Multicast Routing with End-to-End Delay and Delay Variation Constraints

George N. Rouskas, *Member, IEEE,* and Ilia Baldine

***Abstract*— We study the problem of constructing multicast trees to meet the quality of service requirements of real-time interactive applications operating in high-speed packet-switched environments. In particular, we assume that multicast communication depends on: 1) bounded delay along the paths from the source to each destination and 2) bounded variation among the delays along these paths. We first establish that the problem of determining such a constrained tree is NP-complete. We then present a heuristic that demonstrates good average case behavior in terms of the maximum interdestination delay variation. The heuristic achieves its best performance under conditions typical of multicast scenarios in high-speed networks. We also show that it is possible to dynamically reorganize the initial tree in response to changes in the destination set, in a way that is minimally disruptive to the multicast session.**

***Index Terms*— Delay constrained multicast communication, multicast routing.**

I. Introduction

In *multicast* communication, messages are concurrently sent to multiple destinations, all members of the same *multicast group*. Mechanisms to support such a form of communication are becoming an increasingly important component of the design and implementation of high-speed networks [14]. For reasons related to the efficient use of the network resources involved in a multicast session, typical approaches to multicast routing require the transmission of packets along the branches of a tree spanning the source and destination nodes. The problem of constructing multicast trees has received considerable attention in the past. One frequently considered optimization objective is to minimize the total cost of the tree, which is taken as the sum of the costs on the links of the multicast tree. The minimum cost tree is known as the Steiner tree [7] and finding such a tree is a well-known NP-hard problem [4]. Heuristics to construct trees of low overall cost have been developed in [2], [6], [9], and [15].

While total tree cost as a measure of bandwidth efficiency is certainly an important parameter, networks supporting real-time traffic will be required to provide certain quality of service guarantees in terms of the end-to-end delay along the individual paths from the source to each of the destination nodes. The problem of routing multicast traffic with real-time constraints has been studied in [8] and [16], and heuristics to compute low-cost trees which guarantee an upper bound on the end-to-end delay have been developed. For a survey and extensive simulation study of a large number of existing multicast algorithms and an evaluation of their performance in high-speed environments, the reader is referred to [12] and [13].

In this work we assume that, in addition to end-to-end delay bounds, the multicast tree must also guarantee a bound on the *variation* among the delays along the individual source-destination paths. Such a bound provides synchronization among the various receivers and insures that no receiver is "left behind" and that none is "far ahead" during the lifetime of the session. Although delay variation has not, to the best of our knowledge, been considered in the design of multicast tree algorithms, the maximum delay variation among the tree paths was one of the performance metrics included in the comparative study in [12] and [13].

There are several situations in which the need for bounded variation among the path delays arises. During a teleconference, it is important that the speaker be heard by all participants at the same time, or else the communication may lack the feeling of an interactive face-to-face discussion. When multicast messages are used to update multiple copies of a replicated data item in a distributed database system, minimizing the delay variation would minimize the length of time during which the database is in an inconsistent state. For certain applications, the ability to examine the information carried by the multicast message long before others can do the same might translate into gaining a competitive edge. A distributed game scenario in which a number of players are connected to a game server, and compete against each other using information sent by the server to their screens, would be one such example.

Buffering at the source, at the switching nodes, or at the receivers may be used as a tool to combat delay variation. Buffering at the source would require the source to maintain additional information about all destinations. It would also defeat the purpose of using a tree for routing, since each message would have to be buffered a different amount of time for each destination and thus would have to be transmitted by the source multiple times. Buffering at the switching nodes suffers from the same problems. On the other hand, having the receivers buffer multicast messages before passing them to the user is straightforward and could be used to cancel the effects of delay variation. This approach, though, would work well only when the receivers cooperate in order to accomplish a certain task as in the replicated database application above. The network should not rely on receivers to delay messages when end-users may use the information in the messages to compete

Manuscript received January 23, 1996; revised August 10, 1996. This paper was presented in part at the IEEE INFOCOM '96 Conference, San Francisco, CA, March 26–28, 1996.

The authors are with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206 USA.

Publisher Item Identifier S 0733-8716(97)02257-9.

against each other. Furthermore, the amount of buffering needed is proportional to the maximum variation of end-to-end delays. Providing bounds for this variation will result in a more efficient usage of buffering resources. We, therefore, believe that buffering at the receivers may, whenever appropriate, be used along with the multicast routing algorithms presented in this paper in order to more successfully address the problems caused by end-to-end delay variation.

In Section II we present a model that captures the salient features of multicast communication in packet-switched networks. In Section III we show that the problem of constructing trees to guarantee a bound on the variation of the end-to-end delays along the source-destination paths is NP-complete. In Section IV we develop a heuristic for this problem, and in Section V we present an approach to reorganizing the tree as nodes join or leave the multicast group. We present numerical results in Section VI, and we conclude the paper in Section VII.

II. Network Model for Multicasting

We represent a network by a weighted digraph $G = (V, A)$. V denotes the set of nodes, and A, the set of arcs, corresponds to the set of communication links connecting the nodes. We will use $n = |V|$ to refer to the number of nodes in the network. Without loss of generality, we only consider graphs with at most one arc between an ordered pair of nodes. We define a *link-delay function* $\mathcal{D}$: $A \rightarrow \mathcal{R}^+$ which assigns a nonnegative weight to each link in the network. The value $\mathcal{D}(\ell)$ associated with link $\ell \in A$ is a measure of the delay that packets experience on that link, including the queuing, transmission, and propagation components.

Under the multicast routing scenario we are considering, packets originating at some *source* node $s \in V$ in the network have to be delivered to a set $M \subseteq V - \{s\}$ of destination nodes. We will call M the *destination set* or *multicast group* and will use $m = |M|$ to denote its size. Multicast packets are routed from s to the destinations in M via the links of a *multicast tree* $T = (V_T, A_T)$ rooted at s. The multicast tree is a subgraph of G (i.e., $V_T \subseteq V$ and $A_T \subseteq A$) spanning s and the nodes in M (i.e., $M \cup \{s\} \subseteq V_T$). In addition, V_T may contain *relay* nodes, that is, nodes intermediate to the path from the source to a destination. Let $P_T(s, v)$ denote the path from source s to destination $v \in M$ in the tree T. Then, multicast packets from s to v experience a total delay of $\sum_{\ell \in P_T(s, v)} \mathcal{D}(\ell)$.

We now introduce two parameters to characterize the quality of the tree as perceived by the application performing the multicast. These parameters relate the end-to-end delays along individual source-destination paths to the desired level of quality of service.

- *Source-destination delay tolerance,* Δ: Parameter Δ represents an upper bound on the acceptable end-to-end delay along any path from the source to a destination node. This parameter reflects the fact that the information carried by multicast packets becomes stale Δ time units after its transmission at the source.
- *Interdestination delay variation tolerance,* δ: Parameter δ is the maximum difference between the end-to-end delays along the paths from the source to any two destination nodes that can be tolerated by the application. In essence, this parameter defines a synchronization window for the various receivers.

