrative results and comparisons based on both simulation and Planetlab experiments are presented in Section V. We conclude in Section VI.

II. Related Work

In this section, we briefly review previous work. Because a peer in a mesh is served by many parents, a packet scheduling mechanism is needed to decide when a parent should send which packet. There has been much study on this to reduce packet reassembly delay or to improve throughput (see, for examples, [6]–[9] and references therein). Our work is orthogonal to this body of work, and the mesh built may apply any of the scheduling algorithms to achieve low-delay streaming.

It has been proposed to use a single tree to distribute streams among peers, where all peers are arranged into a tree rooted at the source [10]–[13]. The media is streamed down from the source to every peer along the tree edges. Though the tree approach is simple and achieves low delay, the streaming rate cannot be guaranteed as it is limited by the least uplink bandwidth of the nodes in the tree. Furthermore, opposite to mesh, tree cannot accommodate network dynamics well to achieve stream continuity. There has been much work on how to construct trees for overlay streaming. Centralized algorithms such as CoopNet and ALMI build a tree rooted at the streaming source [11], [12]. Narada first constructs an overlay mesh, and then spanning trees with multiple sources are generated on top of the mesh for data delivery [13]. NICE and ZIGZAG are distributed protocols which arrange the participating peers into clusters and layers in a distributed manner to minimize delay and workload [14], [15]. However the structures cannot be easily maintained.

Mesh-based streaming systems are most popular nowadays [16], [17]. Most of the existing work on mesh is based on randomly connecting the participating peers to neighbors for data exchange in a rather ad hoc or random manner (and leave no mesh optimization) [18]–[20]. Based on gossip, the peers gradually connect to some closest parents for data exchange. Such *closest parents* approach leads to high delay from the source-to-end hosts. Chainsaw is built based on request-response data dissemination and gossip protocol [21]. Peers request fresh data from neighbors in a BitTorrent-like manner. As compared to the above, we present a formal study on mesh optimization and a distributed protocol which achieves much lower delay by optimizing the mesh structure through parent selection and adaptations.

The asymmetric bandwidth problem has been studied in Outreach, which aims to minimize the source workload by making full use of the peers' uplink bandwidth [5]. Every peer in the network estimates its upstream and downstream bandwidth difference. A newcomer randomly asks one of the source children to perform the estimation and connects to the peers with the largest bandwidth difference. As compared with Outreach, Fast-Mesh achieves better performance because it continuously adapts and improves the streaming mesh, and strategically places powerful peers in appropriate upstream locations, e.g., at positions where they enjoy low delay.

We have studied the problem of reducing mesh delay in our previous work, which formulates the minimum delay mesh (MDM) problem [22]. Our current work differs and advances from it in three major ways: 1) MDM assumes no loops in the network, i.e., the ancestor-descendant relationship cannot have cycle(s). Under this setting, the problem formulation is more restrictive than the current work. The "no loop" assumption limits itself to fully make use of the uploading bandwidth of peers. In the new MDMT formulation of this paper, we consider the streaming mesh as an aggregation of multiple trees. Each tree delivers a unique substream to all the nodes. In this way, nodes are allowed to exchange substreams with each other, hence allowing loops, i.e., parent-child relationship may have cycles. Our current MDMT formulates a much more general, common, effective, and useful case of peer-to-peer streaming network than MDM. It is hence more powerful and reflects reality. 2) We also propose a new centralized heuristic and present the benefit of loops in the result section. Without the "no loop" constraint, the heuristic algorithm for MDMT formulation achieves lower peer delay because of a better utilization of the uploading bandwidth of peers, especially when the network resource (average peer bandwidth) is scarce. 3) Besides simulations, we have implemented our algorithm and conducted Planetlab experiments on real Internet, and our algorithm is shown to perform very well.

III. Problem Formulation and a Centralized Heuristic

In this section we first present the formulation of the MDMT problem in Section III-A. Given the complete knowledge of the overlay network, we then present a centralized heuristic to solve the problem in Section III-B.

A. Problem Formulation

We model the overlay network as a complete directed graph $G = (V, E)$, where V is the set of vertices including the streaming source and the participating peers. $E = V \times V$ is the set of overlay edges. For any edge $\langle i, j \rangle$ in G, the cost d_{ij} of the edge is the worst-case delay from node i to node j, given by the sum of the underlay unicast path delay from node i to node j in the physical network and the worst-case scheduling delay (given by segment size times the maximum number of children that the node can serve divided by its upload bandwidth).

We consider that bandwidth is normalized according to some units. Each unit is the minimum packet flow rate between any two nodes. For example, if a unit is 100 kbps, then the streaming rate of a 300 kbps video is 3 units. A single source node S streams data to all nodes at a rate of s units (the streaming rate), where $s \in Z^+$, $S \in V$. Each unit of stream is delivered to all the nodes in V by a specific spanning tree. Note that each tree

delivers a unique unit, and hence, there are exactly s spanning trees in total. We consider the general case that the network core is not the bottleneck of the network (i.e., bandwidth is limited by the end hosts since the mesh is designed with close peers connecting with each other). Denote T_k as the spanning tree of the kth unit, and $D_i(T_k)$ as the source-to-end delay of node i in spanning tree T_k. Note that T_k's do not need to be distinct and they may share some edges. Clearly if node j is the parent of node i in T_k, $D_i(T_k) = D_j(T_k) + d_{ji}$. Denote the mesh delay of node i as D_i, which is given by its maximum source-to-end delay $D_i(T_k)$ among all its spanning trees, i.e.,

$$D_i = \max_{k\in[1,s]} D_i(T_k). \tag{1}$$

For every node i in V, it has an uplink bandwidth of U_i units, $U_i \in Z^+$, which represents the maximum number of children it can serve in all spanning trees. For any node i in V, if it gets an aggregate stream of s units from its parents, we call node i *fully served*. In other words, if node i receives streams from all s spanning trees, it is *fully served* and can play back the video smoothly. Note that S has a uplink bandwidth of U_S units, and it does not need to be served by other nodes. We assume a streaming mesh exists, which requires the total uplink bandwidth to be larger than the total downstream bandwidth, i.e.,

$$\sum_{i\in V} U_i \geq (|V| - 1) \times s. \tag{2}$$

The *MDMT problem* is to find a mesh which minimizes the maximum of the peer delay, i.e.,

$$\min \max_{i\in V} D_i \tag{3}$$

subject to the streaming requirement, i.e., all nodes receive an aggregate incoming stream of s units.

