A Distributed Algorithm of Delay-Bounded Multicast Routing for Multimedia Applications in Wide Area Networks

Xiaohua Jia, *Member, IEEE*

***Abstract*— Multicast routing is to find a tree which is rooted from a source node and contains all multicast destinations. There are two requirements of multicast routing in many multimedia applications: optimal network cost and bounded delay. The network cost of a tree is defined as the sum of the cost of all links in the tree. The bounded delay of a routing tree refers to the feature that the accumulated delay from the source to any destination along the tree shall not exceed a prespecified bound. This paper presents a distributed heuristic algorithm which generates routing trees having a suboptimal network cost under the delay bound constraint. The proposed algorithm is fully distributed, efficient in terms of the number of messages and convergence time, and flexible in dynamic membership changes. A large amount of simulations have been done to show the network cost of the routing trees generated by our algorithm is similar to, or even better than, other existing algorithms.**

***Index Terms*— Delay-bounded multicast, distributed routing algorithm, multicast routing, multimedia systems, real-time communications.**

I. Introduction

MULTICAST is a kind of group communication which requires simultaneous transmission of messages from a source to a group of destinations. Real-time multicast is a multicast scheme in which messages should be received by all destinations within a specified delay bound. There are many applications relying on real-time multicast services, such as interactive video conferencing systems, distributed group-game systems, concurrent editing systems, and so on. In real-time communications, a (logical) connection from the source to the destination(s) needs to be established before any data transmission occurs [7]. During the connection setup, sufficient network resources (i.e., network bandwidth, buffer, etc.) are reserved at each network node on the connection so that user required quality of service (QoS) can be guaranteed at data transmission time.

There are two steps of a multicast connection establishment in wide area networks (WAN's): routing and configuration of the connection. The routing is to find a routing tree which is rooted from the source and links all the destinations. The configuration of a connection is to configure the new connection at each node on the tree, which includes the operations, such as reserving network resources, registering the connection in a switching table used to switch incoming packets to the corresponding outgoing links, and so on.

In connection-oriented communication, once a connection is set up all traffic follows the selected route until the connection is torn down. Therefore, it is important for a multicast connection to select a routing tree whose network cost is minimal. The network cost of a tree is defined as the sum of the cost of all links in the tree. Finding such a tree in a network is called the Steiner tree problem [12], which is NP-complete [11]. Many inexpensive heuristic algorithms have been proposed (see surveys in [13], [19], and [20]) to find approximate solutions to Steiner trees. However, there are three major difficulties in applying the existing heuristic algorithms to the routing of real-time multicast connections:

The first difficulty is routing in a fully distributed fashion. Most of the proposed heuristic algorithms are centralized in nature. A centralized algorithm requires a central node (or every node) to be responsible for computing the entire routing tree, and this central node must have the full knowledge about the global network. It suffers from some drawbacks in large networks, such as poor fault tolerance (failure of the central node), heavy computing load at the central node, high communication cost in keeping network information up-to-date, and inaccuracy of routing information.

The second difficulty is the generation of an optimal routing tree under a delay bound. In addition to having the minimal network cost, the routing tree for a real-time connection requires that the delay from the source to any destination shall not exceed a prespecified bound. However, the requirement of minimizing network cost often conflicts with the bounded delay requirement in multicast routing.

The third difficulty is the integration of routing with the connection configuration into a single phase of operations. The existing methods for establishing connections tend to separate the routing operation from the connection configuration. That is, a routing tree is generated first, and then the configuration is performed, node by node, along the tree. It not only incurs more network communication cost, but also makes the time for connection establishment longer.

This paper proposes a distributed routing algorithm for real-time multicast connections, aimed at overcoming the above

Manuscript received November 13, 1996; revised February 28, 1998 and August 14, 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor U. Shankar. This work was supported in part by City University of Hong Kong under Grant 7000677.

The author is with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (e-mail: jia@cs.cityu.edu.hk).

Publisher Item Identifier S 1063-6692(98)09579-X.

three difficulties. The proposed algorithm has the following advantages.

1) *Fully Distributed:* Each node operates based on its local routing information and the coordination with other nodes is done via network message passing.
2) *Suboptimal Routing Trees Under Delay Bound:* The network cost of routing trees is suboptimal under the condition that the delay from a source to any destination along the tree does not exceed a prespecified bound.
3) *Integration of Multicast Routing with Connection Configuration:* Because of this feature, our method uses a small number of messages and short convergence time to establish a connection. The message cost is even comparable to those using centralized routing.
4) *Dynamic Membership Changes of Multicast Groups:* A destination can leave or join a multicast group without restructuring the existing routing tree (i.e., not affect the existing traffic on the connection).

II. System Model and Problem Specification

A. Real-Time Optimal Multicast Routing

The network is modeled by a connected graph $G(V, E)$, where the nodes in the graph represent network routers and the edges correspond to communication links between nodes. Each link $l \in E$ is associated with two parameters $c(l)$ and $d(l)$. $c(l)$ denotes the communication cost of l, which can be the number of hops of l in WAN's. $d(l)$ is a measure of delay that messages experience on l, which includes the queuing, transmission, and propagation components. In this paper, we assume that the two parameters $c(l)$ and $d(l)$ have the following relationship:

$$\forall\, l_1, l_2 \in E\text{: if } c(l_1) \geq c(l_2), \text{ then } d(l_1) \geq d(l_2). \quad (1)$$

Under this assumption, the *least cost path* between two nodes (i.e., the shortest path in terms of cost) is always the *shortest delay path* between them.