By supplying values for parameters Δ and δ, the application in effect imposes a set of constraints on the paths of the multicast tree, as discussed next.

III. Delay- and Delay Variation-Bounded Multicast Trees

Let Δ and δ be the delay and delay variation tolerances, respectively, specified by a higher level application. Our objective is to determine a multicast tree such that the delays along all source-destination paths are within the two tolerances. This *delay- and delay variation-bounded multicast tree (DVBMT)* problem naturally arises as a decision problem.

Problem III.1 (DVBMT): Given a network $G = (V, A)$, a source node $s \in V$, a multicast group $M \subseteq V - \{s\}$, a link-delay function $\mathcal{D} : A \rightarrow \mathcal{R}^+$, a delay tolerance Δ, and a delay variation tolerance δ, is there a tree $T = (V_T, A_T)$ spanning s and the nodes in M, such that

$$\sum_{\ell \in P_T(s, v)} \mathcal{D}(\ell) \leq \Delta \quad \forall v \in M \tag{1}$$

$$\left| \sum_{\ell \in P_T(s, v)} \mathcal{D}(\ell) - \sum_{\ell \in P_T(s, u)} \mathcal{D}(\ell) \right| \leq \delta \quad \forall v, u \in M. \tag{2}$$

We will refer to (1) as the *source-destination delay constraint,* while (2) will be called the *interdestination delay variation constraint.* A tree T is a *feasible* tree if and only if T satisfies both (1) and (2). Constraints (1) and (2) represent two conflicting objectives. Indeed, the delay constraint (1) dictates that short paths be used. But choosing the shortest paths may lead to a violation of the delay variation constraint among nodes that are close to the source and nodes that are far away from it. Consequently, it may be necessary to select longer paths for some nodes in order to satisfy (2). The problem of finding a feasible tree is one of selecting paths in a way that strikes a balance between the two objectives.

The source-destination constraint (1) has been previously considered in the context of constrained Steiner trees [8], [16]. Also, in a recent study [12], [13] to evaluate the performance of a number of multicast algorithms and their suitability to high-speed real-time applications, the following quantity was used as a criterion in the evaluation:

$$\delta_T = \max_{u, v \in M} \left\{ \left| \sum_{\ell \in P_T(s, u)} \mathcal{D}(\ell) - \sum_{\ell \in P_T(s, v)} \mathcal{D}(\ell) \right| \right\}. \tag{3}$$

Quantity δ_T is the maximum interdestination delay variation in a tree T. According to the study, none of the existing algorithms provides good performance in terms of δ_T. This is not surprising, as none of the algorithms considered in [12] and [13] takes the delay variation constraint (2) into account.

The following theorem establishes that DVBMT is NP-complete. A heuristic approach to solving DVBMT is presented in the next section.

Theorem III.1: DVBMT is NP-complete whenever the size of the multicast group $|M| \geq 2$.

Proof: DVBMT can be easily seen to be in the class NP. We now transform PARTITION [5] to DVBMT. It is sufficient to find a transformation for $|M| = 2$. Let $S = \{1, 2, \cdots, k\}$ be the set of elements of weights a_i, $i = 1, \cdots, k$, making up an arbitrary instance of PARTITION, and let $A = \sum_{i=1}^{k} a_i$. We construct an instance of DVBMT as follows (see Fig. 1). The network $G = (V, A)$ has $n = k + 3$ nodes, with $V = \{s, v, u, r_1, r_2, \cdots, r_k\}$, where s is the source node and $M = \{v, u\}$ is the destination set. The set A of links is

$$A = \{(s, v), (s, r_1), \cdots, (s, r_k), (r_1, u), \cdots, (r_k, u), (r_1, r_2), \cdots, (r_1, r_k), (r_2, r_1), (r_2, r_3), \cdots, (r_2, r_k), \cdots, (r_k, r_1), \cdots, (r_k, r_{k-1})\}. \quad (4)$$

In other words, there is a directed link from s to v, one link from s to each node r_i, one link from each node r_i to u, and one link from r_i to r_j, $i, j = 1, \cdots, k$, $i \neq j$ (i.e., the subgraph of G containing only nodes r_i, $i = 1, \cdots, k$, is a complete graph on these nodes). There is only one path from s to destination node v consisting of the single link (s, v), but a path from s to the other destination u may contain any number of the nodes r_i, $i = 1, \cdots, k$, and in any order. The link-delay function is defined as

$$\mathcal{D}(\ell) = \begin{cases} \frac{A}{2}, & \text{if } \ell = (s, v) \\ 0, & \text{if } \ell = (x, u), x \in V \\ a_i, & \text{if } \ell = (x, r_i), x \in V. \end{cases} \quad (5)$$

As a result, if the path from s to u passes through node r_i for some i, then a delay equal to a_i is incurred along the link that leads to r_i. Finally, the delay and delay variation tolerances are $\Delta = \frac{A}{2}$ and $\delta = 0$, respectively.

It is obvious that this transformation can be performed in polynomial time. We now show that a feasible tree exists for this instance of DVBMT if and only if set S has a partition. If S has a partition S_1, S_2, then $S_1 = \{a_{\pi_1}, \cdots, a_{\pi_l}\}$ for some $l < k$. The tree consisting of path (s, v) and path $(s, r_{\pi_1}), (r_{\pi_1}, r_{\pi_2}), \cdots, (r_{\pi_{l-1}}, r_{\pi_l}), (r_{\pi_l}, u)$, is then a feasible tree for DVBMT, as the delay along both paths is equal to $A/2$. Conversely, let T be a feasible tree for DVBMT. Then T must include the path (s, v) of delay $A/2$, as this is the only path from the source to v. Let $(s, r_{\pi_1}), (r_{\pi_1}, r_{\pi_2}), \cdots, (r_{\pi_{l-1}}, r_{\pi_l}), (r_{\pi_l}, u)$, be the path from s to u on tree T. Since T is a feasible tree and $\delta = 0$, the delay along the latter path is equal to $A/2$, and $l < k$ (for if $l = k$, the path from s to u would include all r_i, $i = 1, \cdots, k$, and the delay along the path would equal A, contradicting our hypothesis that T is a feasible tree). Then, $\sum_{i=1}^{l} a_{\pi_i} = A/2$, implying that $S_1 = \{a_{\pi_1}, \cdots, a_{\pi_l}\}$, $S_2 = S - S_1 \neq \phi$, is a partition of S. □

IV. A Multicast Tree Algorithm for DVBMT

We now present an algorithm to construct a tree satisfying constraints (1) and (2) for the given values of the path delay and the interdestination delay variation tolerances. We assume that complete information regarding the network topology is stored locally at node s, making it possible to determine the multicast tree at the source itself. This information may be collected and updated using an existing topology-broadcast algorithm [1].

