Minimum-Delay Overlay Multicast

Kianoosh Mokhtarian and Hans-Arno Jacobsen
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada

***Abstract*—Delivering delay-sensitive data to a group of receivers with minimum latency is a fundamental problem for various distributed applications. In this paper, we study multicast routing with minimum end-to-end delay to the receivers. The delay to each receiver in a multicast tree consist of the time that the data spends in overlay links as well as the latency incurred at each overlay node, which has to send out a piece of data several times over a finite-capacity network connection. The latter portion of the delay, which is proportional to the degree of nodes in the tree, can be a significant portion of the total delay as we show in the paper. Yet, it is often ignored or only partially addressed by previous multicast algorithms. We formulate the actual delay to the receivers in a multicast tree and consider minimizing the average and the maximum delay in the tree. We show the NP-hardness of these problems and prove that they cannot be approximated in polynomial time to within any reasonable approximation ratio. We then present a number of efficient algorithms to build a multicast tree in which the average or the maximum delay is minimized. These algorithms cover a wide range of overlay sizes for both versions of our problem. The effectiveness of our algorithms is demonstrated through comprehensive experiments on different real-world datasets, and using various overlay network models. The results confirm that our algorithms can achieve much lower delays (up to 60% less) and up to orders of magnitude faster running times (i.e., supporting larger scales) than previous minimum-delay multicast approaches.**

I. Introduction

Minimum-delay routing of data in overlay networks is a fundamental problem for several distributed applications. For instance, consider a delay-sensitive event notification system in which an event generated at a node needs to be signaled to a large group of monitoring nodes with minimum latency, e.g., a Distributed Interactive Simulation (DIS) software for military systems [1], [2], financial trading through large groups of globally-interlinked computer systems [3], [4], or massive multiplayer online games [5], [6]. Also note that the group of receivers corresponding to a source node in these systems may not be constant over time, such as a dynamic agent in a virtual environment (e.g., online game) moving across the Area of Interest (AOI) of other entities [7], [8].

Therefore, forming overlay multicast groups (which nodes should join and leave) in such dynamic systems and maintaining the corresponding state information in the intermediate overlay nodes, as in several classic multicast techniques [9]–[12], is not an efficient solution. A naive alternative approach is to send each message directly from the source to each receiver. This solution, however, is not scalable since it requires each node to have (and constantly monitor the state of) a connection to every other node in the network. Moreover, this approach can incur long delays, because a node has a finite-bandwidth connection to the network, over which several copies of the same data should be sent. To avoid these problems, nodes can be connected through a mesh overlay. Then, source-based minimum-delay multicast trees can be calculated on demand for transmitting the data, which is the problem of our interest in this paper.

While the primary goal of typical shortest-path multicast schemes has been to minimize the total link-by-link delay experienced by packets, we observe that a significant portion of the total delay in a multicast scenario is the delay incurred at overlay nodes. This is because each intermediate node has to send out several copies of the same packet through a single, finite-capacity network connection. This node-incurred delay, which is in proportion to the degree of the node in the routing tree, is in addition to the delay occurring in overlay links and can even dominate it as we show shortly. In particular, this issue becomes more critical in large-scale overlay networks with strong connectivity. In these networks, as intuitively expected, most shortest paths consist of a few hops only [13], [14], leading to large node degrees in a multicast tree.

To get a numeric intuition on this problem, suppose we would like to deliver one packet of 1.2 KB (10 Kbits) to 1000 nodes in an event notification overlay. Assume that overlay nodes have a 10 Mbps Internet connection and are on average handling data of 20 concurrent sessions; hence it takes 20 ms for a node to send out one copy of the packet. Also assume that the delay between every pair of nodes is 100 ms, and the average shortest path length is 3 hops in the overlay (i.e., a delay of 300 ms). Thus, the average degree of nodes in the multicast tree is about $1000^{1/3} \simeq 10$, and the average delay incurred by each node to forward the packet is $average_{i=1,\ldots,10}(i \times 20 \text{ ms})$=110 ms, i.e., a total node-incurred delay of 330 ms in a typical 3-hop path. Note that the total link-by-link delay of such path was only 300 ms.

Yet, the problem with delays incurred by node degrees in application-layer multicast is often ignored [15]–[19] or only partially addressed by previous works, such as bounding node degrees in a tree to predefined thresholds [20]–[24]. The problem with large node degrees, however, is of the same type as the shortest-path routing problem—minimizing the incurred delay. It thus needs to be considered together with link delays in the routing decisions, rather than as a separate problem and at the coarse grain of being or not being within a threshold.

In this paper, we study the overlay multicast routing problem

978-1-4673-5946-7/13/$31.00 ©2013 IEEE

for minimizing the actual end-to-end delay. In particular, the contributions of this paper are as follows. We first formulate the two problems of minimizing the average and the maximum delay in multicast trees, and we prove their NP-hardness as well as their inapproximability to within any reasonable ratio. That is, we show that no polynomial-time approximation algorithm can guarantee any better delay than several times beyond the optimal value. We then present a set of efficient algorithms for building multicast trees with minimum average (or minimum maximum) delay. These algorithms support a wide range of overlay sizes for both minimum-average and minimum-maximum applications. To demonstrate the effectiveness of these algorithms, we conduct an extensive evaluation study on different real-world datasets, and using three different overlay network models. We show that the actual delay observed by multicast receivers can be reduced by up to 60%, and the calculation time for multicast trees by up to orders of magnitude (i.e., supporting several times larger scales), if our algorithms are employed.

In the remainder of this paper, we first summarize the related work in Section II. We then formally define our multicast problems in Section III and review the routing model on which our algorithm is built. Section IV presents our algorithms, followed by a thorough evaluation study in Section V. We conclude the paper in Section VI.

II. Related Work

The problem of our interest in this paper is to deliver data with minimum end-to-end delay from a given source to a set of receivers on a mesh overlay. Compared to arranging nodes in fixed overlay multicast trees [20], [23], calculating on-demand per-source trees on a well connected mesh overlay provides much higher flexibility in selecting paths and better resilience against dynamics of the network [11], [16]. Moreover, as discussed earlier we consider source-based multicasting, in which the intermediate nodes do not need to keep any per-session state information or to perform route calculations for each message. Instead, the source node calculates the routing tree and embeds it in the message (e.g., using Bloom Filters [25]), and the intermediate nodes only perform simple forwarding.