1) Claim: The MDMT problem is NP-hard.

Proof: Travelling salesman problem (TSP) is reducible to our MDMT problem in polynomial time. An input to TSP is a weighted, undirected complete graph $G(V,\ E)$ and a vertex $S \in V$. Note that the TSP is to find a tour of minimum cost through all vertices exactly once (Hamiltonian cycle) such that S is both the starting point and ending point.

The problem is in P. The maximum delay of a mesh can be calculated in polynomial time. Therefore given a graph $G(V,\ E)$ and the min-max delay of the problem, we can verify whether the mesh is the optimal solution. Now we prove that TSP can be reduced into MDMT problem. The polynomial time transformation is as follows. Let $G'(V',\ E')$ be the graph of a TSP instance. We transform $G'(V',\ E')$ into $G''(V'',\ E'')$ by adding a vertex S_{end} and edges from all the vertices to S_{end}. In this way, the vertices in V'' represent peers and the weight on the edges are the delay between the two adjacent peers. We let S be the source, and consider the special case that the uplink bandwidth of each peer is 1, and S_{end} has zero uplink bandwidth. Consider also the streaming rate to be 1. In this way, the resulting overlay topology must be a chain starting at S and ending at S_{end}. $D_{\max}$ equals to the delay of S_{end} which is the sum of all delays preceding it. Hence, it is obvious that $D_{\max}$ in G'' is minimum if and only if the cost of a tour in G' is minimum. Therefore, TSP is polynomially reducible to MDMT.

B. Centralized Heuristic

Given the NP-hard nature of our problem, we propose a centralized heuristic based on complete knowledge (i.e., knowledge of the user pool at the beginning and pairwise distances between them). The algorithm is suitable for small networks and serves as a benchmark for the evaluation of our proposed distributed protocol.

From (1), we can see that the mesh delay of a node is determined by the spanning tree with longest delay. Clearly, in order to achieve overall low delay in the mesh, each node should maintain low delay in all s spanning trees constructed. Since the node delay is determined by the slowest tree, a good heuristic should balance the delay of each node in all trees. Therefore for each node, we reserve a certain amount of uplink bandwidth for every tree. In this way if a node connects to a parent with low delay in one tree, it could also connect to the same parent in other trees. As a result, the source-to-end delay of a node in all trees would be similar. In this way, there would be no bottleneck trees which may significantly bring up the overall delay for a peer.

Without loss of generality, we assume that the spanning trees are constructed in order from T_1 to T_s. Let the total residual bandwidth of node i be R_i, $R_i \in Z$. Note that R_i varies over time as the trees are formed. Let $B_i(T_k)$ be the maximum amount of bandwidth that node i uses for T_k, $1 \leq k \leq s$. We consider that the residual bandwidth R_i, given $(k-1)$ trees have been formed, is equally shared among the $(s-k+1)$ remaining trees, i.e.,

$$B_i(T_k) = \left\lceil \frac{R_i}{s-k+1} \right\rceil. \tag{4}$$

Note that under this bandwidth allocation scheme, there is enough uplink bandwidth reserved for each spanning tree to fully serve all the nodes. This is because given that

$$\sum_{i\in V} U_i \geq (|V| - 1) \times s \tag{5}$$

we have

$$\sum_{i\in V} B_i(T_k) \geq \frac{\sum_{i\in V} R_i}{s-k+1} \tag{6}$$

$$= \frac{\sum_{i\in V} U_i - (k-1)(|V|-1)}{s-k+1} \tag{7}$$

$$\geq \frac{s(|V|-1) - (k-1)(|V|-1)}{s-k+1} \tag{8}$$

$$\geq |V| - 1 \tag{9}$$

which simply says that the bandwidth is enough to serve all the nodes.

Define $r_i(T_k)$ as the residual bandwidth of node i as tree T_k is being formed. Our algorithm runs according to Algorithm 1. We iteratively construct s spanning trees. At the beginning of each iteration, we calculate $B_i(T_k)$ according to (4) for each node $i \in V$. Starting with the source node S as the root of the tree, we push all the nodes in V into the spanning tree T_k one by one using Prim's minimum spanning tree algorithm, with d_{ji} as the link cost, and nodes are connected to all of the parents in the subtree with a positive $r_j(T_k)$. After all the nodes are pushed into tree T_k, we move to the next iteration to construct tree T_{k+1}. The algorithm ends when all s spanning trees are constructed.

Algorithm 1 Pseudocode for the Centralized Algorithm

Input:

$G = (V,\ E)$

source node: S

uplink bandwidth: U_i for $i \in V$

edge delay: d_{ij} for $i, j \in V$

streaming rate: s

Output:

Mesh M composed of s spanning trees with minimum source-to-end delay

foreach *node* $i \in V$ **do**

$R_i = U_i$

end

for $1 \leq k \leq s$ **do**

$T_k = \emptyset$;

foreach *node* $i \in V$ **do**

Calculate $B_i(T_k)$;

$R_i(T_k) = B_i(T_k)$;

end

$T_k = T_k + S$;

while $T_k \neq V$ **do**

$minDelay = MAX$;

$u, v = null$;

foreach *node* $j \in T_k$ **do**

if $R_i(T_k) > 0$ **then**

foreach *node* $i \notin T_k$ **do**

if $D_j(T_k) + d_{ji} < minDelay$ **then**

$minDelay = D_j(T_k) + d_{ji}$;

$u = i$;

$v = j$;

end

end

end

Connect u to v;

$T_k = T_k + u$;

$R_v(T_k) = R_v(T_k) - 1$;

$R_v = R_v - 1$;

end

end

The complexity of the centralized algorithm is $O(s|V|^3)$. The algorithm constructs s independent spanning trees one by one. The time needed to construct one spanning tree is $O(|V|^3)$ since pushing each node into the tree needs $O(|V|^2)$ to search for the most suitable parent and child. Therefore, the time complexity of the centralized algorithm is $O(s|V|^3)$.