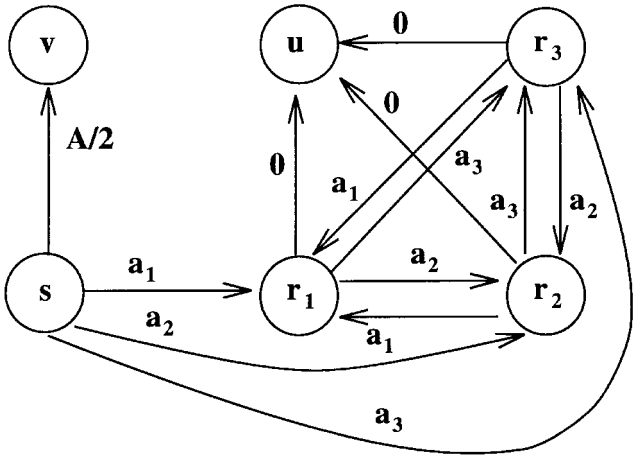


Fig. 1. Instance of DVBMT corresponding to an instance of PARTITION with $S = \{1, 2, 3\}$.

The sequence of actions for constructing a multicast tree is shown in Fig. 2. As a first step, the tree T_0 of shortest paths from s to all nodes in M is constructed using Dijkstra's algorithm [3]. If T_0 does not satisfy the delay constraint (1), no tree may satisfy it, implying that the tolerance Δ is too tight. Negotiation may then be necessary to determine a looser value for Δ. Suppose now that the original or negotiated value of Δ is such that the delay requirement (1) is met for tree T_0. If T_0 also meets the delay variation requirement (2), then it is a feasible tree, and the multicast session may take place over the tree of shortest paths.

If T_0 does not satisfy constraint (2), then the source executes the *delay variation multicast algorithm* (DVMA), a search algorithm described in the next subsection, in an attempt to construct a new tree in which the path delays satisfy both (1) and (2). If the algorithm finds a feasible tree T for the given instance of the DVBMT problem, then the multicast session may proceed. However, a search heuristic may fail to discover a feasible tree, either because no such tree exists or because of the ineffectiveness of the search strategy employed. Hence, DVMA always returns the tree T with the smallest value of δ_T in (3) among the trees considered. Regardless of whether a solution to the given instance of DVBMT problem exists or not, the tree with the smallest value of δ_T is the best tree that can be obtained with the search algorithm at hand. The source may then negotiate with the destinations to determine whether an acceptable level of quality of service can be sustained for the given value of δ_T.

A. Delay Variation Multicast Algorithm (DVMA)

Let T_0 be the tree of shortest paths from source s to the nodes in the destination set M. Let us also assume that T_0 meets the delay requirement (1), but that it does not meet the delay variation requirement (2). The DVMA, described in detail in Fig. 3, can then be used to search through the space

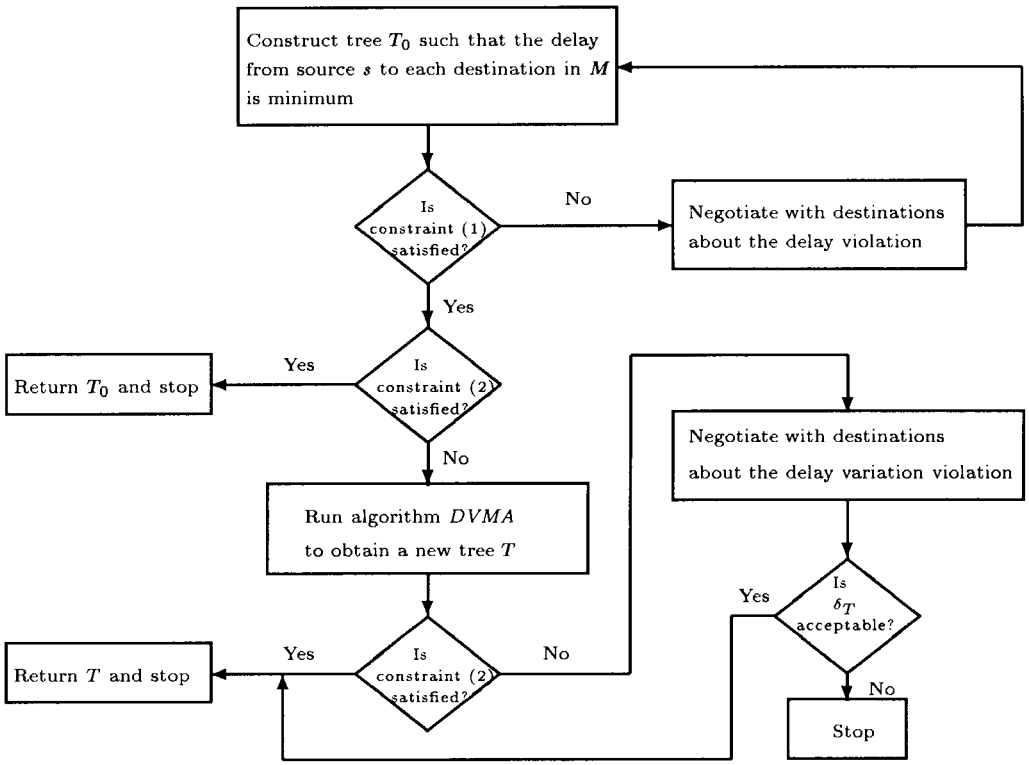


Fig. 2. Flowchart of the approach to obtaining a multicast tree for the DVBMT problem.

of *candidate* trees (i.e., trees spanning s and the nodes in M) for a feasible solution to the DVBMT problem. DVMA either returns a feasible tree, or, having failed to discover such a tree, it returns one which: 1) satisfies the delay constraint (1) and 2) has the least value of δ_T *among the trees considered by the algorithm.* We now describe the basic idea behind the operation of DVMA.

Let M be the destination set, and assume for the moment that a feasible tree $T = (V_T, A_T)$ spanning s and a subset of M has already been determined. Let $U = M - (M \cap V_T)$ be the set of destination nodes not in the tree T. In other words, no paths from the source s to the nodes in U have been determined yet. DVMA operates by augmenting tree T to eventually include all nodes in U. DVMA repeats the following three steps as long as $U \neq \phi$.

1) Select a destination node $u \in U$.
2) Find a "good" path from a node $v \in V_T$ to u that uses no nodes in V_T other than v, and no links in A_T.
3) Construct a new tree T' by including all nodes and links of this path to the initial tree T, and update U to exclude u and any other destination nodes along this path.