Most overlay multicast algorithms only minimize the link-by-link distance to the receivers [15]–[19]. However, as discussed earlier, a significant factor in the actual end-to-end delay is the delay incurred at overlay nodes that send out each message several times. There have been a number of previous works that did consider the node degree problem [21], [22], [24], [26], [27], but they only try to find a routing tree in which the degree of nodes is bounded to predefined thresholds. Ito et al. [27] analyzed several forms of the multicast tree problem, and showed that it is NP-Complete to find various forms of degree-bounded trees, such as one with minimum total distance or one with minimum distance to the farthest receiver. Heuristic algorithms for constructing degree-bounded shortest path trees [22], [26] and degree-bounded minimum-diameter spanning trees [21], [28] have been proposed. However, it is not clear how these degree bounds are selected in practice; for instance, a fixed bound of 10 is used in [28]. In [24] it is proposed to set the degree bounds based on the level of the node in the tree. Nevertheless, these works only aim at bounding node degrees to given thresholds. Rather than bounding degrees at such coarse grain, we capture the delay caused by node degrees together with the delay incurred at overlay links as a single delay cost and minimize it.

The only previous work considering this problem, to the best of our knowledge, is done by Brosh et al. [29] who propose an approximation and a heuristic algorithm for minimizing the maximum delay in a multicast tree (the problem with minimum-average delay is not considered). However, the proposed approximation algorithm and its bound only correspond to the special case of *broadcasting* a message in a *complete* overlay graph, whereas often in practice neither the overlay is a complete graph (i.e., every node maintains and monitors a connection with every other node) nor all messages are destined to all nodes. Furthermore, even for this special case the approximation factor is $O(\log n)$ and $O(\log n / \log\log n)$ for directed and undirected overlay graphs, respectively, which is a considerable amount. In fact, the heuristic algorithm proposed by the authors (with no approximation guarantee) provides lower delays than the approximation algorithm while being also more efficient in running time [29]. Nevertheless, the achieved delay and the running time of this algorithm are significantly larger than our algorithms.

We also note that an important factor determining the scalability of a multicast scheme is the underlying routing protocol. The common approach used in [29] and several other works is link-state based routing [12], [13], [15], [18], [19], [25], which allows all nodes to know the full topology of the network while suffering from high overhead and limited scalability (as we will show). Our multicast scheme, on the other hand, is based on a variant of distance-vector routing and can be up to orders of magnitude more scalable.

III. System Model and Problem Statement

This section presents the formal statement of our multicast problems and their hardness, followed by a description of the routing model underlying our algorithms.

A. Notation

A summary of the notations used in this paper is given in Table I. Consider an overlay network interconnecting a large population of distributed hosts. The overlay is modeled by a graph $G = (V, E)$ in which each vertex $v \in V$ represents a host and each edge $(u, v) \in E$ represents a link between two connected hosts. Let $N = |V|$, and $w(u, v)$ be the length of edge (u, v), which is the network delay between nodes u and v in our application. We assume each overlay node is connected to the underlying network, typically the Internet, via one link (nodes are assumed not multi-homed; also a node with multiple NICs connected to the same network can be modeled as having one NIC with the aggregated bandwidth). The connection bandwidth of each node is a finite number,

TABLE I
NOTATIONS USED IN THIS PAPER.

Notation	Description
G,V,E,N	Overlay graph $G=(V,E)$. $N=\|V\|$.
G_{PV}	Partial view of overlay graph based on path vectors.
T, s, R	The routing tree, the source node, and the set of receiver nodes, respectively.
z	Size of the message being distributed.
$\Delta_u(z)$	Time it takes for node u to send out a message of size z.
$w(u,v)$	Delay of overlay link (u,v).
d_T,$d_G(u)$	Degree of node u in tree T or graph G.
$q_{u,T}(v)$	The *turn* of v among the children of u in T.
$t_T(u)$	Time at which node u receives the message over tree T.
$g_T(u)$	Number of receiver nodes in the subtree rooted at u in tree T (including u itself).
$h_T(u)$	The delay from v to its farthest descendant.
D, D_{max}	Average and maximum node degree in the overlay.
L, L_{max}	Average and maximum hop count of pairwise shortest paths in the overlay.

according to which we define $\Delta_u(z)$ as the time it takes for node u to send z units of data to the network. Since the time for node u to process a message is much smaller than the time it takes to send out (possibly multiple copies of) the message to the network, we ignore the processing time at u and let $\Delta_u(\cdot)$ be a function of the connection bandwidth of u only. For example, for a host with a 10 Mbps Internet connection and a message of $z=1$ bit, $\Delta_u(z)=10^{-7}$ seconds.

We capture the node degree-incurred delays (also referred to as nodal delays) in a multicast tree as follows. Let T denote an arbitrary multicast tree rooted at a given source s and reaching the receiver set $R \subseteq V$, and $d_T(u)$ be the number of children of u in T. Once s started distributing a message of size z at time 0, $t_T(v)$ is the time at which node v receives the message over T. Assuming node u is the parent of v in T, we have:

$$t_T(s)=0;\ t_T(v)=t_T(u)+w(u,v)+\Delta_u(z)\times q_{u,T}(v), \quad (1)$$

where $q_{u,T}(v)$ ($1 \le q_{u,T}(v) \le d_T(u)$) shows the *turn* of node v among the $d_T(u)$ children of u in T that are waiting to receive a copy of the message.

Because in some cases we may not be able to explicitly dictate an order among the children of u in T, we also take the *expected* delay observed at each child of u into account. We define $\hat{t}_T(v)$ as in the following equation, by replacing $q_{u,T}(v)$ in Eq. (1) with $E[q_{u,T}(v)]$, the average of possible turns between 1 and $d_T(u)$ for a child.

$$E[q_{u,T}(v)]=\frac{1}{d_T(u)}\sum_{i=1}^{d_T(u)} i=(d_T(u)+1)/2$$

$$\hat{t}_T(s)=0;\ \hat{t}_T(v)=\hat{t}_T(u)+w(u,v)+\Delta_u(z)\frac{d_T(u)+1}{2}. \quad (2)$$

B. Formal Definition of the Problems

We consider two versions of the minimum delay multicast problem: (i) constructing a multicast tree in which the average delay (equivalently, the total delay) is minimized, and (ii) a tree in which the delay to the farthest node is minimized. We propose algorithms for both versions of the problem.

Our algorithms for case (i) are not based on encoding the tree structure in full in the data; they do not need to specify a *forwarding order* for each intermediate tree node, and can work with $\hat{t}_T(v)$ values defined in Eq. (2). This is preferred, as it enables the use of mechanisms such as Bloom Filters for encoding the tree [25]. The algorithms for case (ii), on the other hand, have to specify a forwarding order for the children of each node (and therefore work with $t_T(v)$ values defined in Eq. (1)), since the maximum delay in a tree depends to a large extent on the ordering of each node's children.