The assumption in (1) is reasonable in most data communication networks. It is especially true when the cost parameter is measured by the number of hops, because a path with less number of hops usually has shorter network delay. Fig. 1 is a simple example of a network graph. For the clarity of diagrams, we use the same integer number to indicate both values of cost and delay parameters of links in the diagram [it does not affect the generality of the diagrams because of the assumption on the relationship between the cost and delay parameters in (1)]. Our algorithm still treats the two parameters separately.

A multicast operation transmits a message to a group of destinations. A multicast route always takes a tree structure to reduce the number of message copies on the network, where message copying occurs only at fork nodes of the tree. A multicast routing tree can be defined as the following.

Given a source node $s \in V$, a set of destination nodes $D \subset V$, with $s \notin D$, a routing tree for a multicast connection is a subtree of the graph $G(V, E)$ rooted from s, that contains

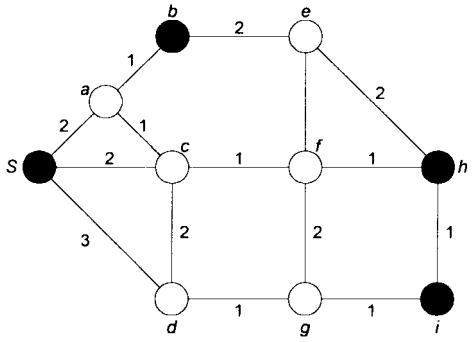

Fig. 1. A simple example of a network graph.

all of the nodes of D and an arbitrary subset of $(V - D)$, whose leaf set consists only of a subset of nodes of D.

When multicasting a message to the nodes of D, source node s (the root of a routing tree) sends a copy of the message to each of its children along the tree. These children in turn transmit the message to their children until all nodes in the tree (thus, all nodes in D) have received the message. If $|D| = 1$, it becomes a unicast, and if $|D| = |V| - 1$, it is a nonselective broadcast. According to the nature of trees, a multicast message flows through each branch of the routing tree once and only once to reach all the destinations. Therefore, the network cost of multicasting a message to a group of destinations is proportional to the sum of the cost of all links in the tree. We define the network cost of a multicast routing tree T as

$$\text{NetworkCost}(T) = \sum_{l \in T} c(l). \quad (2)$$

One important objective of multicast routing is to optimize (i.e., minimize) the network cost of routing trees. By selecting optimal routing trees for multicasts, the network cost of each multicast can be minimized. The gain of the savings is especially significant in multimedia applications where communication occurs more frequently and messages are usually very large in size due to the transmission of audio or video files.

Another requirement of multimedia communication is bounded delay of message delivery. In order for audio or video to be effectively used in interactive communication, the delay from source node s to any destination is expected to be upper bounded. The exact amount of this delay bound depends on particular applications. The bounded delay requirement can be expressed as the following, where $P(s, u)$, $u \in D$, is the path from s to u along the routing tree T and Δ is the delay bound

$$\forall\, u \in D\text{:} \sum_{l \in P(s,\, u)} d(l) \leq \Delta. \quad (3)$$

The network cost requirement is additive over the whole tree, while the bounded delay requirement is additive over individual paths from s to destinations along the tree. Our goal is to find a routing tree with optimal network cost defined in (2) under the delay constraint defined in (3). To ensure the existence of such a tree which satisfies (3), the following

condition must hold, where SDP(s, u) is the shortest delay path from s to u:

$$\forall u \in D: \sum_{l \in \mathrm{SDP}(s,u)} d(l) \leq \Delta. \quad (4)$$

Otherwise, a routing tree which can meet the delay bound Δ does not exist .

B. Related Work

The three major difficulties in routing for real-time multicast connections are: fully distributed routing, delay-bounded suboptimal routing, and integration of routing with connection configurations. We now discuss some existing solutions to those problems.

1) Distributed Steiner Tree Heuristics: Some widely adopted distributed heuristics are based on minimum spanning tree (MST) heuristics [1], [4], [9], [16]. An MST heuristic is to generate an MST of the network graph $G(E, V)$, spanning all nodes in V. Then an approximate Steiner tree is obtained by removing, from the MST, subtrees containing no nodes in $\{s\} \cup D$. Basically, there are two types of distributed MST algorithms. One type is based on the Prim's MST algorithm. Prim's algorithm [6] initializes the tree as the source node and then grows the tree by successively adding the next closest node to the tree, until all nodes are in the tree. The other type is based on Kruskal's MST algorithm [6]. Kruskal's algorithm initializes each of the nodes as a subtree and joins subtrees pairwise repeatedly until all the nodes are in a single tree. There are two disadvantages to these algorithms. First, all the nodes in the network are involved in the execution of the MST algorithms, which is very costly in large networks. Second, it takes two steps to produce a routing tree: 1) generate an MST of the whole network and 2) prune the MST to the routing tree. It costs more network messages and takes a longer time to establish a connection.

2) Delay-Bounded Suboptimal Routing: The cost requirement often conflicts with the delay requirement in multicast routing. A Steiner tree with optimal network cost may have a long delay to the farthest destination. Whereas a shortest path tree (SPT), in which each path from the source to a destination is a shortest path (in terms of delay), has the shortest delay, it may incur a high network cost. A tradeoff algorithm between optimal network cost and minimum average delay was proposed by Kumar *et al.* in [3]. It generates two routing trees, an SPT T and a Steiner tree T'. A k-degree tradeoff (k is a positive integer specified by the user) is achieved by first identifying out k destinations to whom the difference between the delay in T and the delay in T' is the largest. Then the paths to k destinations in T' are replaced by the corresponding shortest paths in T, obtaining a less optimal routing T', but with a shorter average delay. This idea can be extended to delay bounded routing by replacing those paths in the Steiner tree whose delay exceeds the bound by their shortest paths. This extended algorithm is later simulated in Section IV.