IV. Fast-Mesh: A Power-Based Distributed Algorithm

Given complete knowledge of the network topology and user pool, the centralized heuristic works well to minimize delay. Such a centralized heuristic works well for a small or managed network. In practice, we do not usually have such global information. A joining peer does not know all other current peers, let alone all the distances between them. Besides, if the user pool is large, we cannot have central planners keep track of thousands of peers. In this case, our centralized heuristic can serve as a benchmark for comparison. As our centralized heuristic performs well, we propose a simple, efficient, and distributed protocol called Fast-Mesh which is scalable to a large group.

We first introduce the parent selection algorithm for newly joining peers (Section IV-A). By selecting the right parents, newcomers are able to have a low source-to-end delay and good streaming quality. We then discuss the mesh adaptation mechanism, which is used to further adapt the existing mesh to achieve a better performance than the current one (Section IV-B). We end by discussing node leave operation (Section IV-C).

A. Node Joins and Parents Selection

In a distributed streaming system, it is not feasible to construct spanning trees one by one. Therefore, we construct a mesh directly by connecting joining peers to multiple parents. Each parent serves several units/s of streams until the streaming rate is received by the joining peer. For protocol simplicity, we do not keep track of the delays in each spanning tree; instead, the delay of peer i is calculated as $D_i = \max_{j \in P_i}(D_j + d_{ji})$, where P_i is the set of parents of node i.

In mesh construction, we need to find a balance between delay and bandwidth. Ideally peers should connect to a parent with low delay to the source, but such parents may have residual bandwidth lower than a full stream. If more parents are connected, the delay inevitably increases. Therefore, a greedy approach based only on delay is not expected to perform well. We propose to make use of a concept similar to *power* to achieve

a balance between the delay and residual bandwidth. Traditionally in networking, *power* is defined as the throughput divided by delay. In this paper, we define the *power* between a peer i and its parent j, denoted as $P_i(j)$, as the rate that j is serving i divided by the source-to-end delay of i from j, i.e.,

$$P_i(j) = \frac{\min(R_j,\ s)}{D_i(j)} \tag{10}$$

where

$$D_i(j) \equiv D_j + d_{ji}. \tag{11}$$

The larger $P_i(j)$ is, the better node j is as a parent of node i. [We have used such power concept in our centralized heuristic, where s in (10) is replaced by 1, as each substream is of one unit of bandwidth.]

A new peer i contacts a *rendezvous point (RP)* which caches a list of recently arrived peers. The *RP* returns a few nodes to i. Peer i checks its delay from these nodes and requests their residual bandwidth. It then evaluates its power to each of them according to (10), and chooses parents in a greedy manner, i.e., it first connects to the parent with the highest power. If this parent cannot fully serve it, it then connects to the second one and this process repeats until it is fully served. If the peers returned by the *RP* cannot fully serve the newcomer, the newcomer requests the neighbor of those peers. Then the above process is repeated until it is fully served.

B. Adaptation to Support Peer Churn

In a distributed environment, peers may join or leave at any time. Therefore, an optimal mesh should adapt to the current network environment. This means that some nodes may need to shift in position in the streaming mesh to achieve lower overall delay. This is the goal of our adaptation algorithm.

The adaptation consists of three steps, *Request*, *Grant* and *Accept*, detailed as follows.

1) Request: If its residual bandwidth is greater than the streaming rate, a peer sends all its parents a REQUEST message which contains its total uplink bandwidth and a time-to-live field (TTL) indicating the number of upstream levels REQUEST will go.

2) Grant: Upon receiving REQUEST, a node first decrements the TTL. If the TTL is nonzero, it forwards the REQUEST message upstream to all its parents; otherwise, it is discarded. The node also checks whether its uplink bandwidth is lower than that of the REQUEST sender. If it is, the node responds to the sender with a GRANT message which contains its source-to-end delay. The GRANT message gives the REQUEST sender the permission of taking over the node's position in the mesh.

3) Accept: A node may receive a number of GRANT messages, from which the ancestor that is the shortest distance from the source is chosen. The node replaces all its existing parents with that ancestor's parents and adopts the ancestor as its own child. In this way, the higher-bandwidth node is moved closer to the source.

The key parameter in the adaptation protocol is the TTL which determines how far away the REQUEST messages travel. If TTL is high, on the one hand, more ancestors may be reached and hence the node may perform a longer jump in upstream positions. On the other hand in each step, this may result in flooding the network. Our simulation results show that a TTL of around 2 gives very good performance.

C. Node Leaves and Failures

When a peer is about to leave, it initiates a LEAVE message to its parents, which releases the uplink bandwidth to the leaving peer. It also sends a LEAVE message to all of its children so that they can look for new parents in the mesh using the join process. Peers also periodically exchange KEEP-ALIVE messages with their parent nodes and children. When a node failure occurs, its parents release their uplink bandwidth and its children seek for new parents in the mesh.

Note that the protocol described here constructs a loop-free mesh with low delay. In Section V, we see that if the total overlay bandwidth is not a terribly scarce resource (the usual case), the resulting mesh is likely to be loop-free. Therefore, we choose not to use loops to keep protocol simple.

V. Illustrative Simulation Results

In this section, we first present our simulation environments and metrics (Section V-A), followed by our illustrative simulation results in Section V-B. We have implemented Fast-Mesh and conduct experiments in the PlanetLab. The experimental results are discussed in Section V-C.