Problem 1: Given a source node s and a set of receivers $R \subseteq V$, construct a multicast tree T such that $\sum_{u\in R}\hat{t}_T(u)$ is minimized.

Problem 2: With the same inputs as in Problem 1, construct a multicast tree T such that $\max_{u\in R} t_T(u)$ is minimized.

Theorem 1 shows the NP-hardness as well as an inapproximability factor for Problems 1 and 2. The proofs are omitted due to space limitations and can be found in [30].

Theorem 1: Problems 1 and 2 are NP-hard, and also no polynomial time approximation algorithm for either of them can guarantee an approximation factor of $(1-\epsilon)\ln n$ for any $\epsilon > 0$ (under the conventional assumptions for P and NP).

C. Routing Model

To enable a fully distributed routing scheme, overlay nodes need to generate and exchange periodic routing information. We adopt a modified version of the distance-vector routing protocol, where each node announces its shortest distance to each destination, as well as the path itself; in our case the nodes also announce their $\Delta_u(\cdot)$ values along with this information. This is similar to the technique employed in the BGP protocol and is usually referred to as path-vector routing. This approach allows each node u to construct a graph $G^u_{PV}=(V^u_{PV}, E^u_{PV})$ where $V^u_{PV}=V$, and E^u_{PV} consists of only a *representative* subset of E for u: the edges on the shortest path, as well as up to $d_G(u)-1$ alternative short paths, from u to all destinations in the graph. A brief comparison of the overhead and other desirable features of path-vector routing compared to the two alternatives, distance-vector and link-state routing (used in most overlay routing schemes), is given in our extended technical report [30].

IV. Minimum-Delay Multicast Algorithms

In this section, we first provide an overview of our minimum-delay multicast tree algorithms, and then present the details as well as the analysis of the algorithms.

A. Overview of the Algorithms

Our algorithms include two operation modes: MinSum for minimizing the expected total delay (Problem 1), and MinMax for minimizing the maximum delay in the tree (Problem 2). We refer to these algorithms as **MSDOM** and **MMDOM** (MinSum/MinMax Delay Overlay Multicast). These algorithms

outperform the previous related approaches in both multicast tree efficiency and running time, as analyzed in the next section. Nevertheless, to further extend the application of our work to larger overlays, we design an additional algorithm for each operation mode (MinSum/MinMax) that is optimized for speed, with orders of magnitude faster running times. These algorithms are particularly suitable for large overlays where our former algorithms (and the related previous approaches) cannot operate fast enough. We refer to the former (delay-efficient) algorithms as MSDOM/MMDOM-*e*, and to the latter (fast) version as MSDOM/MMDOM-*f* algorithms.

Each overlay node that is to distribute a message runs the relevant version of the algorithm based on the target applications. The input to the algorithm consists of the state of the node's links to its neighbors, as well as the path vector information that the node has received from its neighbors. The resulting multicast tree is then encoded in the message. The intermediate overlay nodes are free of any additional computations or keeping any per-session multicast state information—they only forward the data to their neighbors according to the information embedded in it.

The multicast tree calculated by the source node can either be encoded in full in the message (with an overhead of $O(N)$), or be encoded as a digest, e.g., using fixed-size Bloom Filters that can significantly reduce the overhead with negligible false positives [25]. In this paper we only study the calculation of minimum-delay multicast trees, and the detailed encoding of the tree is out of the scope of the paper. Nevertheless, we highlight that the MSDOM algorithms do not need to specify the full tree structure for the intermediate forwarding nodes—they allow the use of Bloom Filter digests. The MMDOM algorithms, on the other hand, requires to signal the full tree structure (as in the similar work in [29]), because a forwarding order among the children of each node needs to be specified so that the forecasted maximum delay can be actually met in the network; see $q_{u,T}(v)$ in Eq. (1).

The routing table underlying our algorithms is a $d_G(s) \times N$ matrix at each node s (based on which the G_{PV} view is created at s); see Table I and Section III-C. An entry (i, j) of this matrix represents the shortest path to overlay node j through the i-th neighbor of s. Row i of this matrix is maintained over time according to the path vector information that the node receives from the corresponding neighbor. Each path vector entry from this neighbor represents the shortest path of this neighbor to some overlay node: the path's hops and hop-by-hop distances (i.e., $O(L_{max})$). Moreover, a min-heap is maintained on each column of the table, to quickly return the best neighbor for reaching each destination. The complexity of maintaining the routing table is analyzed shortly.

B. Detailed Multicast Algorithms

Our multicast algorithms are specified in Figures 1 and 2. In the MSDOM-*e* algorithm, we start from the source s and build the multicast tree by incrementally adding nodes according to their distances to the current tree. More specifically, for any potential attachment (u, v) to the tree T where $u \in T$ and

Minimum End-to-end Delay Overlay Multicast

MSDOM_e()†

1. $T = \mathit{BuildEmptyTree}(s)$; $t[i = 1, \ldots, N] = \infty$; $t[s] = 0$
2. **while** $R \neq \emptyset$ **do**
3. $\quad \mathit{cost}[] = \emptyset$; $\mathit{prev}[] = \emptyset$ // To find the best node to attach to T
4. $\quad$ **for** v in $V - T$ **do**
5. $\quad\quad \mathit{cost}'[] = \emptyset$ // To find the best attachment point for v
6. $\quad\quad$ **for** u in $\mathit{neighbors}(v, G_{PV})$ s.t. $u \in T$ **do**
7. $\quad\quad\quad$ // Cost of attaching v to T through u:
8. $\quad\quad\quad \mathit{cost}'[u] = t[v] + \tilde{w}(u, v) + (d_T(u) + 2)/2 \times \Delta_u(z)$
9. $\quad\quad\quad \mathit{cost}'[u] \mathrel{+}= 1/2 \times \Delta_u(z) \times (\mathit{SubTreeSize}_T(u) - 1)$
10. $\quad\quad u^* = \mathit{argmin}(\mathit{cost}')$ // Best attachment point for v
11. $\quad\quad$ **if** $u^* ==$ *NULL* **then continue**
12. $\quad\quad \mathit{cost}[v] = \mathit{cost}'[u^*]$
13. $\quad\quad \mathit{prev}[v] = u^*$
14. $\quad v^* = \mathit{argmin}(\mathit{cost})$; $u^* = \mathit{prev}[v^*]$
15. $\quad \mathit{AttachToTree}(T, v^*, u^*)$
16. $\quad$ **if** $v^* \in R$ **then** $R.\mathit{remove}(v^*)$
17. $\quad \mathit{Update\_t}(T, u^*, t[])$
18. $\mathit{CleanUp}(T, s)$
19. **return** T