The second step is crucial to the operation of DVMA and warrants further explanation. Recall that our objective is to construct a feasible tree that includes all nodes in M. Therefore, a "good" path in 2) above is one which, if connected to T in 3), the resulting tree T' would be a feasible tree for the subset of the set of destination nodes it contains. To find such a path, we construct the l shortest paths from a node v of T to u. The graph used to find these paths is created by excluding all nodes of T other than v and all links of T from the original graph G. The exclusion of these nodes and links from G guarantees that connecting any of the l paths so constructed to T will not create a cycle.

It is possible, though, that none of the l paths from v to u will yield a feasible tree. For this reason, we repeat the process for all nodes $v \in V_T$ in an attempt to find a "good" path between any $v \in V_T$ and u. Even so, the algorithm may still not be able to find such a path. For instance, a feasible tree for this destination set may not exist in the first place. Recall, however, that we would like the algorithm to return the best tree (in terms of maximum interdestination delay variation) it can find. We now modify our definition of a "good" path so that, if a path yielding a feasible tree T' can not be found, a "good" path is one which: 1) the total delay from s to u (i.e., the delay from s to v in T, plus the delay from v to u over the path) is at most Δ, and 2) the tree T' created by connecting this path to T has the least value of maximum delay variation among the trees constructed by connecting the other paths to T. In essence, the purpose of the greedy rule 2) is to prune the search space, i.e., to prevent certain candidate trees from receiving further consideration.

The only question that remains to be answered, is how an initial tree T is constructed. To answer this question consider T_0, the tree of shortest paths, which, by hypothesis, does not satisfy the delay variation constraint (2). Let w be the destination node with the longest path in this tree. Since it is not possible to make the delay from s to w any smaller than the delay incurred over the path from s to w in T_0, to construct a feasible tree we must find longer paths from s to some or all of the other destination nodes. Hence, we start with an initial

Delay Variation Multicast Algorithm (DVMA)

The algorithm is executed if T_0, the tree of shortest paths, satisfies constraint (1) but does not satisfy constraint (2). We let $w \in M$ be a node such that $\sum_{\ell \in P_{T_0}(s,w)} \mathcal{D}(\ell) = \max_{v \in M} \left\{ \sum_{\ell \in P_{T_0}(s,v)} \mathcal{D}(\ell) \right\}$.

1. begin
2. Let $T = T_0$ // T is the tree returned by the algorithm
3. Find the first k shortest paths from s to w in the original graph $G = (V, A)$, such that the delay from s to w over these paths is less than Δ; label these paths $p_1, \ldots, p_k$ in increasing order of delay
4. for $i = 1$ to k do // construct a multicast tree T_i for each path p_i
5. Initialize $T_i = (V_i, A_i)$ to include all the nodes and links of path p_i; obviously, $s, w \in V_i$
6. Let $U = M - (M \cap V_i)$ be the set of destinations not yet connected to the tree T_i
7. while $U \neq \phi$ do
8. Pick any node $u \in U$ // will connect u to the tree T_i
9. for each node $v \in V_i$ do // find a path from v to u
10. Construct a new graph G' starting with the initial graph G and excluding all nodes in $V_i - \{v\}$ and all links in A_i, and all nodes in $U - \{u\}$ and their links
11. Find the first l shortest paths from v to u in the new graph G'
12. Of these l paths choose the best one (as described in Section 4.1) and call it q_v
13. end of for each node $v \in V_i$ loop
14. Select the best path q among all paths $q_v, v \in V_i$ (as in Step 12 above)
15. Update $T_i = (V_i, A_i)$ to include all nodes and links in path q
16. Update $U = M - (M \cap V_i)$
// node u, and possibly other nodes in U have now been connected to T_i
17. end of while loop // construction of tree T_i has been completed
18. If tree T_i satisfies constraint (2) return T_i and stop
19. Let T be the tree among T and T_i with the smallest value of δ_T in (3)
20. end of for i loop
21. return T // no tree satisfied the inter-destination delay variation constraint
22. end of the algorithm

Fig. 3. Heuristic algorithm for the DVBMT problem.

tree T consisting only of the shortest path from s to w and repeat the three steps described above to create a feasible tree that will include all other destination nodes.

To complete the description of the search strategy employed by DVMA, note that it is possible that no feasible tree for the given destination set includes the shortest path from s to w. However, if a feasible tree exists, it will contain *some* path from s to w. If the process of constructing a feasible tree starting from the shortest path from s to w fails, the second shortest path from s to w is considered as the initial tree, and the process is repeated. Our search for a feasible tree terminates when one is found, or when trees based on the first k shortest paths from s to w have been constructed, whichever occurs first. In the latter case, the algorithm will return the tree with the smallest value of δ_T in (3). The details of the resulting algorithm (DVMA) can be found in Fig. 3.

The correctness of DVMA is provided by Lemma IV.1, while Lemma IV.2 determines the running time complexity of DVMA.

Lemma IV.1—Correctness of DVMA: Algorithm DVMA returns a tree T spanning s and all nodes $v \in M$. The tree T satisfies constraint (1) and either satisfies constraint (2) or is the one with the smallest value of δ_T in (3) among the trees considered by the algorithm.

Proof: We first show that the algorithm returns a tree T spanning s and the nodes in M. If DVMA returns T_0, there is nothing to prove. Otherwise, T is one of the T_i's constructed during one iteration of the loop that starts at line 4. T is initialized to some path p_i at line 5. Clearly, at this point T is a tree containing the source s and at least one more destination $w \in M$. New nodes and links are added to T at line 15, where a new path q from a node in $v \in V_T$ to a node $u \in M$, $u \notin V_T$ is incorporated. The resulting graph is a tree since path q cannot contain any nodes or links of T other than v itself. All other nodes and links of T were removed at line 10 before path q was determined. The new tree T has at least one more node, $u \in M$. Since s was in the tree initially, no nodes are ever removed from T, and paths are added to it until all nodes in M are in T, our first claim is true.

The final tree also satisfies the delay constraint (1). If $T = T_0$ this is true by hypothesis, If $T \neq T_0$, this is also true since no path is ever added to any tree T_i unless (1) is satisfied (refer to lines 3 and 12). Finally, if the algorithm terminates at line 18, the tree returned is a feasible one. Otherwise, line 19 guarantees that the tree returned is the one with the smallest value of δ_T among the ones constructed during the execution of the algorithm. □

Lemma IV.2: The worst case complexity of DVMA is $\mathcal{O}(klmn^4)$, where k is the number of paths generated at line 3 of Fig. 3, l the number of paths generated at line 11, $m = |M|$ is the size of the multicast group M, and $n = |V|$ is the number of nodes in the network.

