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

George N. Rouskas *Ilia Baldine*
Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206

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 (a) bounded delay along the paths from the source to each destination, and (b) bounded variation among the delays along these paths. We first establish that the problem of determining such a constrained tree is $\mathcal{NP}$-complete. We then derive heuristics that demonstrate good average case behavior in terms of the maximum inter-destination delay variation of the final tree. We also show how 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.

1 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 distributed systems [1]. One of the core issues that needs to be addressed as part of providing such mechanisms is the issue of routing, which primarily refers to the determination of a set of paths to be used for carrying the messages from the source to the destinations nodes. For reasons related to the efficient use of network resources, 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 computing multicast trees has received considerable attention in the past, and several algorithms have been proposed based on a number of optimization goals. 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 [2], and finding such a tree is a well-known $\mathcal{NP}$-complete problem [3]. Heuristics to construct low cost trees have been developed in [4, 5, 6, 7].

While total tree cost as a measure of bandwidth efficiency is certainly an important parameter, it is not sufficient to characterize the quality of the tree as perceived by interactive multimedia and real-time applications. Networks supporting real-time traffic need 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. Heuristics to compute low-cost trees which guarantee a bound on the end-to-end delay are presented in [8, 9].

In this work we consider an additional criterion to characterize the quality of the multicast tree for interactive, real-time applications. In particular, we assume that the multicast tree must guarantee bounds on the *variation* among the delays along the individual source-destination paths. 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 paths of the tree was one of the performance metrics included in a simulation study of existing multicast algorithms in [10].

There are several situations in which the need for bounded variation among the end-to-end delays arises. During a teleconference, it is important that the current 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. Consider also the use of multicast messages to update multiple copies of a replicated data item in a distributed database system. Minimizing the delay variation in this case would minimize the length of time during which the database is in an inconsistent state. Finally, being able to look at 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 the 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.

Section 2 presents a network model for multicast communication, and in Section 3 we show that the problem of constructing trees to guarantee a bound on the variation of the end-to-end delays is $\mathcal{NP}$-complete. In Section 4 we develop heuristic algorithms, and outline an approach to dynamically reorganizing the initial tree. We present numerical results in Section 5, and conclude the paper in Section 6.

2 Network Model for Multicasting

We consider the routing of multicast connections in a packet-switched communication network. The network is represented by a weighted directed graph $G = (V, A)$, where V denotes the set of nodes, and A, the set of arcs, corresponds to the set of communication links connecting the various nodes.

*This work was supported in part by a Faculty Development and Research grant, North Carolina State University.

We will use $n =\mid V \mid$ to refer to the number of nodes in the network. We define a *link-delay function* $\mathcal{D} : A \rightarrow \mathcal{R}^+$ which assigns a non-negative weight to each link in the network. The value $\mathcal{D}(\ell)$ associated with link $\ell \in A$ is a measure of the total delay that packets experience on that link, including the queueing, transmission, and propagation components.

Under the multicast routing scenario we are considering, packets originating at *source* node $s \in V$ have to be delivered to a number of destinations. We will call the set $M \subseteq V - \{s\}$ of destination nodes the *destination set* or *multicast group*, and will use $m =\mid M \mid$ to denote its size. Several multicast sessions may proceed concurrently within the network, each characterized by a source node and a destination set.

We assume that communication in the network is connection-oriented, and that multicast connections are established by issuing a *connect request*; similarly, at the conclusion of a session a *disconnect request* is issued. In response to a connect request, and prior to any data been transferred from the source to the destinations, a connection establishment process is initiated. Central to the connection establishment is the determination of routes between the source and the destinations, over which multicast packets will be carried.

Let s and M be the source and multicast group, respectively, of a certain multicast session. Multicast packets for this session 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 (that is, $M \cup \{s\} \subseteq V_T$). In addition, V_T may contain *relay* nodes, that is, nodes intermediate to the path from the source to a destination. Relay nodes are not consumers of multicast packets; rather, they simply forward these packets along the downstream links of the tree.

Let T be a multicast tree for the source-multicast group pair (s, M), and let $P_T(s, v)$ denote the unique path from source s to destination $v \in M$ in the tree T. Multicast packets from s to v experience a total delay of $\sum_{\ell \in P_T(s,v)} \mathcal{D}(\ell)$ along this path. We now introduce two parameters that relate the end-to-end delays along individual source-destination paths to the desired level of quality of service required by the application performing the multicast:

- *Source-destination delay tolerance*, Δ, representing 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.
- *Inter-destination delay variation tolerance*, δ, the maximum difference between the end-to-end delays along the paths from the source to any two destination nodes that can be tolerated. 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. The application will proceed only if a tree satisfying these constraints can be found; otherwise, the application will abort. In the following section we take a closer look at the problem of determining multicast trees that guarantee a desired level of performance in terms of the quality of service criteria discussed above.