MMDOM_e()†

1. $T = \mathit{BuildEmptyTree}(s)$; $t[i = 1, \ldots, N] = \infty$; $t[s] = 0$
2. **while** $R \neq \emptyset$ **do**
3. $\quad$ **for** u in V **do**
4. $\quad\quad \mathit{prev}[u] = -1$
5. $\quad\quad \mathit{dist}[u] = t[u] + d_T(u) \times \Delta_u(z)$
6. $\quad S = V$
7. $\quad$ **while** $S \neq \emptyset$ **do**
8. $\quad\quad u^* = \mathit{argmin}_{u \in S}(\mathit{dist}[u])$
9. $\quad\quad S.\mathit{remove}(u^*)$
10. $\quad\quad$ **for** v in $\mathit{neighbors}(u^*, G_{PV})$ s.t. $v \notin T$ **do**
11. $\quad\quad\quad d = \mathit{dist}[u^*] + \Delta_{u^*}(z) + \tilde{w}(u^*, v)$
12. $\quad\quad\quad$ **if** $d < \mathit{dist}[v]$ **then**
13. $\quad\quad\quad\quad \mathit{dist}[v] = d$
14. $\quad\quad\quad\quad \mathit{prev}[v] = u^*$
15. $\quad$ // Attach the farthest node to T:
16. $\quad v^* = \mathit{argmax}_{v \in R \setminus T}(\mathit{dist}[v])$
17. $\quad H =$ List containing hops of path to v^* according to $\mathit{prev}[]$
18. $\quad$ **for** v from $H.\mathit{FirstNodeNotInTree}(T)$ to $H.\mathit{last}()$ **do**
19. $\quad\quad u = \mathit{prev}[v]$ // Parent of the to-be-attached node v in T
20. $\quad\quad \mathit{AttachToTree}(T, v, u)$
21. $\quad\quad t[v] = \mathit{dist}[v]$
22. $\quad\quad$ **if** $v \in R$ **then** $R.\mathit{remove}(v)$
23. **return** T

† Variables used in the code are described in Table I.

Fig. 1. The proposed overlay multicast algorithm.

$v \notin T$ (Lines 4 and 6), we find the cost as the increase in the expected total delay to all nodes caused by this attachment (Lines 8 and 9). This delay consists of the expected delay to node v itself, as well as the expected delay to be suffered from by other descendants of u since the degree of u is going to increase (see Eqs. 1 and 2). Having applied the minimum-cost attachment (u^*, v^*) (Lines 14 and 15), we update the current distance value (array t) of all affected nodes (Line 17), i.e., the children and all further descendants of node u^*. Finally, we clean up the tree (Line 18) by keeping only the paths that end at some receiver node.

The MMDOM-e algorithm repeatedly runs our modified version of the Dijkstra algorithm to find the farthest node from the current tree T. This modified shortest-path algorithm (Lines 3 to 14) can start from multiple nodes (see the initialization of *dist*[] in Line 5), considers the current degree of nodes in the existing tree (Line 5), and expands based on aggregated link and nodal delays given the current tree (Line 11). Having found the farthest node v^* from the tree, v^* as well as its predecessors on the path starting from some node in T (i.e., nodes in list H in Line 17) are added to the tree. Note that after this addition, the degree of a number of nodes changes, making the recently calculated shortest-path distances no longer accurate. Thus, the next farthest node is searched for again in the next iteration.

The running time of the MinSum and MinMax algorithms presented so far, as we analyze shortly, is $O(N^2 D_{PV})$ where D_{PV} is the average degree in the path-vector based view of the overlay graph. This running time may not be efficient enough for large overlays, as quantified in the next section. We therefore develop additional algorithms, MSDOM/MMDOM-f, which are optimized for speed.

In these algorithms, which are illustrated in Figure 2, we first calculate the regular shortest-path tree from s to the receivers. This is simply done by merging the shortest paths to the destinations given in the path-vector routing table (an example on the MSDOM/MMDOM-f algorithms and routing tables can be found in the extended technical report [30]). This tree is then refined according to the given objective: minimizing the total delay or the maximum delay.

In the MSDOM-f algorithm, the tree is refined in a top-down manner from the root downwards. For each node u, we look at each of its children v and consider routing to it through an alternative route (Lines 5 and 6 of function RefineTree[MinSum]). The alternatives for a node v are to route to v through any other of the $d_G(s)$ neighbors of the root s than the one currently used to reach v. Given the path vector information available at s, we evaluate the possible alternative routes, and change the current route to v if necessary (Lines 7 to 10). To refine the route to the children of node u, we first consider those children that have the highest number of receivers in their subtrees (the sort operation in Line 2), since any saving in the delay to those children will likely yield a higher overall saving. Once finished relaxing the degree of node u, the algorithm proceeds with the next level which is refining the subtree of each child of u.

For the MMDOM-f algorithm, we first note that the maximum delay in the subtree rooted at a node u can vary much based on the ordering of u's children to receive the message (see $q_{u,T}(v)$ in Eq. 1). Thus, we need to obtain an optimal ordering for the children of u. Denoting by $h_T(v)$ the delay from v to its farthest descendant, the optimal ordering of u's children for minimizing the maximum delay corresponds to sorting the children in descending order of their $h_T(v)$ values. This is done for all nodes of the tree in function T.*FixForMinMax* (Line 1; similarly done later for any affected node in Line 13). In each iteration, the algorithm picks the node with maximum delay (Line 3), and tries to find an alternative shorter route to the node (Lines 5 and 6).

Minimum End-to-end Delay Overlay Multicast

MSDOM_MMDOM_f(*mode*)†
// *mode*: MinSum or MinMax.
1. $T =$ *BuildRegularShortestPathTree*(s)
2. *RefineTree*$[mode](T, s)$
3. **return** T

RefineTree[MinSum](T, u)
1. **if** u.*IsLeaf*() **then return**
2. $C[]$ = u.*children*() sorted in descending order of $g_T(C[i])$
3. **for** v in $C[]$ **do**
4. $\quad t'[] = \emptyset$
5. $\quad$ **for** a in T.*root.neighbors*() **do**
6. $\quad\quad t'[a] =$ *SavingByRouteChange*(T, v, a)
7. $\quad a^* =$ *argmax*$\{t'[]\}$
8. $\quad$ **if** $t'[a^*] > 0$ **then**
9. $\quad\quad T$.*DeleteRoute(v)* // Detach v and its subtree
10. $\quad\quad T$.*InsertRoute*(v, a^*)
11. Sort $C[]$ again in descending order of $g_T(C[i])$
12. **for** v in $C[]$ **do**
13. $\quad$ *RefineTree*(T, v)