Proof: The running time of DVMA is dominated by the iteration between lines 4 and 20. This outer loop is executed at most k times. During one iteration of the outer loop, the "while" loop at line 7 is executed at most $m-1$ times. Let t_j be the number of nodes in the tree during the jth iteration of the "while" loop. The innermost loop starting at line nine will iterate t_j times. Inside this loop the complexity is determined by the l-shortest path algorithm at line 11, which takes time $\mathcal{O}(lN^3)$ [10] for a graph with N nodes. Graph G' has $n-t_j+1$ nodes throughout the innermost loop. The latter then takes time proportional to $lt_j(n-t_j+1)^3$. For a worst case analysis, we let t_j, for all iterations j, take the value that maximizes the quantity $t_j(x-t_j)^3$, where $x = n+1$. It is straightforward to show that for this value of t_j the complexity of the innermost loop becomes $\mathcal{O}(\ln^4)$. After accounting for the "while" and outer loops, the overall complexity of the algorithm is $\mathcal{O}(klmn^4)$. □

The maximum value that parameters k and l can take is, in the worst case, equal to the maximum number of paths of delay at most Δ between any two nodes in the network. If Δ is not very large, we expect the maximum value of both k and l to be a small constant. The actual values of k and l were left unspecified in the description of the algorithm, as in any particular implementation they will be determined by the desired compromise between the quality of the final solution of the algorithm and its speed.

V. Dynamic Reorganization of the Multicast Tree

For certain applications, nodes may join or leave the initial multicast group during the lifetime of the multicast connection. We assume that nodes currently in the multicast group may leave the group after issuing a *leave request,* while nodes that wish to join an ongoing multicast session must first issue a *join request.* Under such a scenario, it is necessary to dynamically update the multicast tree in response to changes in multicast group membership to insure that constraints (1) and (2) are always satisfied for the current destination set.

Let T be the multicast tree of an ongoing multicast session with destination set M and suppose that as a result of a join or leave request, the new destination set is M'. One possible way of approaching this *dynamic* version of the DVBMT problem would be to run DVMA anew to obtain a feasible tree T' for set M' and, following a transition period, use the new tree for routing subsequent packets of this session. There is a certain overhead associated with this approach, including the computational cost of running DVMA and the cost of the network resources involved in the transition from T to T' (i.e., the cost of tearing down old paths and establishing new ones). Since the new tree T' can be significantly different than T, this overhead can be very high. Furthermore, such a radical approach may cause receivers totally unrelated to the destination nodes added or deleted to experience disruption in service. All these drawbacks make this strategy inappropriate for real-time environments and applications where frequent changes in the destination set are anticipated.

We now describe a different strategy that minimizes both the cost incurred during the transition period and the disruption caused to the receivers. Specifically, the multicast tree is never modified unless it is absolutely necessary to do so. Even then, the new tree is not computed anew, rather, a feasible tree for the new multicast group is constructed by making incremental and localized changes to the old tree. Our approach has the additional advantage that the algorithm used to construct an initial tree for the multicast connection can also be used to reorganize the tree during the lifetime of the session.

We first describe how leave requests are handled under our approach. Assume that node $v \in M$ decides to end its participation in the multicast session. If v is not a leaf node in the current multicast tree T, then no action needs to be taken. The new tree T' can be the same as T, with the only difference being that node v will stop forwarding the multicast packets to its local user. If, however, v is a leaf node of T, then tree T has to be pruned to exclude v and, possibly, relay nodes used in T solely for forwarding packets to v. The new tree T' is essentially the same as T except in parts of the path from the source to v.

When a node $u \notin M$ decides to join the multicast group, we distinguish the following three cases.

- $u \notin V_T$, i.e., the new node is not part of the multicast tree T. We augment T to include a path from a node $v \in V_T$ to the new node u by letting $T_i = T$ and $U = \{u\}$ at lines 5 and 6, respectively, of DVMA (see Fig. 3) and executing the code between lines 7 and 17 to search for a path that would result in a feasible tree for the set $M \cup \{u\}$. Hence, the transition phase involves only the establishment of a

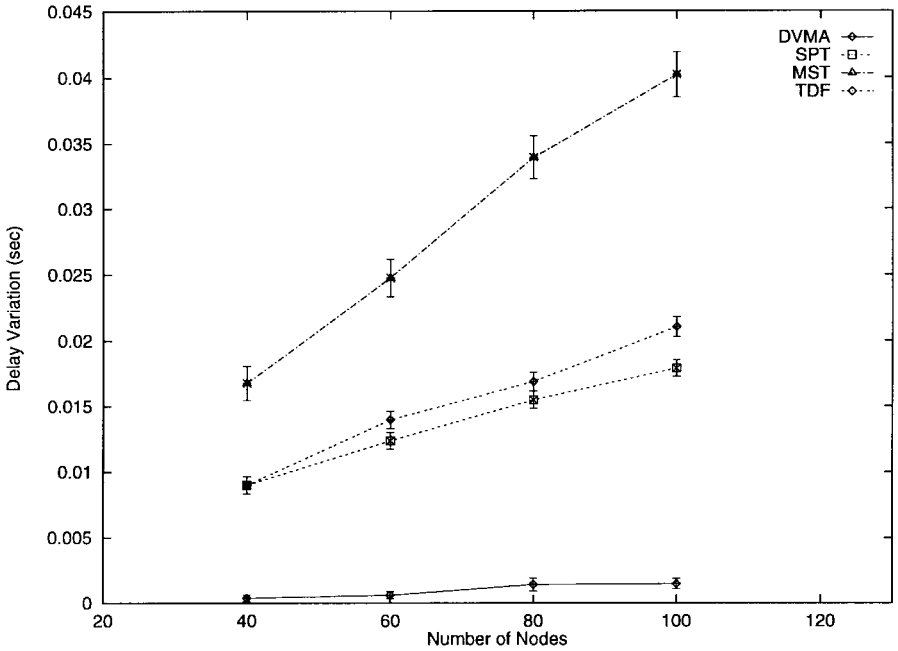


Fig. 4. Algorithm comparison for networks with average node degree equal to 2.5 and multicast group size equal to 5% of the number of nodes.

new path and does not affect any of the paths from the source to nodes already in the multicast group.[1]

- $u \in V_T$, i.e., u is a relay node of T, and the path from the source node s to u is such that the delay variation constraint (2) is satisfied for the new multicast group $M' = M \cup \{u\}$. Tree T is then a feasible tree for the new set M' and can be used without any change other than having node u now forward multicast packets to its user, in addition to forwarding them to the downstream nodes.
- $u \in V_T$, but the path from s to u is such that constraint (2) is not satisfied for the new set $M \cup \{u\}$. Consequently, a longer path from s to u has to be found. Let $W \subseteq M$ be the destination nodes in M that are downstream of u, i.e., those destination nodes in the subtree of T rooted at u). Let T_1 be the tree T after excluding its subtree rooted at u. Our approach then is to let $T_i = T_1$ and $U = W \cup \{u\}$ at lines 5 and 6, respectively, of DVMA. We then execute the code between lines 7 and 17 to connect the destination nodes in U into tree T_1. In the new tree T', packets will be routed from s to the nodes in W over new paths, but none of the paths to nodes in M–W will change.