A. Simulation Environment and Metrics

We use BRITE to generate different two-level top-down hierarchical topologies [23]. Each topology consists of a number (8) of autonomous systems, each of which has many (625) routers. Brite also provides us link latency in a millisecond and peers are attached to the routers randomly. The peer bandwidth is neither the same nor uniformly distributed; instead, rich diversity has often been found in the Internet [24], [25]. Their access link bandwidth distribution is shown in Table I, which is according to an extensive bandwidth measurement from a large-scale real-world streaming event [26]. (For the sake of clarity, we measure the bandwidth in streaming units, a normalized measure of bandwidth in kbps.) We have also run the simulations with other bandwidth distribution, and the results are qualitatively the same. Unless otherwise stated, we use the following baseline parameters: $s = 10$ units, $TTL = 2$, number of peers $= 500$. We assume the realistic scenario of a distributed global network with small segment size (the usual case for tree-push as in our study), and hence, the scheduling delay is negligible as compared with propagation delay. (Our results are qualitatively the same if a worst-case scheduling delay is added on each overlay path.)

We use the following evaluation metrics.

- *Delay:* The primary concern of our protocol is the source-to-end delay of the peers. This is the time taken for data to travel from the streaming server to the peers.

TABLE I
UPLINK BANDWIDTH DISTRIBUTION OF PEERS

Uplink Bandwidth, uniform integral steps (unit)	Num of Peers (percentage)
0	30%
$1-10$	58%
$20-90$	5%
100	7%

We are mainly interested in the distribution, average, and maximum delay among all peers in the mesh.

- *Hop count:* Hop count is the maximum number of intermediate peers on the overlay path from the source to a peer. It gives us an idea of the depth of the overlay. Though a lower hop count does not necessarily mean lower delay, it does tell us the depth of the mesh and how the algorithms position peers. Putting high-bandwidth peers near the source allows branching to occur earlier, thus giving a flatter topology. We are interested in both the average and maximum hop count of the peers.
- *Source workload:* Source workload is defined as the amount of bandwidth that the source uploads data to the peers directly connected to it.

We compare the performance of our centralized scheme and Fast-Mesh with two other schemes, namely, closest parent and Outreach. In the closest parent scheme, newly-arrived peers choose parents closest to them. This scheme is slightly better than picking parents randomly as it can capture the locality of the peers. This is simple, and therefore, quite a number of streaming systems nowadays adopt it. Outreach have been reviewed in Section II, and its details can be found in [5]. At last, we explore the influence of loops in mesh streaming (by comparing our loop-based algorithm with the loop-free one as stated in [22]).

B. Illustrative Simulation Results

To illustrate the benefit of adaptation, in Fig. 2, we show the average delay of peers against their bandwidth with and without adaptation. Clearly, we see that the average delay is reduced by using adaptation. This means that by putting high bandwidth peers closer to the source, most peers can receive the data streams sooner.

It should be noted that adaptation does not guarantee bandwidth to vary with delay, though the peers with high bandwidth are moved upwards aggressively. There are two reasons for this. First, the ancestors of high-bandwidth peer may also possess high bandwidth, so adaptation does not happen between them. Second, those ancestors may have some other children (i.e., siblings to the successor) with lower bandwidth. These peers, therefore, stay in the upper level of the mesh and enjoy relatively low delay.

We compare the average and maximum delay of various schemes versus the number of peers in Fig. 3. As the simulation shows, delay increases when the number of peers increases. The data need to go through more hops before reaching all the peers. The centralized scheme performs the best due to its complete knowledge of the network. Fast-Mesh performs close to the centralized algorithm and substantially better than the other schemes. Having a small delay with low growth rate means Fast-Mesh is scalable to a large number of peers. This illustrates the effectiveness of the heuristic combined with adaptation in achieving low delay streaming mesh.

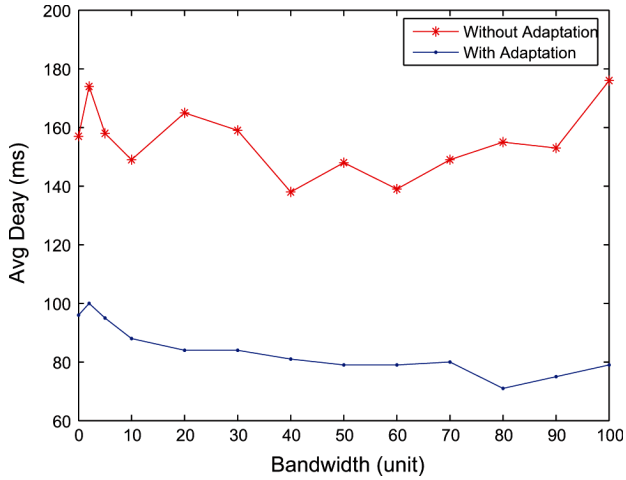


Fig. 2. Delay of peers with different bandwidths.