RefineTree[MinMax](T, u)
1. T.*FixForMinMax*()
2. **for** $i = 1$ to *MAX_REFININGS* **do**
3. $\quad v = T$.*FarthestLeaf*()
4. $\quad t'[] = \emptyset$
5. $\quad$ **for** a in T.*root.neighbors*() **do**
6. $\quad\quad t'[a] =$ *SavingByRouteChange*(T, v, a)
7. $\quad a^* =$ *argmax*$\{t'[]\}$
8. $\quad$ **if** $t'[a^*] \leq 0$ **then break** // No more refinings
9. $\quad T$.*DeleteRoute(v)* // Detach v and its subtree
10. $\quad T$.*InsertRoute*(v, a^*)
11. $\quad$ **while** $v \neq T$.*root* **do**
12. $\quad\quad v = v$.*parent*
13. $\quad\quad v$.*FixChildrenForMinMax*()
14. $\quad T$.*FixNodesDS*()

† Variables used in the code are described in Table I.

Fig. 2. The proposed routing algorithm.

The algorithm terminates if the tree reaches a state where the distance to the farthest node cannot be shortened (Line 8), or if the total number of refining steps reaches a bound (MAX_REFININGS in the code). We set this bound to $N/\log N)$ to keep the worst-case running time to $O(N^2)$ (analyzed shortly), although in our experiments the algorithm has terminated much earlier than this bound. To enable efficient reordering of each node's children and retrieval of the farthest node in the tree, at each node v we maintain certain information including $t_T(v)$ and $h_T(v)$. After each modification in the tree, the information maintained at up to $O(N)$ nodes may need to be updated. For example, after moving node v from its old parent u to a new parent u', the delay to each descendant of v, the correct ordering of the children of u' and those of u''s ancestors, and accordingly the delay to each descendant of u' and u''s ancestors needs to be updated. Therefore, we simply update the data structures for all nodes of the tree in $O(N)$ time (Line 14 of the pseudocode).

C. Analysis

A summary of the memory requirement and running time of our algorithms is as follows. The total memory space for maintaining the required data structures at a node and running our algorithms is on average $O(NDL_{max})$, where L_{max} is the maximum hop count in a shortest path (i.e., the hop count of the overlay diameter). Moreover, the time taken by a node to update the routing table, upon receiving a path vector of size $O(NL_{max})$ from a neighbor, is of $O(N(L_{max} + \log N))$. The running time of both MSDOM-*e* and MMDOM-*e* algorithms is bounded by $O(N^2 D_{PV})$. The MSDOM-*f* algorithm runs in $O(NDL_{max})$ time. The MMDOM-*f* algorithm takes $O(N^2)$ time, given that we bound the number of its refining steps (each taking $O(N \log N)$ time) to $N/\log N$ as mentioned earlier; though we also note that the number of refining steps taken by this algorithm in our experiments has been far less than $N/\log N$. Additional details on our complexity analysis can be found in the extended technical report [30].

V. Evaluation

We evaluate the performance of our algorithms on hundreds of overlay networks built on top of two different real-world Internet delay datasets, and according to three different overlay graph models. Our evaluation setup and the obtained results are presented in this section.

A. Setup

To capture the actual delay between hosts in the Internet, which is a key factor determining the effectiveness of any overlay multicast algorithm, we use the data from two different measurements: the Delay Space Synthesizer (**DS**^{2}) project [31] which provides the measured pairwise delay between about 4000 Internet hosts, and the **Meridian** project [32] containing the delay between 2500 hosts. We sometimes need to down-sample the set of 4000 (or 2500) nodes to $N < 4000$ nodes in our experiments. To ensure having a representative subset, we use a variant of the k-means clustering algorithm [33] that takes the best N center points among the original 4000 (or 2500); it minimizes the sum of the squared distance between each original point and its closest center point.

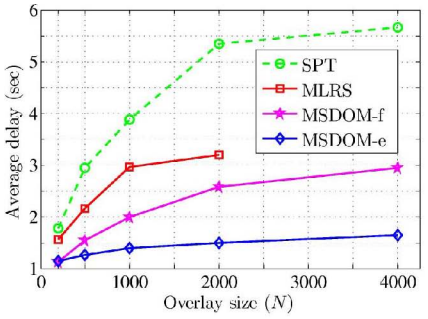
On top of these datasets, we create overlay networks based on three different overlay graph models: **small world** [34], **random** [35], and **power law** [36]. In small-world networks, links are more likely to exist between nearby nodes than distant nodes [34]. These networks are commonly observed in social and P2P networks [37]; they have also yielded the smallest shortest-path length between nodes in our experiments. We generate these networks by connecting each pair of nodes (u, v) with probability $\alpha \times \mathit{distance}(u, v)^{-1}$, where the coefficient α is set according to the desired average node degree D in the overlay. On the other hand, random networks are simpler in which all edges of different lengths are treated similarly. Specifically, we generate random networks according to the Erdos-Renyi model [35] where each pair of nodes (u, v) exists with probability p, which we set according to the desired average degree D. In a power-law network (also called a scale-free network), such as the worldwide web, nodes that are popular are more likely to receive new links; the network therefore has a power-law degree distribution [36]. We generate these networks following the Barabasi-Albert model [36], where a new node x connects to an existing node u with probability $\alpha d_G(u)/\sum_{v \in V} d_G(v)$ (α determining the average degree).

The parameters that we vary in our evaluations include the overlay size (N), average node degree (D), number of receivers ($|R|$), and node-incurred delays (including their dynamics). Specifically, we define $\bar{\Delta}$ as the average time that it takes for a node to send out each copy of the message. To get a sense of various combinations of node bandwidth and workload as well as message size (which are the factors governing the nodal delay), we consider a diverse range of values for $\bar{\Delta}$ from 10 ms to 1000 ms in our experiments, assumed the same ($\pm 50\%$) for all nodes. For example, $\bar{\Delta} = 10$ ms can represent the case where the message is a 10 Kbits (1.2 KB) IP packet, and each node has a 10 Mbps Internet connection and is handling 10 concurrent multicast sessions on average; we also analyze the dynamics of nodal delays in our evaluations. Furthermore, to consider the delivery of a continuous flow of data, we note that each node would send the data in chunks to its children in the multicast tree. Assuming TCP transport, to maximize the throughput, the node sends a continuous chunk of up to a full TCP window to its first child, then to the second child, and so on. The message size z can therefore be assumed up to the maximum window size, e.g., 128 KB (1 Mbits) as it is the default value in most Linux distributions. Thus, $\bar{\Delta} = 1000$ ms can represent the multicast of a continuous flow on the aforementioned network.