In multimedia applications, there is actually no need to minimize the average delay to all destinations. It often requires that the delay to any destinations shall be within a bound. A recent survey of delay-bounded routing algorithms and the evaluation of them can be found in [17]. Kompella *et al.* proposed two *centralized* heuristics in [15]: the *cost-delay* heuristic and the *cost* heuristic. Both of them are based on Prim's MST algorithm. A routing tree grows up from the source s. Each time when selecting the next nontree node v to add to the tree, the *cost-delay* heuristic uses the following function to convert the cost and delay of a link into the weight

$$\text{if } (\mathcal{D}[u] + t[u, v]) < \Delta$$
$$\text{then } w[u, v] = c[u, v]/(\Delta - (\mathcal{D}[u] + t[u, v]))$$

otherwise

$$w[u, v] = \infty$$

where c and t are the cost matrix and delay matrix, respectively; u is a tree node and $\mathcal{D}[u]$ is the delay from s to u along the tree. Then it selects node v which has the smallest $w[u, v]$ and adds it to the tree. The *cost* heuristic simply selects node v whose cost to the tree is minimal under the condition $\mathcal{D}[v] \leq \Delta$ and adds it to the tree. These two heuristics are later extended to distributed versions in [16]. In their distributed versions, the two policies (heuristics) of selecting edges remain the same. But each time of selecting an edge, adding to the tree, it takes one round trip along the tree formed so far: a *FIND* message is first sent from s down to the tree leaves, node by node, to determine the best out-going edge at each node; then the best out-going edge is propagated from the leaves back to s, replaced by a better choice along the way. This is a time-consuming and message-costly procedure.

Jia *et al.* [14] proposed a *centralized* algorithm which improves the performance of Kompella's *cost* heuristic. When selecting node v to add to the tree, if $\mathcal{D}[v] > \Delta$, it backtracks the tree formed so far to find a structure of the tree which can link v to the tree with the least cost under the condition $\mathcal{D}[v] \leq \Delta$. It then restructures the tree to include v into the tree. Zhu *et al.* proposed another *centralized* algorithm called the bounded shortest multicast algorithm (BSMA) in [21]. BSMA starts with an SPT to all destinations. It then iteratively replaces super-edges in the tree with lower cost paths not in the tree without violating the delay bound. A super-edge is a path in the tree between two branching nodes, two destinations, or a branching node and a destination. The operation continues until the total cost of the tree cannot be reduced any further. The computation cost of searching the best replacement of a super-edge under the delay constraint is very high. The computing complexity is $O(Kn^3 \log n)$, where K is the convergence number in searching a delay-bounded shortest path (shortest in terms of cost) for a super-edge, and n is $|V|$.

3) Integration of Routing with Connection Configuration: In a distributed environment, routing operation must be integrated with the operation of connection configuration so that a connection can be established faster. This is because connection configuration needs to be done node by node along the direction from the root to all leaf nodes. If the routing operation is separate from the configuration, it would take an

extra traverse of the whole tree to configure the connection, which is costly in both time and network messages. Centralized algorithms compute routing trees at a central node, so they need another phase for connection configuration. Some distributed heuristics, such as those based on Kruskal's MST algorithm [1], [9], construct routing trees in parallel. The input and output directions at a tree node are not known during the tree construction until the whole tree is generated. Therefore, they also need a separate phase to configure the connection after the routing tree is generated.

There is no such solution so far which can overcome all the difficulties discussed above. The principal goal of our work is to develop a routing method that is fully distributed, produces delay-bounded optimal trees, and can be integrated with the operation of connection configurations.

III. Our Algorithm

A. Requirements of Distributed Heuristics and Assumptions

Distributed heuristic algorithms do not need any node to keep the information of the whole network, which is potentially suitable for routing in large networks. A distributed algorithm used for network routing must satisfy the following criteria.

1) The execution of the algorithm is restricted on those nodes which are involved in the multicast. The whole network will not get involved in each multicast routing.
2) Each node operates based on its local routing information, so that there is no need for a central node (or every node) to keep the information of the whole network.
3) The communication cost for a multicast routing must be small and the time required for a connection establishment must be short.

The idea of our algorithm in constructing routing trees mimics Prim's MST algorithm [6] and is in combination with a distributed shortest path algorithm. We assume that each node has the information about the shortest path (in terms of cost) and the delay of the path to every other node in the graph (the information is stored in the local routing table). This can be achieved by running the Bellman–Ford's distance-vector [2], [8] on both cost and delay metrics. Furthermore, we assume there are no long live loops in the shortest paths. Several loop-free shortest path routing algorithms based on Bellman–Ford's algorithm have been already proposed, such as in [5] and [10]. Sometimes dynamic changes of routing information may still cause transient loops in shortest path routing. In our method, a routing tree is constructed sequentially (i.e., the tree grows link by link from the root) and the connection on the routing tree under construction is registered into a table of connections at each tree node as the tree grows. Thus, a loop can be identified immediately once it is formed. Our routing algorithm simply terminates and this connection setup request will be rejected when a loop is encountered.

B. Our Distributed Steiner Tree Heuristic

1) Definitions: Multicast routing is to find a routing tree which spans all destinations and whose network cost is minimum. Before discussing the details of our algorithm, we give the following definitions.

Definition 1: A shortest path from a tree T to a nontree node v (v is not in T) is a shortest path (in terms of cost) from a tree node u (u is in T) to v, denoted by $SP(u, v)$, and u satisfies

$$\text{For any tree node } k\text{: } \sum_{l \in SP(u,v)} c(l) \le \sum_{l \in SP(k,v)} c(l) \quad (5)$$

u is said to be *the tree node closest to* v.

Definition 2: The cost from a tree T to a nontree node v, denoted by $C(T, v)$, is

$$C(T, v) = \sum_{l \in SP(u,v)} c(l) \quad (6)$$

where u is the tree node closest to v.