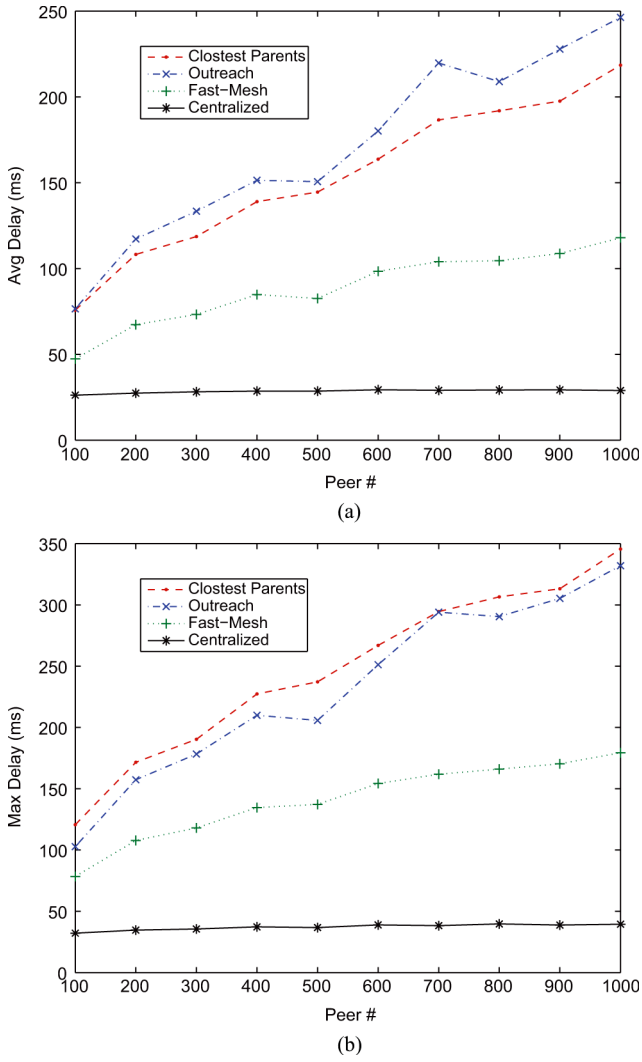
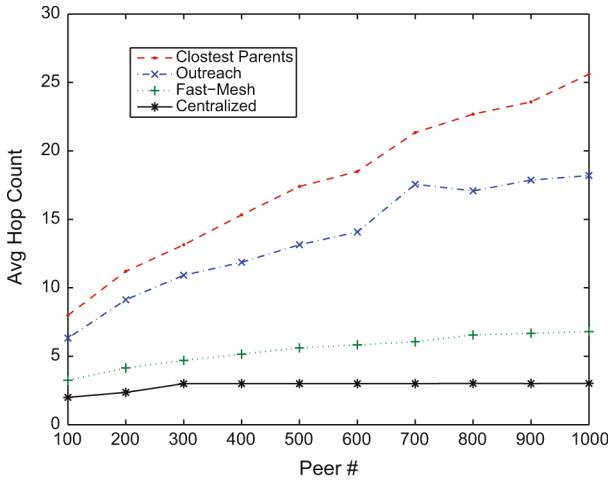
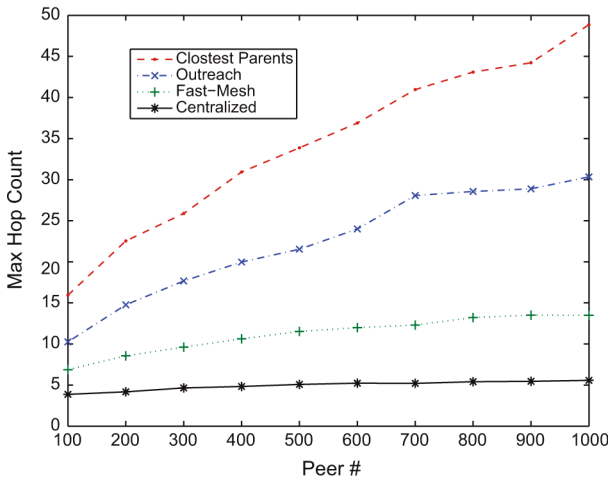


Fig. 3. Delay versus number of peers. (a) Average delay. (b) Maximum delay.



(a)



(b)

Fig. 4. Hop count versus number of peers. (a) Average hop count. (b) Maximum hop count.

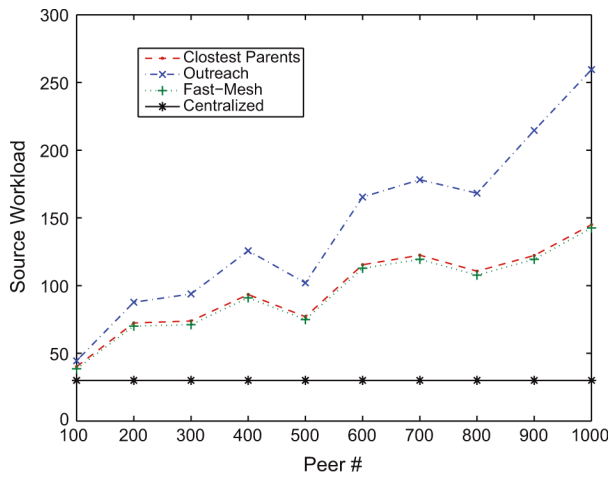


Fig. 5. Source workload versus number of peers.

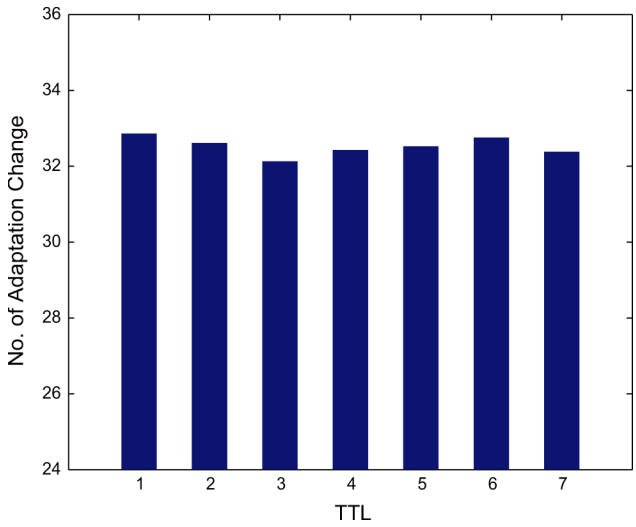


Fig. 6. Number of adaptations versus TTL.

Fig. 4 shows the average and maximum hop count of the schemes versus the number of peers. Similar to delay, the hop count increases with peer number and Fast-Mesh performs close to the centralized scheme and outperforms the other schemes.

Fig. 5 shows the amount of bandwidth consumed at the source (source workload) versus the number of peers. The source uploads to more peers as the number of peers increases. The centralized scheme performs the best since it can most efficiently utilize the uplink bandwidth of the peers. In Fast-Mesh and closest parent scheme, the source has roughly contributed the same amount of resources in order to keep the streaming mesh performing. In fact, both of these schemes will try to utilize the peers bandwidth as much as possible. Sometimes after a considerable number of trials searching for good parents, a newcomer still cannot find satisfactory parents. The best thing it can do is to connect to the streaming source. Although the source workload of these schemes is larger than the centralized scheme, in the lack of global knowledge, these values are acceptable. Outreach actively places peers under source, and thus, its reliance on the source is relatively larger.