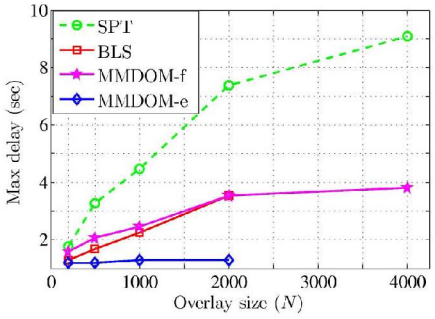
Given an overlay network of N nodes and a number of receivers $|R|$, in each run we select a sender and $|R|$ receivers at random from $[1, N]$. We then generate multicast trees using ours as well as previous approaches (described shortly) on the same input: G_{PV} view of the overlay, s and R. We repeat each experiment 100 times: on 10 different overlays, and 10 different sender/receivers selection on each overlay. We then measure the average delay and running time obtained in the 100 cases. Finally, since in the MinSum algorithms a source does not transmit the full (ordered) tree structure (see Section III), we simulate this lack of knowledge by shuffling the children of each node in the created tree before evaluating the average delay to the receivers.

B. Results

We evaluate the MSDOM algorithms by comparing the achieved average delay with the average delay in the regular shortest-path tree (**SPT**) for the same overlay and receiver set. We also compare these results with the delay achieved by the algorithm in [26], which builds a MinSum-delay multicast tree with bounded degrees—the closest work in the literature to our MinSum algorithms, to the best of our knowledge. We run a binary search to find the best degree bound value for this algorithm, though in our running time measurements presented shortly we only measure the time taken by one run of this



(a) MinSum-delay trees.



(b) MinMax-delay trees.

Fig. 3. Average and maximum delay of multicast trees created by different algorithms.

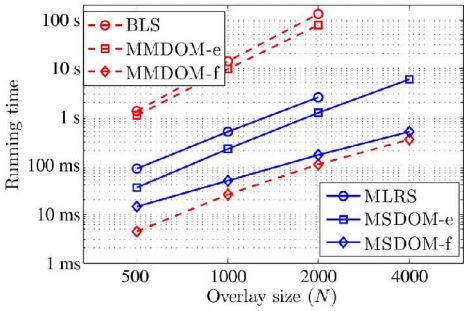


Fig. 4. Running time of the algorithms in Figure 3.

algorithm (not the multiple times done in the binary search). This algorithm is labeled as "**MLRS**" in the following figures (taken from the names of the authors).

We analyze the MMDOM algorithm by comparing the maximum delay in our tree with the maximum delay in the SPT as well as the tree created by the algorithm in [29] for the same overlay and receiver set. As discussed in Section II, the algorithm in [29] is the only previous study on minimizing the joint link and node incurred delays in a multicast tree, to the best of our knowledge. We only consider the heuristic algorithm proposed by the authors, since it outperforms the proposed alternative approximation algorithm (see Section II) in both tree-efficiency and running time [29]. We also note that this work addresses only the MinMax version of our problem. This algorithm is label as "**BLS**" in the following figures.

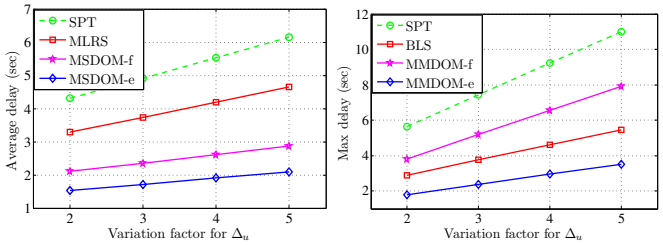
We first evaluate the algorithms on overlays of different sizes, and present the results in Figure 3. We also report the time taken by these algorithms in Figure 4. Our experiments are run on a commodity PC with Intel Core i5 2.67 GHz CPU with 8 GB of memory. The following experiment (Figures 3 and 4) is conducted on the DS^2 dataset, using the small-world overlay model, with $R = N-1$, $D = N/10$ and $\bar{\Delta} = 100$ ms. The cases with $N = 4000$ are skipped for MLRS, BLS, and MMDOM-*e* algorithms for their running times. Also, we do not report a running time for SPT in Figure 4 as it takes a very short time given the up-to-date routing table—only merging the given shortest paths.

It is noteworthy that one might add the running time to the delay achieved by the tree, since a message is not sent out until the tree calculation is done. However, we also note that each calculated tree is typically expected to be used for a number of messages. For instance, back to the example of the dynamic online agent (the source) moving across the area-of-interest of others (the receivers) in a virtual environment, we can realistically expect that the receiver group changes at a quite slower pace (in the order of seconds) than the rate of disseminating messages (several per second). We therefore separately report both times, tree delay and running time.

In Figure 3, we first observe that the commonly used shortest-path trees, despite being fast to build, suffer from large average and maximum delays as they are unaware of nodal delays. We also observe in Figure 3(a) that both of our algorithms outperform the previous related work MLRS [26]. MSDOM-*e* trees can provide an average delay as low as half the MLRS ones, while also being created multiple times faster (note the log-log scale in Figure 4). Our fast algorithm MSDOM-*f*, is several times faster (taking 490 ms for $N = 4000$) while still providing better average delays than MLRS.

Similarly for the MinMax version in Figure 3(b), the MMDOM-*e* algorithm yields the smallest farthest-node delays (< 1.3 s), significantly less (8–64%) than that of the related previous work BLS [29] which is up to 4 s. We also subtly note that in a few cases the maximum delay achieved by MMDOM-*f* is even slightly better that the average delay achieved by MSDOM-*f*, which is because the MinMax algorithms have the advantage of transmitting the ordered tree structure while the MinSum ones do not (see Sections III-B and V-A). Furthermore, we observe in Figure 3(b) that the maximum delay achieved by MMDOM-*f* and the one by BLS are relatively close, with BLS being slightly better in some cases. On the other hand, the running time of MMDOM-*f* is *2 to 3 orders of magnitude* smaller. Thus, having the MMDOM-*e* algorithm for overlay sizes of up to hundreds, and MMDOM-*f* for larger overlays, we can create multicast trees with much smaller MinMax delays (less than half) and/or faster running times (orders of magnitude) than the alternative approach BLS.