Definition 3: A nontree node w, which is said to be the closest to a tree T, satisfies

$$\text{For any nontree node } v\text{: } C(T, w) \le C(T, v). \quad (7)$$

The above definitions of trees are also applicable to paths, which are a special case of trees: single-branch trees.

2) Basic Idea of the Algorithm: The basic idea of our algorithm is as follows. The construction of a routing tree starts with a tree containing only source node s. A destination in D which is the closest to the tree is selected. The shortest path from the tree to this destination is added into the tree. By adding a path to a tree, all nodes on the path are included into the tree. Then the next destination, which is the closest to the tree under the delay constraint defined in (3), is selected and the shortest path from the tree to it is added to the tree. At each step, an unselected destination, which is the closest to the tree under the delay constraint, is added to the tree. This operation repeats until all nodes in D are in the tree.

To record the cost from the tree to the destinations as the tree grows, a table $T2D$ (tree to destination) is introduced. An entry $T2D[d_i]$, for each destination d_i, has three fields $T2D[d_i]{\cdot}c$, $T2D[d_i]{\cdot}trenode$, and $T2D[d_i]{\cdot}tag$, representing, respectively, the cost from the tree to d_i, the tree node closest to d_i, and the status showing if d_i is in the tree or not.

Each tree node u needs to remember the accumulated delay from source node s to itself along the tree, denoted by $\mathcal{D}_u$. Both $T2D$ and $\mathcal{D}_u$ are initialized at s and carried in a connection *setup* message from node to node during routing. They are updated as the message passes through each new tree node.

For the ease of description, we use $Routab$ to denote the *local* routing table at each node. For each node u in the network, there is an entry $Routab[u]$ in the table. $Routab[u]{\cdot}c$ is the cost of the shortest path (in terms of cost) from the local node to u, and $Routab[u] \cdot d$ is the delay of this path.

3) The Algorithm Details: Every node in the system executes the same routing algorithm. It is initially in an idle state waiting for connection setup requests. The pseudocode of the algorithm is given in the Appendix.

When a node receives a request for opening a multicast connection with parameters D and Δ, it becomes the source s of the multicast connection (see procedure *open* in the

TABLE I
$T2D$ TO BE SENT OUT TO THE FIRST DESTINATION

dest	*cost*	*trenode*	*tag*
b	3	*s*	*yes*
h	4	*s*	*no*
i	5	*s*	*no*

TABLE II
$T2D$ UPDATED BY NODE a

dest	*cost*	*trenode*	*tag*
b	3	*s*	*yes*
h	3	*a*	*no*
i	5	*s*	*no*

Appendix). A $T2D$ table is first initialized. The *trenode* and *tag* fields of all entries in the $T2D$ are set to s and "*no*," respectively, indicating that the shortest paths to all destinations start from s and no destination is in the tree initially. Then the destination closest to the tree is selected and its entry is marked as "*yes*." A connection *setup* message is sent to the neighbor v via which the selected destination can be reached by the shortest path. This *setup* message carries the information of $T2D$ and $\mathcal{D}_v$ ($\mathcal{D}_v$ is initialized to $Routab[v] \cdot d$ at this node). Table I shows the $T2D$ carried in the *setup* message from s in the example of Fig. 1 with $D = \{b, h, i\}$. In this example, b is selected as the first destination to be added to the tree.

When this *setup* message arrives at an intermediate node, say u, on the way to the designated destination (see procedure *setup* in the Appendix), $\mathcal{D}_u$ is extracted from the message and recorded at this node. For each destination d_i not already in the tree, the following condition is checked:

$$(Routab[d_i].c < T2D[d_i].c) \text{ AND } (\mathcal{D}_u + Routab[d_i].d \leq \Delta). \tag{8}$$

Notice that $Routab[d_i] \cdot c$ is the cost from this node (i.e., u) to d_i, and $T2D[d_i] \cdot c$ is the cost from the so-far-formed tree to d_i. If (8) is true, entry $T2D[d_i]$ is updated as

$$T2D[d_i].c = Routab[d_i] \cdot c; \quad T2D[d_i].trenode = u.$$

Then the *setup* message is sent to the next neighbor, say u', leading to the designated destination. The accumulated delay from s to u' is $\mathcal{D}_u + Routab[u'].d$.

Table II shows the updated $T2D$ at node a when the *setup* message is sent down to b along path $\langle sab \rangle$. Suppose $\Delta = 5$. Since the cost from a to h is less than that from s to h and the accumulated delay to h via a is $5 \leq \Delta$, entry $T2D[h]$ is updated. Whereas the delay to destination i via a exceeds Δ, though the cost from a to i is less than that in $T2D$.

When the connection *setup* message reaches the designated destination, a nontree destination which is the closest to the tree is selected as the next one to be linked into the tree. The $trenode$ field of the selected destination's entry in $T2D$ is the tree node closest to this destination. This tree node will be a fork node that branches to this destination. A *fork* message is then sent to this tree node. Upon the receipt of this *fork* message, the tree node continues the connection setup by sending down a *setup* message via its neighbor to the newly selected destination (see procedure *fork* in the Appendix). In our example, node h is selected as the next destination and node a is the tree node to h.

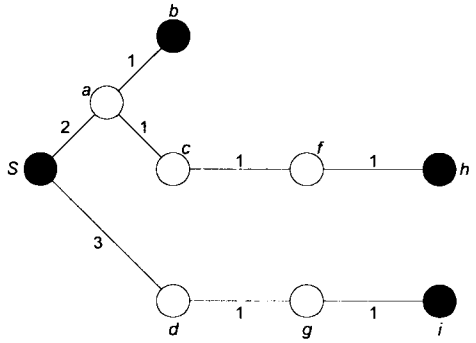

Fig. 2. A multicast routing tree from Fig. 1.