VI. Numerical Results

We first study the average case behavior of four algorithms in terms of the maximum delay variation δ_T in (3). The four algorithms are: 1) DVMA, with $\Delta = 0.05s$ and $\delta = 0$ (this value of δ forces the algorithm to return the tree with the smallest value of δ_T it can find); 2) Dijkstra's algorithm [3] which constructs the tree of shortest paths (SPT) from the source to any node in the network (the tree is pruned so that all leaves are destination nodes); 3) Prim's algorithm [11] which constructs a tree of minimum weight (MST) spanning all nodes in the network (this tree is also pruned as above); and 4) the *tradeoff* (TDF) algorithm [2] between the minimum spanning tree heuristic for the Steiner tree problem [6] and SPT. We have run the algorithms on randomly generated graphs constructed to resemble real-world networks using the method described in [15]. The nodes of the graphs were placed in a grid of dimensions 4900 $\times$ 4900 km (roughly the size of the continental United States), and the delay along each link was set to the propagation delay of light along the link. Figs. 4–7 plot δ_T against the number of nodes n in the network for the various algorithms. Each point plotted represents the average over 300 graphs for the stated values of n, m, and the average degree of each node. Also shown in the figures are 95% confidence intervals.

The results shown in Figs. 4–6 correspond to networks with average node degree equal to 2.5 and multicast groups of sizes equal to 5, 10, and 15% of the total number of nodes. We observe that the trees constructed by DVMA have a maximum delay variation that is always smaller than that of the SPT, TDF, and MST trees. The MST is by far the worst tree in terms of δ_T. This is expected as Prim's algorithm minimizes the *total* weight of the tree, without paying any attention to the individual source-destination paths. The tree of shortest paths SPT results in values of δ_T that are between those of the MST and those of DVMA. The tradeoff algorithm TDF constructs trees with maximum delay variation larger than that of SPT, a result that is in contrast to the expectations expressed in [12].

From Figs. 4–6, we see that as the size m of the multicast group increases as a percentage of the size n of the network, the improvement of DVMA over SPT decreases from roughly an order of magnitude when $m = 0.05n$ to about 40% when $m = 0.15n$. This is expected since, the smaller the size of the multicast group, the easier it is for DVMA to find alternative,

[1] If this fails to discover such a path, there are two possible courses of action: 1) run DVMA for the new multicast group or 2) deny node u its participation in the multicast session. Which course of action to be taken will depend on the nature of the application and the cost of rerouting the connection.

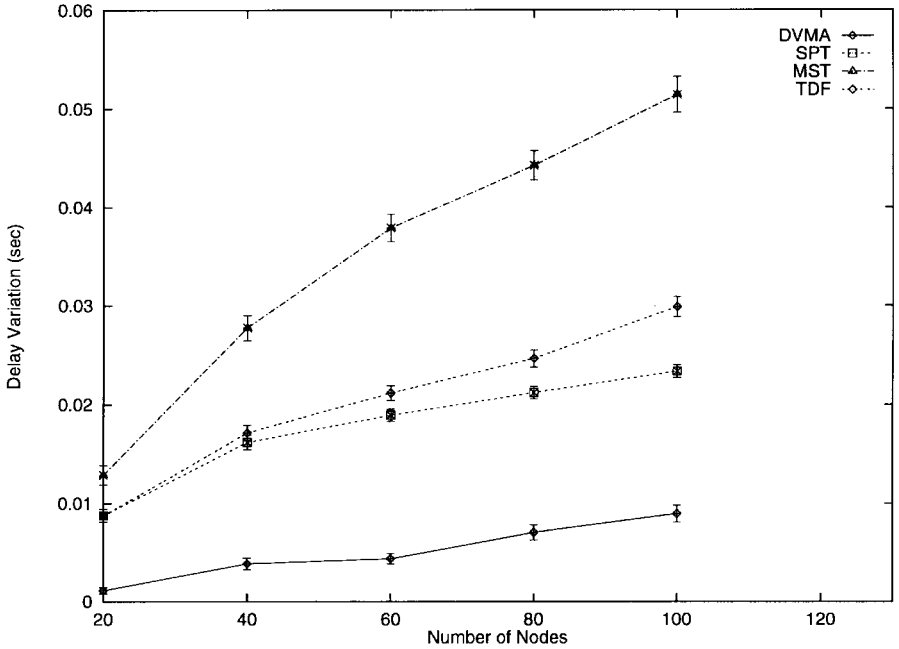


Fig. 5. Algorithm comparison for networks with average node degree equal to 2.5 and multicast group size equal to 10% of the number of nodes.

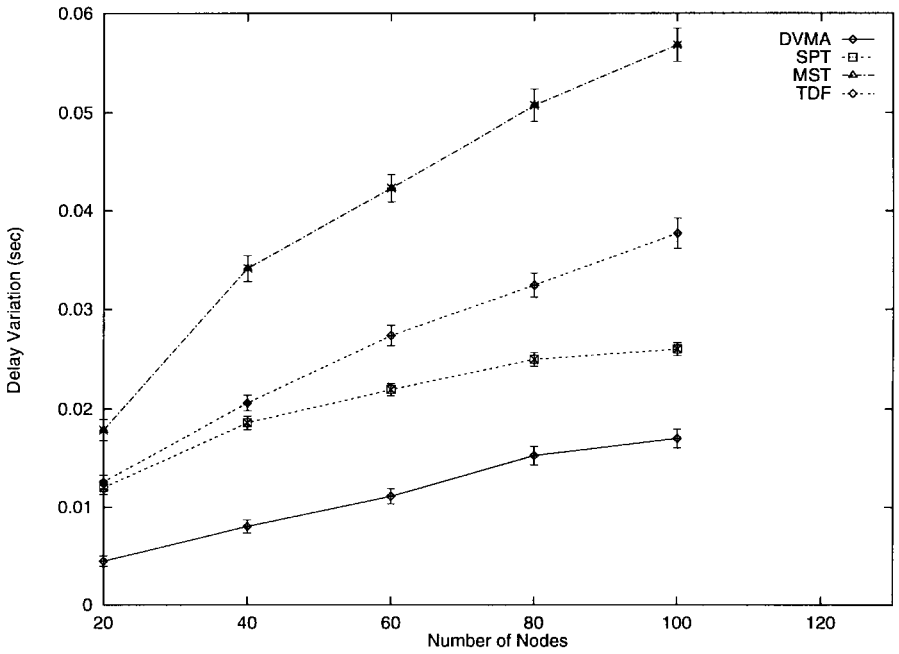


Fig. 6. Algorithm comparison for networks with average node degree equal to 2.5 and multicast group size equal to 15% of the number of nodes.