Next, we analyze our algorithms in the following experiments: (i) multicast with different levels of network connectivity, by varying the average node degree D in [50, 400] (with $N = 1000$, $|R| = 999$, $\bar{\Delta} = 100$ ms), (ii) multicasting to various numbers of receivers (varying $|R|$ in [250, 999]), and (iii) multicast with different nodal delays (varying $\bar{\Delta}$ in [10 ms, 1000 ms]). A summary of these experiments is as follows. In all cases, the same trend as in the previous experiment is observed among the achieved delay and running time of the algorithms: both MSDOM-*e* and MMDOM-*e* algorithms achieve a smaller average/maximum delay (up to 60%) than the related works (MLRS/BLS) while still running faster. MSDOM-*f* trees too provide smaller average delays (up to 35% less) than the related approach MLRS, while also being created up to 20 times faster. For the MinMax version, similar to the previous experiment, the farthest-node delay in



(a) MinSum-delay trees. (b) MinMax-delay trees.

Fig. 5. Dynamics of nodal delays (Δ_u).

MMDOM-*f* and BLS trees are close, while the former one is created 2 to 3 orders of magnitude faster (5–100 ms vs. 13+ seconds in different experiments). Finally, regular shortest-path trees suffer from high average/maximum delays in all cases: up to 5 times higher delay than that of our algorithms. Additional details on these experiments including figures and numerical analyses can be found in [30].

In an actual system, the nodal delays ($\Delta_u(\cdot)$), just like the round-trip delay between nodes (i.e., link delays), are not static values. Although each node u announces its Δ_u value (short for $\Delta_u(1)$, to be precise) as a representative average over time, e.g., EWMA averaged, the momentary nodal delay of u at the time of forwarding our message may be considerably different than the announced average. Neither ours nor previous algorithms are specifically designed to capture the uncertainty of these delays. Nevertheless, to have a thorough evaluation study, we analyze the impact of Δ_u dynamics on the efficiency of the different multicast trees. Therefore, right before evaluating a created tree, we change the Δ_u value of each node u to have a random increase or decrease by *multiple* times: denoting the *variation factor* of a nodal delay by v ($v = 1, 2, \dots$), we change each Δ_u to a randomly selected value in $[\Delta_u/v, \Delta_u v]$; that is, Δ_u is multiplied by e^x where $x \sim U(-\ln v, \ln v)$. Figure 5 shows the impact of these dynamics. As expected, the average and maximum delay of all trees rise by the increased fluctuation of nodal delays; we should also note that this is partially because the average nodal delay in the network increases by 8%, 20%, 35%, and 50% for $v = 2, \dots, 5$, respectively. We observe that with varying Δ_u dynamics, the performance of the algorithms relative to each other remains more or less the same, with MSDOM/MMDOM-*e* algorithms always resulting in the lowest-delay trees.

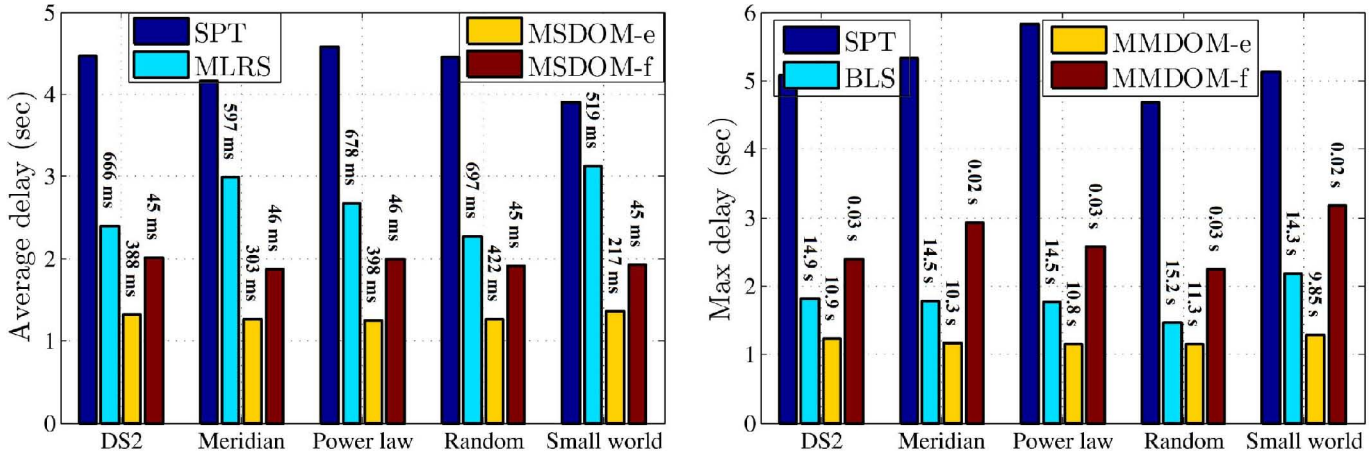
We also evaluate our algorithms on both datasets DS^2 and Meridian, and using all the three network models of small world, random, and power law. The results, including tree delays and running times, are presented in Figure 6. The running times for SPT are omitted as they are negligible—few milliseconds to only merge the shortest paths given in the routing table. These experiments are conducted on overlays of size $N = 1000$ with $R = 999$, $D = 100$, and $\bar{\Delta} = 100$ ms. The experiments for each of the two datasets, represented by the first two sets of bars in Figures 6(a) and 6(b), are conducted on all the three overlay models and the results are averaged. Similarly, the experiments for each overlay model, represented by the last three sets of bars in each figure, are run on both datasets and the average is plotted. We observe in these figures that the different algorithms perform more or less the same as in the previous experiments: MSDOM/MMDOM-*e* algorithms produce the most delay-efficient trees (45–60% smaller delays than MLRS and 20–40% than BLS) while also running slightly faster than the previous approaches; note the running times written on top of the bars. MSDOM-*f* also has better tree efficiency (15–40%) and running time (over an order of magnitude) than MLRS. MMDOM-*f* runs in almost zero time for the same inputs on which MMDOM-*e* and BLS algorithms have taken 10+ seconds, while still creating trees with reasonable delay-efficiency—compare to SPT which is the only relevant alternative according to running times.

Summary. We have observed that our algorithms together can support a wide range of overlay sizes for both MinSum and MinMax versions. The MSDOM/MMDOM-*e* algorithms outperform previous approaches in both tree efficiency and running time, and are the best choice for overlays of 100x nodes. For larger overlays, where previous approaches as well as MSDOM/MMDOM-*e* are not feasible, MSDOM/MMDOM-*f* algorithms are suggested which can run fast (100–160 ms for $N = 2000$ and 340–490 ms for $N = 4000$ in our experiments in Figure 4) while producing trees with reasonable delay efficiencies: 40–55% smaller delays than SPT trees which are the only applicable alternative.