The above operation continues as the multicast connection is extended to destinations one after another, until all destinations in D are in the tree. When the last destination is added to the tree, a *completion* message is sent to s. The connection setup is completed. In our example, when the *setup* message arrives at h, node i is the last destination to be linked to the tree. Since the accumulated delay to i via h is $6 > \Delta$, link (h, i) cannot be used to link i to the tree even though the cost from h to i is very small. The shortest path from the tree to i under the delay constraint is just the shortest path from s to i in this case. Fig. 2 is the generated routing tree of Fig. 1.

This is a fully distributed algorithm. Every node involved in the execution of the algorithm operates based on its local routing information. The operations of each node are triggered by the arrivals of control messages. There are four types of control messages for a connection setup: 1) an *open* request initially from the application; 2) *setup* messages passing through every node to destinations in the routing tree; 3) *fork* messages to fork nodes; and 4) a *completion* message to the source s. Each node reacts to those messages asynchronously with others, and as the overall effect the multicast connection is established collaboratively.

C. Discussion of the Algorithm

Theorem 1: There is no cycle in a delay bounded routing tree, assuming the underlying shortest path routing is loop-free.

Proof: Assume a cycle is formed when extending the connection to destination d_i and n_j is the fork node branching to d_i. Thus, n_j is the tree node closest to d_i and path $\langle n_j d_i \rangle$ along the tree is the shortest path from n_j to d_i in our algorithm. To form the cycle, $\langle n_j d_i \rangle$ must meet another tree node, because $\langle n_j d_i \rangle$ is the shortest path and itself does not contain a cycle (a shortest path does not contain a loop). Suppose $\langle n_j d_i \rangle$ meets tree node n'_j, $n_j \neq n'_j$, as shown in Fig. 3. It is obvious that n'_j is a tree node which is closer to d_i than n_j, which contradicts the original assumption that n_j is the closest tree node to d_i at the time when d_i is selected as the next destination to be added to the tree. Thus, the theorem holds.

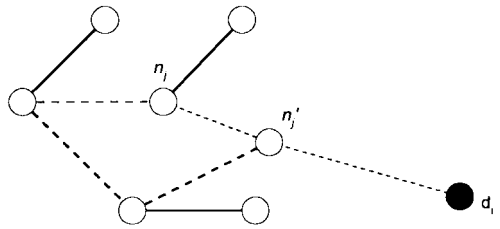

Fig. 3. Impossibility of a cycle in a routing tree.

From the above theorem, it can be seen that messages in the multicast connection would never get into an infinite loop in the network. Also, our algorithm will always find a routing tree within a finite amount of time, because at each step of the tree expansion a destination will be added into the tree via a determined shortest path between it and the tree. Our algorithm does not have the deadlock problem of as discussed in those parallel algorithms [1], [9] during connection setup.

Theorem 2: A delay bounded multicast routing tree will be always found if one exists.

Proof: Suppose there exists a delay bounded routing tree for a multicast source s, a set of destinations D, and a delay bound Δ. That is, the condition in (4) holds for the given s, D, and Δ. From the condition in (4), we know that the shortest path tree (in terms of delay) meets the delay bound. In our algorithm, the routing tree is initialized as a tree in which the route from s to any destination is the least cost path. Under the assumption of relationship between delay and cost parameters in (1), such an initial tree is also the shortest path tree in terms of delay. Therefore, even under the most stringent delay requirement (Δ is equal to the delay of the shortest path to the farthest destination), the shortest path tree will be found by our algorithm, which is the final routing tree.

D. Cost Analysis

There are two important criteria to evaluate distributed routing algorithms: the number of messages and the convergence time (i.e., time needed for a tree construction).

Theorem 3: In the worst case, our method uses two$m$ messages to construct a multicast routing tree to m destinations.

Proof: Each node in the network graph represents a router and we assume our routing algorithms run on every router. We count the transmission of a control message from a source node to a destination node as one message passing, without counting the message relay operation performed by intermediate nodes between the source and the destination. Our routing method is sequential, which uses the following messages to construct a tree:

1) m *setup* messages from s or fork nodes to the m destinations;
2) at most $m-1$ *fork* messages from $m-1$ destinations to fork nodes;
3) one *completion* message to inform s of the completion of the routing.

Thus, it takes at most $2m$ messages in total to construct a routing tree.

In terms of convergence time, our method requires $2m$ time units to complete a routing tree, assuming one time unit is the time needed for the transmission of a message between two nodes. This is obvious, because our method constructs a tree sequentially.

TABLE III
COMPARISON OF ROUTING ALGORITHMS

Algorithms	No. of messages	Converge time
Our method	$2m$	$2m$
MST heuristic	$5n\log_2 n+2\|E\|+m$	$5n\log_2 n$
Bauer's	mn	mR
Kompella's	n^3	n^3

1) Number of Messages and Convergence Time of Other Methods: For MST heuristics, an efficient distributed implementation of theMST algorithm [9] takes $O(5n\log_2 n+2|E|)$ messages and $O(5n\log_2 n)$ time units to generate a MST for a graph, where $n=|V|$, and at least another $O(m)$ messages are needed to prune the MST into a routing tree. The two distributed heuristics (shortest path heuristic and Kruskal's shortest path heuristic) proposed by Bauer *et al.* in [1] use $O(mn)$ messages and $O(mR)$ time units to construct a routing tree in the worst case, where R is the diameter of the network. The distributed algorithms proposed by Kompella *et al.* in [16] need $O(n^3)$ messages and $O(n^3)$ time units to generate a delay-bounded routing tree. Table III summarizes the costs of these algorithms.