i.e., longer, paths for the nodes physically closer to the source. On the other hand, the performance of DVMA improves dramatically as the average nodal degree increases. This can be seen by comparing Fig. 4 to Fig. 7 which presents plots for the same size of destination set ($m = 0.05n$), but for an average nodal degree equal to four. The improvement is a result of the fact that a higher nodal degree translates into a larger number of paths between any two nodes, and a larger number of trees for DVMA to choose from. In addition, when the average nodal degree equals four, DVMA is able to construct trees with $\delta_T \approx 0$, independently of the number of nodes in the network. These trees would be able to meet the delay variation requirements of even the most demanding applications. The behavior of the other algorithms is not significantly affected by the nodal degree, as none of these attempt to optimize in terms of δ_T. In SPT, for instance, δ_T is determined by the relative distance of the various destinations from the source, which is almost independent of the nodal degree. Overall, our results suggest that DVMA achieves its best performance under conditions that are typical of multicast applications running in high speed networks, namely, when: 1) the size of the multicast group is relatively small compared to the total

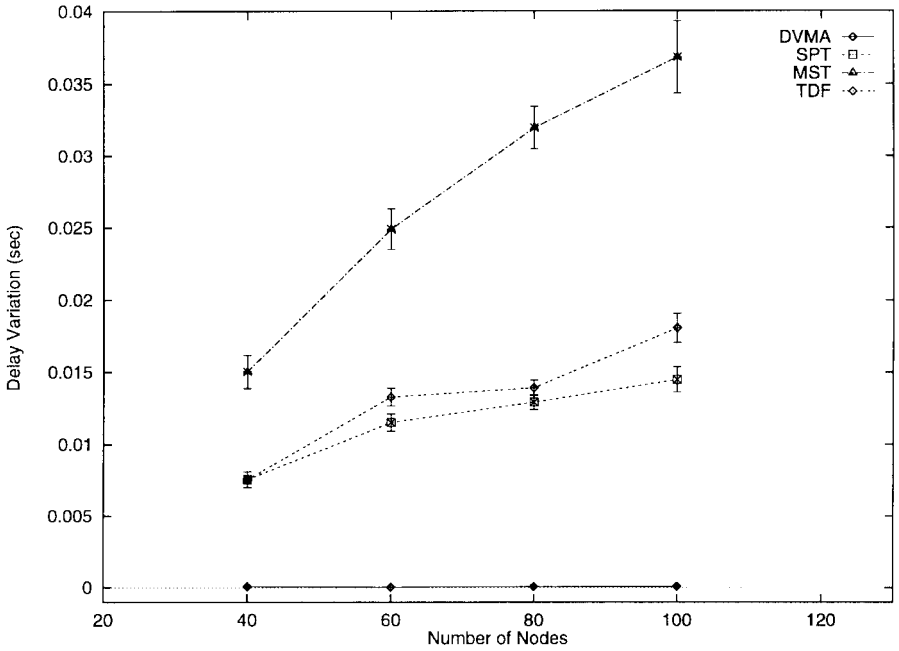


Fig. 7. Algorithm comparison for networks with average node degree equal to four and multicast group size equal to 5% of the number of nodes.

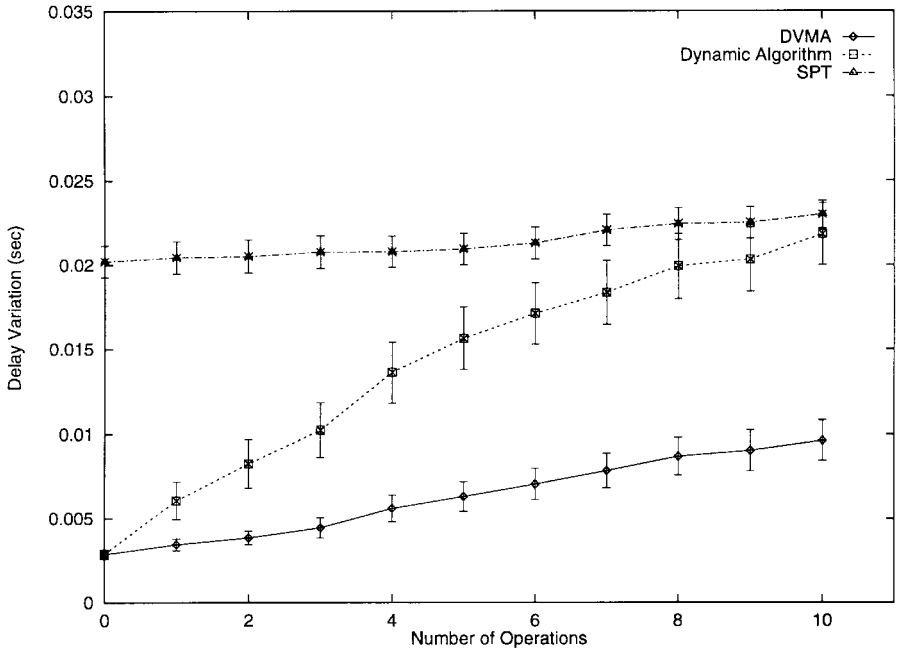


Fig. 8. Algorithm comparison for dynamic reorganization of the multicast tree (100-node networks, average node degree is three, initial group size is ten, 75% join, and 25% leave operations).

number of nodes and/or 2) the number of incoming/outgoing links at each node is relatively large.

We now compare the behavior of three approaches to reorganizing the multicast tree in response to changes in group membership. Our objective is to investigate how the value of δ_T changes over time as nodes join or leave the group. The first approach, denoted by the label "Dynamic Algorithm" in Figs. 8 and 9, is to make incremental and localized changes to the existing tree to accommodate additions or deletions of destinations, as described in Section V. The second approach is to run DVMA anew each time the multicast group changes. The third approach is to use a new SPT whenever a node is added to or deleted from the multicast group. We present results for 100-node networks, average nodal degree equal to three, and an initial multicast group of size ten. The initial tree for the first two approaches was constructed using DVMA. A total of ten join or leave requests was performed for each group. In a leave request, the destination to be deleted was selected with equal probability among the nodes in the group, while in a join request the receiver to be added was also