VI. Conclusions

We have studied the problem of delivering data from a source to a group of receivers with minimum end-to-end delay in an overlay network. We show that multicast routing algorithms that simply find a shortest-path tree can result in large delays as they only minimize the link-by-link cost, ignoring the important delay incurred at high-degree nodes in the tree. We formulate the problems of minimizing the average and the maximum delay in a multicast tree, and show that they are NP-hard. We also show that no polynomial-time approximation algorithm can be found for these problem with a reasonable approximation ratio. We design four algorithms that heuristically create multicast trees with minimum delays: for each of the two minimum-average and minimum-maximum delay cases, we design a delay-efficient algorithm as well as a scale-efficient one that is orders of magnitude faster. The collection of these algorithms supports a wide range of overlay scales, from a few hundred to a few thousand nodes. We have conducted a comprehensive evaluation of our algorithms on different real-world dataset and on overlays created by three diverse network models. Our results confirm that our algorithms can achieve significantly lower delays (up to 60%) and smaller running times (up to orders of magnitude) than previous minimum-delay multicast algorithms.

Next, we are working on algorithms to efficiently *update* a previously created multicast tree, rather than re-building it, if the group of receivers has not significantly changed (such as in the example at the beginning of Section I).



(a) MinSum-delay trees. (b) MinMax-delay trees.

Fig. 6. The performance (tree-efficiency as well as running time) of the algorithms on different datasets and overlay models.

REFERENCES

[1] P. Davis, "Distributed interactive simulation in the evolution of DoD warfare modeling and simulation," *Proceedings of the IEEE*, vol. 83, no. 8, 1995.

[2] R. Hofer and M. Loper, "DIS today [distributed interactive simulation]," *Proceedings of the IEEE*, vol. 83, no. 8, 1995.

[3] D. Schneider, "Trading at the speed of light," *IEEE Spectrum*, vol. 48, no. 10, October 2011.

[4] "Trading shares in milliseconds," MIT Technology Review, February 2010, http://www.technologyreview.com/computing/24167.

[5] S. Krause, "A case for mutual notification: a survey of P2P protocols for massively multiplayer online games," in *ACM SIGCOMM'08 Workshops*.

[6] X. Jiang, F. Safaei, and P. Boustead, "Latency and scalability: a survey of issues and techniques for supporting networked games," in *IEEE ICN'05*.

[7] D. Ahmed and S. Shirmohammadi, "A dynamic area of interest management and collaboration model for P2P MMOGs," in *IEEE/ACM DS-RT'08*.

[8] S. Benford, C. Greenhalgh, T. Rodden, and J. Pycock, "Collaborative virtual environments," *Communications of the ACM*, vol. 44, no. 7, 2001.

[9] D. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, 2004.

[10] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: a framework for delivering multicast to end users," in *IEEE INFOCOM'02*.

[11] M. Hosseini, D. Ahmed, S. Shirmohammadi, and N. Georganas, "A survey of application-layer multicast protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, 2007.

[12] J. Kurian and K. Sarac, "A survey on the design, applications, and enhancements of application-layer overlay networks," *ACM Computing Surveys*, vol. 43, no. 1, 2010.

[13] Y. Zhu, C. Dovrolis, and M. Ammar, "Dynamic overlay routing based on available bandwidth estimation: A simulation study," *Computer Networks*, vol. 50, no. 6, 2006.

[14] H. Zhang, L. Tang, and J. Li, "Impact of overlay routing on end-to-end delay," in *ICCCN'06*.

[15] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *ACM SOSP'01*.

[16] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, 2002.

[17] H. Li, L. Mason, and M. Rabbat, "Distributed adaptive diverse routing for voice-over-IP in service overlay networks," *IEEE Transactions on Network and Service Management*, vol. 6, no. 3, 2009.

[18] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis, "An overlay architecture for high-quality VoIP streams," *IEEE Transactions on Multimedia*, vol. 8, no. 6, 2006.

[19] G. Rodolakis, A. Laouiti, P. Jacquet, and A. Naimi, "Multicast overlay spanning trees in ad hoc networks: Capacity bounds, protocol design and performance evaluation," *Computer Communications*, vol. 31, no. 7, 2008.

[20] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *ACM SIGCOMM'02*.

[21] S. Shi, J. Turner, and M. Waldvogel, "Dimensioning server access bandwidth and multicast routing in overlay networks," in *ACM NOSSDAV'01*.

[22] F. Bauer and A. Varma, "Degree-constrained multicasting in point-to-point networks," in *IEEE INFOCOM'95*.

[23] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "OMNI: An efficient overlay multicast infrastructure for real-time applications," *Computer Networks*, vol. 50, no. 6, 2006.

[24] W. Zhang, Q. Zheng, H. Li, and F. Tian, "An overlay multicast protocol for live streaming and delay-guaranteed interactive media," *Journal of Network and Computer Applications*, vol. 35, no. 1, 2011.

[25] P. Jokela, A. Zahemszky, C. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," in *ACM SIGCOMM'09*.

[26] N. Malouch, L. Zhen, D. Rubenstein, and S. Sahu, "A graph theoretic approach to bounding delay in proxy-assisted, end-system multicast," in *IEEE IWQoS'02*.

[27] H. Ito, H. Nagamochi, Y. Sugiyama, and M. Fujita, "File transfer tree problems," in *ISAAC'02*.

[28] S. Shi and J. Turner, "Routing in overlay multicast networks," in *IEEE INFOCOM'02*.

[29] E. Brosh, A. Levin, and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application-layer multicast trees," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, 2007.

[30] K. Mokhtarian and H.-A. Jacobsen, "Minimum-delay overlay multicast," Technical Report, July 2012, www.msrg.org/papers/TR20120722-MJ12.

[31] B. Zhang, T. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement-based analysis, modeling, and synthesis of the Internet delay space," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, 2010.

[32] B. Wong, A. Slivkins, and E. Sirer, "Meridian: a lightweight network location service without virtual coordinates," in *ACM SIGCOMM'05*.

[33] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, 1982.

[34] J. Kleinberg, "The small-world phenomenon: an algorithm perspective," in *ACM STOC'00*.

[35] P. Erdos and A. Renyi, "On random graphs," *Publicationes Mathematicae Debrecen*, vol. 6, 1959.

[36] R. Albert and A. Barabasi, "Statistical mechanics of complex networks," *Review of Modern Physics*, vol. 74, 2002.

[37] D. Stutzbach, R. Rejaie, and S. Sen, "Characterizing unstructured overlay topologies in modern P2P file-sharing systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, 2008.