The centralized delay-bounded routing algorithms in [14] and [21] do not need any network message to compute a routing tree. However, a centralized method has many drawbacks, as discussed in the introduction.

2) Comparisons of Total Cost for a Connection Establishment: To set up a connection, it requires operations of routing and connection configuration. One important feature of our method is that multicast routing is integrated with the connection configuration. Since our algorithm generates a routing tree sequentially from the source to destinations, connection configuration can be done at each node as the *setup* message is processed during the routing. By our method, when the construction of the routing tree is completed, the configuration of the connection is also done and the connection is ready to use. In a centralized algorithm, after a routing tree is computed, it still needs $O(m)$ messages to configure the connection. The distributed algorithms which construct routing trees in parallel also need another phase for connection configuration, which means extra cost on top of routing tree generation.

From the above discussion, we can see that our algorithm takes less network messages and less convergence time to set up a connection than other distributed multicast routing algorithms. The message cost of our method is almost comparable to those using centralized routing. The quality of the trees produced by our algorithm is analyzed and compared with other algorithms in the simulation section.

E. Dynamic Change of Multicast Memberships

In many multicast applications, such as teleconferencing systems, multicast participants are free to leave or join a

multicast session dynamically. It is important to ensure that any change of multicast memberships will not affect the traffic on the current connection and that the routing tree after the change will remain suboptimal in terms of network cost.

A widely referenced example of group management protocol is Internet group management protocol (IGMP), which has become a standard of group management for the Internet multicast. Due to the connectionless nature of the Internet, there is no logical connection among the group members and thus IGMP does not consider the network cost of a connection. In the connection-oriented communication, there is a connection for each multicast group and this connection needs to be restructured when members dynamically leave or join the group. Centralized or parallel routing methods do not allow any change of a connection once it is set up. Any change of the multicast membership requires the recomputation of a new routing tree and the setup of a new connection. Thus, they cannot provide a smooth transition from the old connection to the new one while the traffic is continuously flowing on the old one.

In our method, when a destination requests to leave a multicast group, it is disconnected from the connection. If the destination is a leaf node of the routing tree, the disconnection can be done easily. A *leave* request is sent upward (to the root direction) along the routing, node by node, until it reaches a fork node (or another destination) immediately above this destination. At each node this request passes through, the connection is released. As the result, the destination is disconnected from the multicast tree, while the rest of the connection remains intact. If the destination is not a leaf node, i.e., this destination is also responsible to forward multicast messages to other destinations, then this node cannot disconnect itself from the multicast tree. In this case it simply changes the function of this node to perform only switching operation, which switches incoming messages to the output ports of this connection. The applications are not affected.

When a node wants to join an existing multicast group, it sends a request to the source of the group. The source sends a *join* request to the current destinations of the group on the multicast channel. The request message carries a 3-tuple (*newnode, cost, trenode*) to record the cost from the tree to the new destination. At each node this request passes through, it modifies the *cost* and *trenode* information if it has a less costly path to the new destination under the delay constraint. When the request arrives at a leaf node along the tree, the leaf node sends this *cost* and *trenode* information back to the source. After collecting replies from all the leaf nodes of the tree, the source knows the tree node closest to the new destination and the cost from the tree to it. It then sends a *fork* message to this tree node to extend the connection to the new destination. By adding the new destination into the multicast group, the existing connection is unaffected.

Regarding the cost of a membership change, it takes only one control message to disconnect a destination from a connection, and $2m + 2$ messages for a new member to join a connection. Since the *join* request is sent to all destinations in parallel (by using the multicast connection), it takes only four time units to complete a *join* operation.

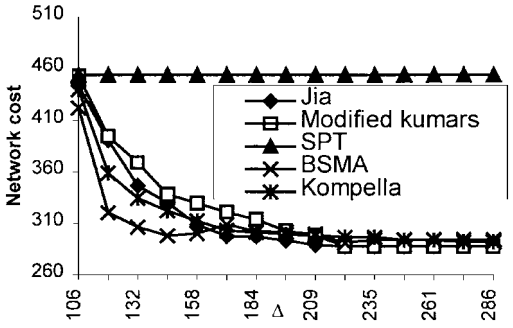

Fig. 4. Network cost versus Δ.

IV. Simulations

In the following simulations, we compare our algorithm with other delay-bounded routing algorithms on the network cost of routing trees they generate. Five algorithms, namely the SPT algorithm, the modified Kumar's algorithm [3], Kompella's *cost* heuristic [16], Zhu's BSMA algorithm [21], and the one we proposed (Jia's) are simulated.

Network topologies used in the simulations are deliberately manipulated to simulate wide-area *sparse networks*. A large network is likely to be loosely interconnected. An n-node graph is considered to be sparse when less than 5% of the possible $\binom{n}{2}$ edges are present in the graph. Since we only consider a connected graph, the number of edges in an n-node connected graph can vary from $n-1$ to $\binom{n}{2}$. The network graphs used in the simulations are constructed using the approach given in [18]. The nodes are distributed randomly over a rectangular coordinate grid. Each node is placed at a location with integer coordinates. A link between two nodes u and v is added by using the probability function $P(u, v) = \lambda \exp(-d(u, v)/\rho L)$, where $d(u, v)$ is the distance between u and v, L is the maximum distance between any two nodes, and $0 < \lambda \leq 1$, $0 < \rho \leq 1$. Larger values of λ result in graphs with higher link densities, while small values of ρ increases the density of short links relative to longer ones. In our simulations, ρ is set to 0.7. λ is decided to be 0.7 to make the graph look sparse after many tests. Graphs are generated and tested until one with a single connected component is found. The cost of a link (u, v) in the graph is the distance of nodes u and v on the rectangular coordinate grid. Again for simplicity, the delay of a link is made equal to its cost in the simulations.