We define the *Number of Adaptation Changes* as the number of peer position changes before the mesh settles into a steady state. We evaluate the effect of different TTL values in the adaptation protocol. Fig. 6 shows the number of adaptation changes versus the value of TTL. The more the adaptation happens, the more often peers change their positions in the mesh. We can therefore think of the cost of adaptation proportional to the number of adaptations that occur. As expected, small TTL values result in more adaptation; however, increasing the TTL does not reduce the amount of adaptation changes by much. From the simulation, we find that most adaptations, even for large TTL values, only actually advance peers a few levels. This indicates that even a small change in peers position can give a considerable improvement over delay.

Beside comparing the number of adaptation occurrences, we would like to know the delay reduction with different TTLs. To this end, we use *average (maximum) delay reduction*, the ratio of the average (maximum) delay reduction by adaptation to the average delay without adaptation, to measure the proportion of peer delay reduction. If these values are positive, there is a reduction in delay time; otherwise, the delay has in fact gotten longer.

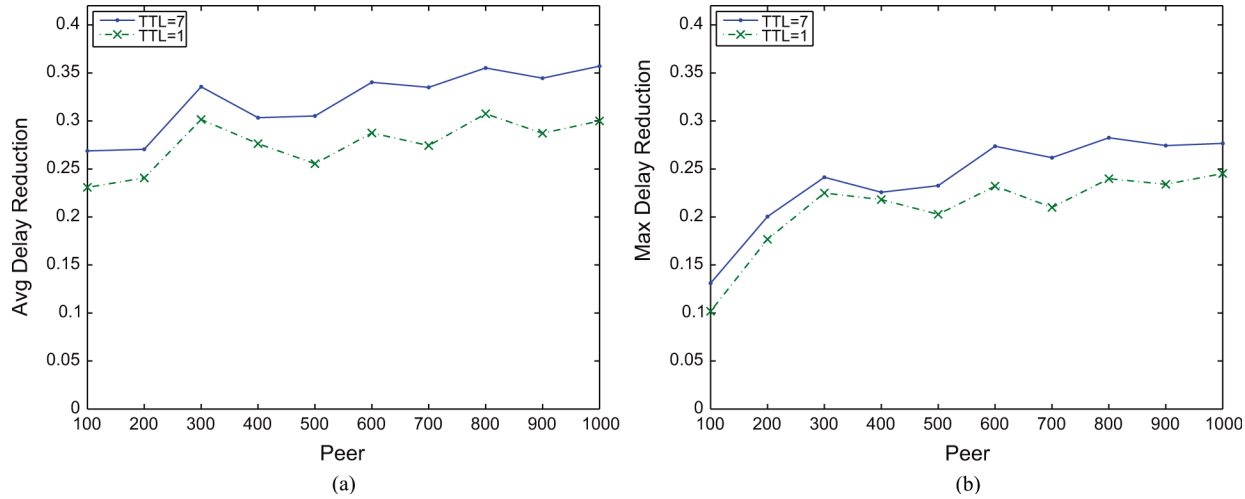


Fig. 7. Delay reduction versus number of peers. (a) Average. (b) Maximum.

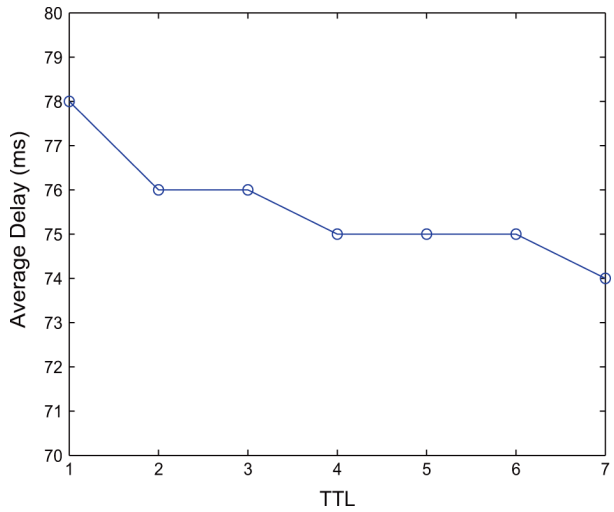


Fig. 8. Average peer delay versus TTL.

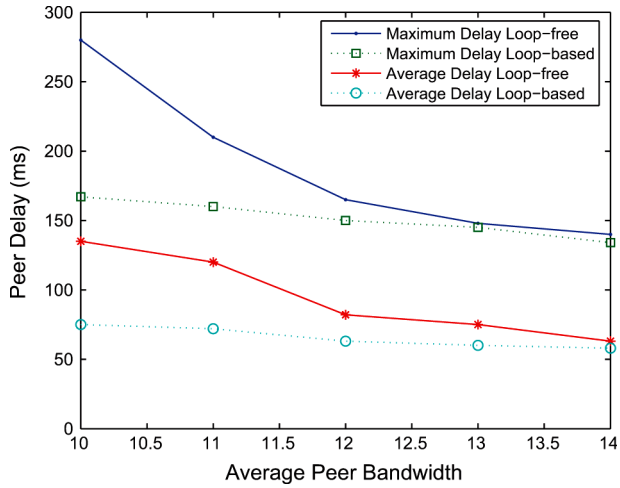


Fig. 9. Delay under loop and loop-free conditions with different bandwidth.

Fig. 7 plots the average and maximum delay reduction versus the number of peers. For clarity reasons, we only show the extreme cases of TTL equaling 1 and TTL equaling 7 (the graphs of other TTL values lie in between these two). The delay reduction TTL equaling to 7 is slightly larger than the case when TTL equals to 1. The adaptation can significantly reduce peer delay. It is important to delay sensitive streaming applications because it allows shorter delivery time to transmit data from the source to end-hosts. In this way, the peers will be less likely to miss the playing deadline when it receives the data. Fig. 8 shows that when the TTL increases from 1 to a higher value (say 7), delay drops by only a little. Given system complexity considerations, the TTL does not need to be high. Good values to have range from one to three.