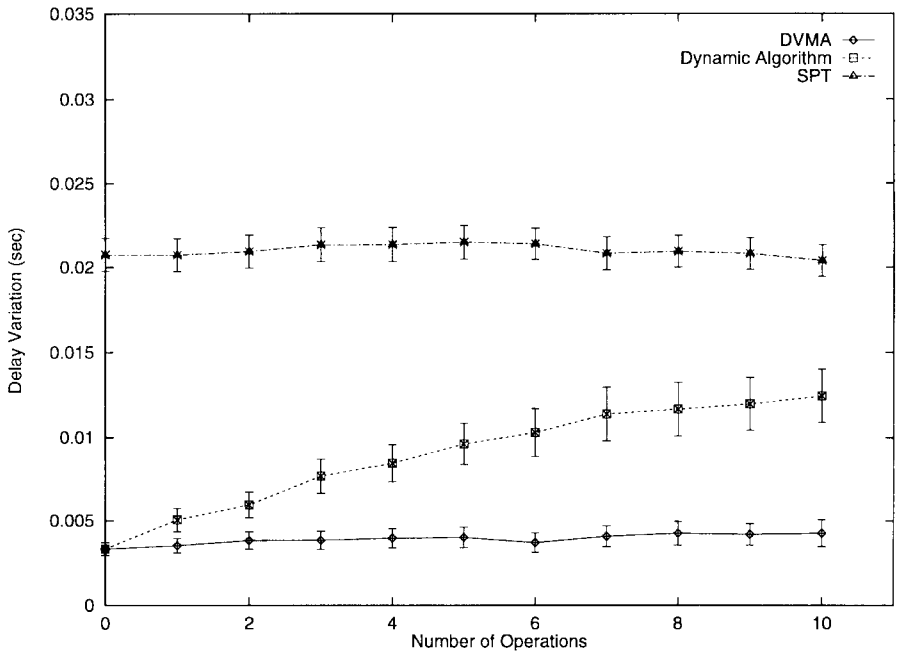


Fig. 9. Algorithm comparison for dynamic reorganization of the multicast tree (100-node networks, average node degree is three, initial group size is ten, 50% join and 50% leave operations).

selected with equal probability among the nodes that were not part of the destination set. Figs. 8 and 9 plot the value of δ_T for the trees after each join or leave request. The value plotted at point zero of the x axis corresponds to the value of δ_T for the initial group before any nodes are added or deleted. In Fig. 8, a request was chosen as either a join or leave request with probability 0.75 and 0.25, respectively, independently of previous requests. These probabilities were both equal to 0.5 in the scenario of Fig. 9.

DVMA always constructs trees with values of δ_T lower than those of trees constructed through incremental changes. This improved performance is achieved at the expense of constructing a totally new tree. On the other hand, by making incremental changes to the tree, the value of δ_T is always lower than that of SPT. Our results suggest that, if the number of *join* operations is not very large, the dynamic approach of Section V performs reasonably well. However, as the number of nodes added to the multicast group increases, it may be necessary to periodically run DVMA anew to keep the value of δ_T low.

VII. Concluding Remarks

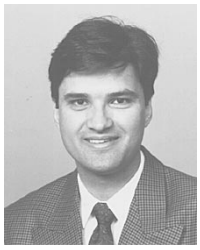
We considered the problem of determining multicast trees that guarantee certain bounds on the end-to-end delays from the source to the destination nodes, as well as on the variation among these delays. The problem of constructing such trees is NP-complete, and we developed a heuristic that exhibits good average case behavior. The heuristic performs well under conditions typical of multicast scenarios in high-speed networks, namely, when the network is not too sparse and when the size of the multicast group is small compared to the total number of nodes. The strategy employed by the heuristic is also applicable to the problem of reorganizing the tree in response to changes in the multicast group membership.

Our heuristic does not attempt to optimize the multicast tree in terms of cost (bandwidth consumption). In fact, since its strategy for satisfying the interdestination delay variation constraint is to select longer paths for some of the destination nodes, the cost of the final tree may be somewhat high. One straightforward approach to addressing the cost issue is to modify the heuristic to: 1) return the least cost tree among the feasible trees it constructs and/or 2) restrict the search space by excluding from consideration candidate trees of high cost. Techniques similar to the ones used to construct constrained Steiner trees [8], [16] might also be applicable here. The specification and analysis of algorithms that minimize the tree cost subject to delay and delay variation constraints should be explored in future research.

References

[1] D. Bertsekas and R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice Hall, 1992.
[2] K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Trans. Commun.*, vol. COMM-31, no. 3, pp. 343–351, Mar. 1983.
[3] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.*, vol. 1, pp. 269–271, 1959.
[4] M. R. Garey, R. L. Graham, and D. S. Johnson, "The complexity of computing steiner minimal trees," *SIAM J. Appl. Math.*, vol. 32, no. 4, pp. 835–859, June 1977.
[5] M. R. Garey and D. S. Johnson, *Computers and Intractability.* New York: Freeman, 1979.
[6] E. N. Gilbert and H. O. Pollak, "Steiner minimal tree," *SIAM J. Appl. Math.*, vol. 16, 1968.
[7] S. L. Hakimi, "Steiner's problem in graphs and its implications," *Networks*, vol. 1, pp. 113–133, 1971.
[8] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 286–292, June 1993.

[9] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica*, vol. 15, pp. 141–145, 1981.
[10] E. Lawler, *Combinatorial Optimization: Networks and Matroids.* New York: Holt, Rinehart and Winston, 1976.
[11] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, Nov. 1957.
[12] H. Salama, D. Reeves, Y. Viniotis, and T. Sheu, "Comparison of multicast routing algorithms for high-speed networks," IBM, Tech. Rep. IBM-TR29.1930, Sept. 1994.
[13] ———, "Evaluation of multicast routing algorithms for distributed real-time applications in high-speed networks," in *Proc. 6th IFIP Conf. High Speed Networks*, Sept. 1995, pp. 27–42.
[14] J. S. Turner, "New directions in communications (or which way to the information age?)," *IEEE Commun. Mag.*, vol. 24, no. 10, pp. 8–15, Oct. 1986.
[15] B. W. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.
[16] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, "A source-based algorithm for near-optimum delay-constrained multicasting," in *Proc. IEEE Infocom '95*, Mar. 1995, pp. 377–385.



George N. Rouskas (S'92–M'95) received the Diploma in Electrical Engineering from the National Technical University of Athens (NTUA), Athens, Greece, in 1989, and the M.S. and Ph.D. degrees in computer science from the College of Computing, Georgia Institute of Technology, Atlanta, in 1991 and 1994, respectively.

He joined the Department of Computer Science, North Carolina State University, Raleigh, in August 1994 as an Assistant Professor. His research interests include lightwave network architectures, multicast communication, high-speed networks, and performance evaluation.

Dr. Rouskas received the 1995 Outstanding New Teacher Award from the Department of Computer Science, North Carolina State University. He was also the recipient of the 1994 College of Computing Graduate Research Assistant Award. He is a member of the ACM and of the Technical Chamber of Greece.

Ilia Baldine was born in Dubna, Moscow Region, Russia, in 1972. He received the B.S. degree in computer science from the Illinois Institute of Technology, Chicago, in 1993 and the M.S. degree in computer science from North Carolina State University, Raleigh, in 1995. He is currently working on the Ph.D. degree in computer science at North Carolina State University, Raleigh.

His research interests include all-optical networks, ATM networks, network protocols, and security.