The network cost is measured by the mean value of the total number of simulation runs. At each simulation point, the simulation runs 100 times. Each time, the nodes in group D are randomly picked up from the network graph. The network cost of routing trees is simulated against two parameters: delay bound Δ and multicast group size. In order to simulate the real situations, group size is always made less than 30% of the total nodes, because multimedia applications running in WAN's usually involve only a small number of nodes of the network.

Fig. 4 is the network cost versus Δ. During this round of simulations, the network size is fixed at 200 nodes, and group size is 10. We define the smallest meaningful value of Δ as $d_{\max} = \max(\{d_u |$ for any $u \in D$: d_u is the delay on

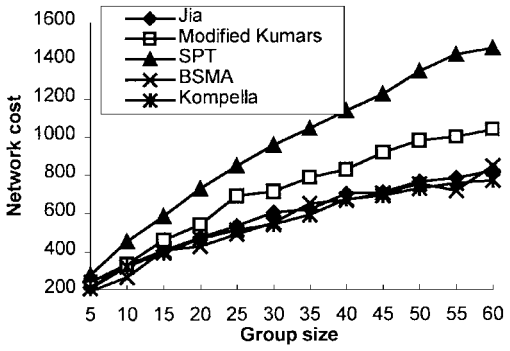

Fig. 5. Network cost versus group size.

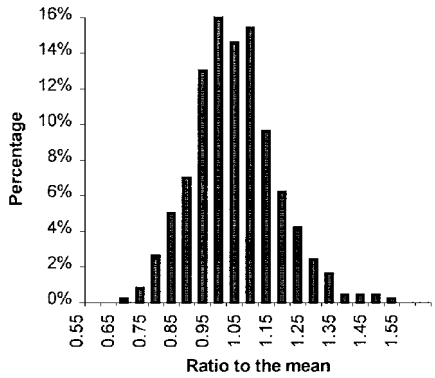

Fig. 6. Distribution of simulation results.

the shortest path from s to $u\}$). Δ starts from $d_{\max}$. With an even smaller Δ, there does not exist such a routing tree which can meet Δ. Δ is incremented by $d_{\max}/8$ each time. The increment of $d_{\max}/8$ is used after many simulation runs in order to capture any meaningful changes of network cost against the change of Δ. Since for each simulation run D is different, $d_{\max}$ is different. The Δ values on the X axis are the mean values of Δ in all runs.

From Fig. 4, we can see that the network cost of the SPT algorithm is on the top and does not change as Δ increases. This is because the generation of the shortest path tree does not depend on Δ. The other four algorithms perform much better than the SPT as Δ increases. When Δ is small, the path sharing of a routing tree is discouraged, which makes the routing tree wider (bush-like) so that the delay from the root to any node is shorter (to meet the stringent delay bound). This contributes to a high network cost of the four heuristics at small values of Δ. As Δ increases, more destinations can share path segments, which results in the decrease of the network cost. When Δ becomes large enough so that Kumar's method need not replace any path of its Steiner tree (i.e., even the longest path in its Steiner tree can meet Δ), the final routing tree generated by its Steiner tree. The curves of Jia's, Kompella's, and BSMA methods merge with (or very close to) the curve of Kumar's at some points where Δ no longer is a constraint in computing routing trees, and the network cost remains unchanged thereafter. BSMA performs slightly better than Jia's, Kompella's, and Kumar's methods when Δ is small. As we mentioned before, BSMA is a centralized algorithm. This small performance gain is at the cost of high complexity of computation. Our method performs closely to Kompella's. However, as discussed before, our method takes many fewer messages and shorter time than Kompella's to set up a connection.

Fig. 5 is the network cost versus group size. In this round of simulations, the network size is set to 200 and Δ is $d_{\max} + (\frac{3}{8})\ d_{\max}$. The group size starts at 5. It is interesting to see that the three curves representing Jia's, Kompella's, and BSMA algorithms stay very close with each other, which are significantly lower than the curves of SPT and Kumar's algorithms. It also can be seen that the increasing speed of the network cost of the three lower curves is slower than the other two, as the increase of group size. It demonstrates that our algorithm also performs well when the group size is large.

The simulation results presented above are mean values of many simulation runs. Sometimes a few bad cases may drag down the mean value significantly, which affects the general performance of an algorithm. Thus, we present the distribution of the simulation results of our algorithm in Fig. 6. The x axis is the ratio of the network cost to the mean value. The y axis is the percentage of the simulation cases whose network cost is in the range of the ratio. Each small range on the x axis is 0.05 unit (centered on the indexing numbers). During this round of simulations, network size is fixed at 200 and network topology does not change. The group size is 10 and Δ is $d_{\max} + (\frac{3}{8})\ d_{\max}$. The simulation runs 500 times, each time with a different set of destinations D.

From Fig. 6, it can be seen that the performance of our algorithm is very stable, without dramatic fluctuation. The performance is also in a good normal distribution. All the cases fall into a range between (0.70–0.025) and (1.55 + 0.025) of the mean value, with over 90% of the cases in the narrow range of 0.80–1.20.

V. Conclusions

We have presented a distributed delay-bounded multicast routing algorithm. In addition to the distributed nature and low execution cost of our algorithm, it has three special features. First, it integrates routing with connection configuration as a single phase of operations, which significantly reduces the number of messages and time of the connection setup. Second, it allows dynamic change of memberships without affecting the existing traffic on the connection, which is desired by many multimedia applications. Third, it generates good-quality routing trees with low network cost, and performance is stable.