Finally, we explore the influence of loops in mesh streaming by comparing our loop-based centralized algorithm with the loop-free centralized algorithm proposed in [22]. The average and maximum delays of the two algorithms we study are shown in Fig. 9, where our algorithm is denoted as "loop-based" and the algorithm in [22] is denoted as "loop-free". We see that our "loop-based" algorithm is generally better than the "loop-free" algorithm. Moreover, under a scarce resource setting, where there is little residual bandwidth left after fulfilling the required stream rate for all the peers, the proposed algorithm significantly outperforms the loop-free mesh.

Fig. 9 shows that having loops in the mesh helps reduce delay in a tense bandwidth situation, and under a more realistic bandwidth setting, the meshes, with or without loops, have a similar delay. This is why we construct a loop-free mesh under a real-world bandwidth setting, and also because loops complicate the mesh structure and bring new scheduling problems.

C. Illustrative PlanetLab Experiment Results

We have implemented Fast-Mesh and run it in PlanetLab. PlanetLab is a testbed for overlay networks, and it has been widely used in the networking community to study the performance of new protocols. We present some illustrative experimental results in this section. The PlanetLab nodes in the experiments are mainly located in East Asia, North America, and Europe. The server is located in HKUST (Hong Kong) with a ten-unit streaming rate. (We assume each unit is 30 kb/s.) The

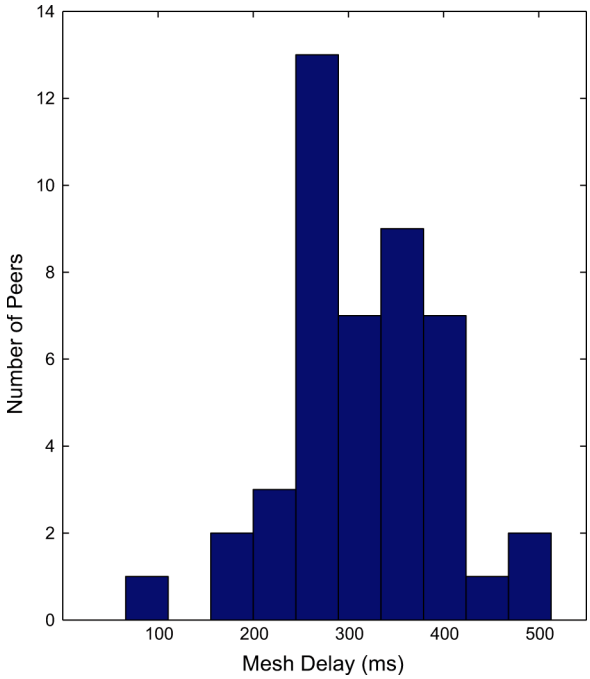


Fig. 10. Peer delay distribution.

distribution of the uplink bandwidth of the nodes follows the same distribution as in Table I. The TTL field is set to 1 in the experiment for simplicity.

Fig. 10 shows the delay distribution among all the participating peers. Most of the peers maintain a relatively low mesh delay. Since the server is placed in Hong Kong, peers in the U.S. and Europe have a higher delay than those located in mainland China, Taiwan, and Japan.

Fig. 11 shows a snapshot of Fast-Mesh at a steady state (for 45 PlanetLab nodes). We make several important observations below. First of all, the mesh is "tree-like"; most nodes have only one parent delivering all ten substreams. Not many (10%–20%) peers have multiple parents to meet the streaming bandwidth requirement; even in this case, the nodes have few parents (mostly two), and they are mostly located on the leaves. This all agrees with our simulation results and shows that given a certain bandwidth requirement, Fast-Mesh performs well in locating close parents and constructing low-delay mesh. From the mesh, we also observe that, due to ISP-peering issues, some substreams need to go to Japan before going to China, and most of the nodes in Europe are served via America instead of directly from Asia.

The experiment shows that Fast-Mesh constructs a low-delay mesh suitable for P2P live streaming. The data we collected from the experiment agree with our expectation as well as the simulation results.

VI. Conclusion and Future Work

In this paper, we investigate mesh construction minimizing delay for P2P live streaming, with peers of asymmetric bandwidths. We formulate the mesh optimization as the MDMT problem, which is to form a low-delay mesh meeting a certain streaming rate requirement. We show that the problem is NP-hard and derive a centralized algorithm as the benchmark. We then propose a distributed protocol called Fast-Mesh based on heuristic and periodic adaptation. In Fast-Mesh, peers select parents with high bandwidth and low delay based on the *power* heuristic. The mesh continuously improves itself with peer adaptation to further reduce delay.

Fig. 11. Mesh formed by Fast-Mesh.

We have conducted simulation and PlanetLab studies on our centralized algorithm and Fast-Mesh. Simulation results show that Fast-Mesh achieves much lower delays than techniques commonly used. The adaptation process significantly reduces mesh delay and source workload is kept low. From the PlanetLab experiment conducted, we see that our Fast-Mesh protocol is able to construct a low-delay mesh suitable for live streaming. In the future work, we plan to enhance our Fast-Mesh in terms of robustness to accommodate the highly dynamic network, i.e., an environment with frequent peer departures and failures.

References

[1] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "Insights into PPlive: A measurement study of a large-scale P2P IPTV system," in *Proc. IPTV Workshop, Int. World Wide Web Conf.*, 2006.

[2] X. Hei and C. Liang, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007.

[3] T. Silverston and O. Fourmaux, "Measuring P2P IPTV systems," in *Proc. 17th Int. Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV07)*, Jun. 2007.

[4] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree, a comparative study of live P2P streaming approaches," in *Proc. IEEE INFOCOM*, Anchorage, AK, May 2007, pp. 1424–1432.