Our method generates routing trees sequentially, i.e., extending the routing tree to one destination after another. If the multicast group is very large, it could take quite a long time to set up a connection due to the sequential nature. However, in multimedia applications and many other modern network applications, the group sizes of multicasts usually are small. In these cases, our method can set up a connection in a short time.

Appendix

Each node in the system executes the following pseudocode, where variable *local* nodes are the local node number and

$Routab$ is the local routing table. $Routab[d_i]\cdot n$, $Routab[d_i]\cdot d$, and $Routab[d_i]\cdot c$ denote, respectively, the next neighbor leading to d_i, the delay, and cost to d_i via the shortest path.

```
main()
    wait for request req-in message;
    switch (req-in.type)
        case open: open();
        case setup: setup();
        case fork: fork();
        case completion: complete();
end main;
open()
    s = local;
    D = 0;
    initialize T2D;

    // choose the first destination to go
    choose d_i ∈ D: (T2D(d_i).tag = no ) and
        T2D(d_i).c is the smallest);

    T2D(d_i).tag = yes;
    req-out.type = setup;
    req-out.T2D = T2D;
    req-out.dest = d_i;

    n = Routab(d_i).n;
    req-out.D = D + Routab[n].d;
    send(n, req − out);
end open;
    T2D = req-in.T2D;
    dest = req-in.dest;
    D = req-in.D ;

    // update information in T2D
    for each d_k ∈ D and T2D(d_k).tag = no do
        if (D + Routab(d_k).d ≤ Δ) and
            (Routab(d_k).c < T2D(d_k).c) then
            T2D(d_k).c = Routab(d_k).c;
            T2D(d_k).trenode = local;

    if local ≠ dest then
    // pass setup req to the next neighbor
        req-out.type = setup;
        req-out.T2D = T2D;
        req-out.dest = dest;
        n = Routab(dest).n;
        req-out.D = D + Routab[n].d;
        send(n, req-out);

    elseif ∀d_k ∈ D: T2D(d_k).tag = yes then
    // in the case all destinations are in the tree
        req-out.type = completion;
        send(s, req-out);
    else
    // select the next destination to go
        choose d_i ∈ D: (T2D(d_i).tag = no) and
            (T2D(d_i).c is the smallest);
setup()
    req-out.type = fork;
    req-out.T2D = T2D;
    fork = T2D(d_i).trenode;
    send(fork, req-out);
end setup;
fork()
    T2D = req-in.T2D;

    choose d_i ∈ D: T2D(d_i).tag = no and
    T2D(d_i).c is the smallest;

    T2D(d_i).tag = yes;
    req-out.type = setup;
    req-out.T2D = T2D;
    req-out.dest = d_i;

    n = Routab(d_i).n;
    req-out.D = D + Routab[n].d;
    send(n, req-out);
end fork.
```

ACKNOWLEDGMENT

The author wishes to thank the anonymous referees for their very valuable comments and constructive suggestions which helped improve the quality of the paper.

REFERENCES

[1] F. Bauer and A. Varma, "Distributed algorithms for multicast path set up in data networks," *IEEE/ACM Trans. Networking*, vol. 4, pp. 181–191, Apr. 1996.
[2] R. E. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton Univ., 1957.
[3] K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Trans. Commun.*, vol. COM-31, pp. 343–351, Mar. 1983.
[4] G. Chen, M. Houle, and M. Kuo, "The Steiner problem in distributed computing systems," *Inform. Sci.*, vol. 74, no. 1–2, pp. 73–96, Oct. 1993.
[5] C. Cheng, R. Riley, S. Kumar, and J. Garcia-Luna-Aceves, "A loop-free extended Bellman–Ford routing protocol without bouncing effect," *ACM Computer Commun. Rev.*, vol. 19, no. 4, pp. 224–236, Sept. 1989.
[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: MIT, 1992.
[7] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 368–379, Apr. 1990.
[8] J. Ford and D. R. Fulkerson, *Flows in Networks.* Princeton, NJ: Princeton Univ., 1962.
[9] R. G. Gallarger, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Programming Languages and Systems*, vol. 5, no. 1, pp. 66–77, Jan. 1983.
[10] J. Garcia-Luna-Aceves and W. Zaumen, "Area-based, loop-free Internet routing," in *Proc. IEEE INFOCOM 94*, Canada, June 1994, pp. 1000–1007.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.
[12] Gilbert and H. O. Pollak, "Steiner minimal tree," *SIAM J. Appl. Math.*, vol. 16, 1968.
[13] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, pp. 55–89, 1992.
[14] X. Jia, N. Pissinou, and K. Makki, "A real-time multicast routing algorithm for multimedia applications," *Computer Commun. J.*, vol. 20, no. 12, pp. 1098–1106, Nov. 1997.
[15] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol. 1, pp. 286–292, June 1993.
[16] ——, "Optimal multicast routing with quality of service constraints," *J. Network Syst. Management*, vol. 4, no. 2, pp. 107–131, 1996.

[17] H. F. Salama, D. S. Reeves, and Y. Viniotis, "Evaluation of multicast routing algorithms for real-time communication on high-speed networks," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 332–345, Apr. 1997.
[18] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, Dec. 1988.
[19] P. Winter, "Steiner problem in networks: A survey," *Networks*, vol. 17, pp. 129–167, 1987.
[20] P. Winter and J. M. Smith, "Path-distance heuristics for the Steiner problem in undirected networks," *Algorithmica*, pp. 309–327, 1992.
[21] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *Proc. IEEE INFOCOM'95*, 1995, pp. 377–385.

Xiaohua Jia received the Sc.D. degree in information science from the University of Tokyo, Tokyo, Japan, in 1991.

He is currently an Associate Professor in the Department of Computer Science at City University of Hong Kong, Kowloon. His research interests include operating systems, distributed systems, network systems, computer communications, and real-time communications.