[5] T. Small, B. Li, and B. Liang, "Outreach: Peer-to-peer topology construction towards minimized server bandwidth costs," *IEEE J. Select. Areas Commun.*, vol. 25, no. 1, pp. 35–45, Jan. 2007.

[6] E. Setton, J. Noh, and B. Girod, "Congestion-distortion optimized peer-to-peer video streaming," in *Proc. 2006 IEEE Int. Conf. Image Processing*, Oct. 2006, pp. 721–724.

[7] C. Y. Chan and J. Y. B. Lee, "A decentralized scheduler for distributed video streaming in a server-less video streaming system," in *Proc. 4th IEEE/ACM Int. Symp. Cluster Computing and the Grid*, Apr. 2004.

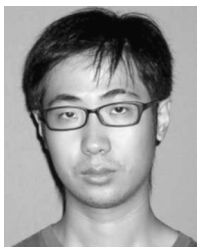
[8] H. Chi, Q. Zhang, J. Jia, and X. Shen, "Efficient search and scheduling in P2P-based media-on-demand streaming service," *IEEE J. Select. Areas Commun.*, vol. 25, no. 1, pp. 119–130, Jan. 2007.

[9] E. Setton, J. Noh, and B. Girod, "Low latency video streaming over peer-to-peer networks," in *Proc. IEEE Int. Conf. Multimedia & Expo (ICME)*, Toronto, ON, Canada, Jul. 9–12, 2006, pp. 569–572.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proc. 19th ACM Symp. Operating Systems Principles*, New York, Oct. 2003, pp. 298–313.

[11] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proc. 12th ACM Int. Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Miami Beach, FL, May 2002, pp. 177–186.

[12] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proc. 3rd USENIX Symp. Internet Technologies and Systems (USITS'01)*, 2001, pp. 49–60.
[13] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE J. Select. Areas Commun.*, vol. 20, no. 8, pp. 1456–1471, Oct. 2002.
[14] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. ACM SIGCOMM'02*, Aug. 2002, pp. 205–217.
[15] D. A. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," *IEEE J. Select. Areas Commun.*, vol. 22, no. 1, pp. 121–133, Jan. 2004.
[16] C. Wu, B. Li, and S. Zhao, "Multi-channel live P2P streaming: Refocusing on servers," in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008.
[17] F. Wang, J. Liu, and Y. Xiong, "Stable peers: Existence, importance, and application in peer-to-peer live video streaming," in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008.
[18] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A data-driven overlay network for live media streaming," in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005, pp. 2102–2111.
[19] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proc. 19th ACM Symp. Operating Systems Principles*, New York, Oct. 2003, pp. 282–297.
[20] GridMedia. [Online]. Available: http://www.gridmedia.com.cn/.
[21] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *Proc. 4th Int. Workshop Peer-to-Peer Systems*, Feb. 2005.
[22] D. Ren, Y.-T. H Li, and S.-H. G. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008.
[23] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal topology generation from a user's perspective," in *Proc. MASCOTS'01*, Jan. 2001.
[24] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proc. 4th ACM SIGCOMM Conf. Internet Measurement*, New York, 2004, pp. 41–54.
[25] E. Veloso, V. Almeida, W. M. , Jr, A. Bestavros, and S. Jin, "A hierarchical characterization of a live streaming media workload," *IEEE/ACM Trans. Netw.*, vol. 14, pp. 133–146, Feb. 2006.
[26] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. ACM SIGCOMM*, Portland, OR, Aug. 2004, pp. 107–120.

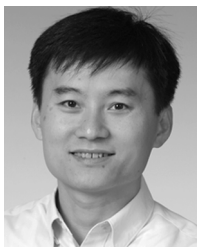


Dongni Ren received the B.Eng. degree in computer science from the Hong Kong University of Science and Technology (HKUST), Kowloon, in 2007. He is currently pursuing the M.Phil. degree at the Department of Computer Science and Engineering, HKUST.

His research interests include multimedia networking and peer-to-peer live streaming.

Yui-Tung Hillman Li received the B.Eng. degree in computer engineering from the Hong Kong University of Science and Technology, Kowloon, in 2006, where he is currently pursuing the M.Phil degree.

His research interests include overlay network, peer-to-peer streaming, and other technologies that enhance the effectiveness of peer-to-peer communication in the Internet.



S.-H. Gary Chan (S'89–M'98–SM'03) received the B.S.E. degree (highest honor) in electrical engineering from Princeton University, Princeton, NJ, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems; and the M.S.E. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1999, respectively, with a minor in business administration.

He is an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST), Kowloon. His research interest includes multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks.

Dr. Chan is an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA and a Vice-Chair of the Peer-to-Peer Networking and Communications Technical Sub-Committee, IEEE COMSOC Emerging Technologies Committee. He is the TPC chair of the IEEE Consumer Communications and Networking Conference (CCNC) in 2010. Since 2003, he has been serving as a Vice-Chair of IEEE COMSOC Multimedia Communications Technical Committee (MMTC). He was a guest editor of the special issue on "Peer-to-Peer Multimedia Streaming" in the IEEE COMMUNICATION MAGAZINE (2007) and "Advances in Consumer Communications and Networking" in Springer Multimedia Tools and Applications (2007). He was a co-chair of multimedia symposium in IEEE Globecom (2007 and 2006) and IEEE ICC (2007 and 2005), and of the workshop on "Advances in Peer-to-Peer Multimedia Streaming" in ACM Multimedia Conference (2005). He is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa. He was a Visiting Associate Professor at Stanford University (2008–2009), Visiting Research Collaborator at Princeton University (2009), Director of Computer Engineering Program at the HKUST (2006–2008), a Visiting Assistant Professor in networking at the Department of Computer Science, University of California at Davis, CA (1998–1999), and was a Research Intern at the NEC Research Institute, Princeton, NJ (1992–1993). He was a William and Leila Fellow at Stanford University in 1993–1994. At Princeton, he was the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence in 1993.