3 Delay Variation Bounded Multicast Trees

Let Δ and δ be the delay and delay variation tolerances, respectively, as specified by a higher level application that wishes to initiate a multicast session. Our objective is to determine a multicast tree such that delays along all source-destination paths in the tree are within the two tolerances. This problem, which we will call the *Delay- and Delay Variation-Bounded Multicast Tree (DVBMT)* problem, can be naturally expressed as a decision problem:

Problem 3.1 (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, does there exist a tree $T = (V_T, A_T)$ spanning s and the nodes in M, such that:*

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

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

We will refer to (1) as the *source-destination delay constraint*, while (2) will be called the *inter-destination delay variation constraint*. We will also say that tree T is a *feasible* tree for a multicast session with source s and destination set M, if and only if T satisfies both (1) and (2). Note that, in order for the multicast session to proceed, it is necessary and sufficient that a *single* feasible tree be constructed, as *any* feasible tree can meet the requirements expressed by Δ and δ.

The source-destination delay constraint (1) has been previously considered in the context of designing constrained Steiner trees for real-time, interactive applications [8, 9], but we are not aware of any work that explicitly considers the inter-destination delay variation constraint (2) in the construction of multicast trees. However, as part of a recent study [10] to evaluate the relative performance of a large number of multicast algorithms and their suitability to high-speed real-time applications, the following quantity was measured and used as a criterion in the evaluation:

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

Quantity δ_T is the maximum inter-destination delay variation in tree T, and, given a value for δ, it can be used to determine whether tree T can meet the quality of service requirements of the application. 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 [10] takes the delay variation constraint (2) into account. Our work addresses the problem of designing multicast algorithms that overcome this inefficiency.

Before proceeding, we would like to resolve the open question regarding the existence of efficient algorithms for *DVBMT*. Unfortunately, the following theorem establishes that *DVBMT* is $\mathcal{NP}$-complete; its proof is given in Appendix A. The next section presents a heuristic approach to determining feasible trees for arbitrary instances of *DVBMT*.

Theorem 3.1 DVBMT *is $\mathcal{NP}$-complete whenever the size of the multicast group* $\mid M \mid \geq 2$.

4 Multicast Tree Algorithms for *DVBMT*

Consider an application running at node s, and suppose that the application issues a request for establishing a multicast connection with destination set M. Along with the request, the application also supplies values for the path delay tolerance Δ, and inter-destination delay variation tolerance δ. As part of the connection establishment process, a multicast tree satisfying constraints (1) and (2) needs to be determined. In this section we present algorithms that can be used to construct such a tree. Our algorithms operate under the assumption 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 one of several existing topology-broadcast algorithms [11].

The sequence of actions taken by node s during the course of constructing a multicast tree is illustrated in the flowchart of Figure 1, where we have assumed that the values of the delay and delay variation tolerances Δ and δ, respectively, provided by the application are negotiable. As a first step, the tree T_0 of shortest paths [12] from s to all nodes in M is constructed. If T_0 does not satisfy the path delay constraint (1) no tree may satisfy it, implying that the delay tolerance Δ is too tight; negotiation may then be necessary to determine a looser value of 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 T_0 is a feasible tree for this instance of the *DVBMT* problem, and the multicast session may take place over the tree of shortest paths. As a result, the route determination phase completes successfully, and the connection establishment process may then proceed to a subsequent phase (such as bandwidth reservation, etc.).

It is possible, though, that tree T_0 fail to satisfy constraint (2). Our approach then is to have the source execute a search algorithm in an attempt to construct a new tree satisfying both (1) and (2). Since *DVBMT* is $\mathcal{NP}$-complete, however, the search algorithm has to employ a heuristic approach. Nevertheless, suppose that a heuristic algorithm is available, and that it returns a tree which constitutes a solution to the given instance of the *DVBMT* problem; then a tree for the multicast session has been found.

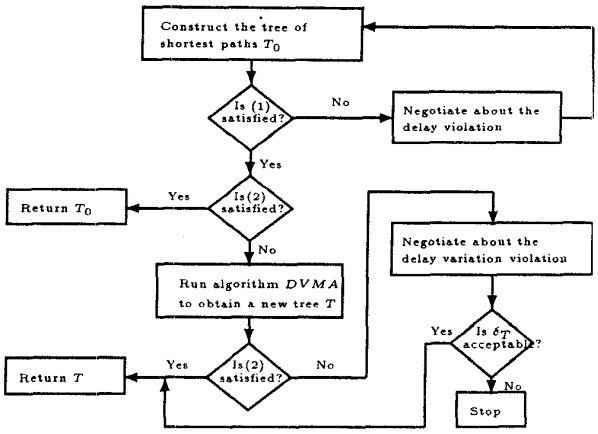

Figure 1: Obtaining a tree for the *DVBMT* problem

However, a heuristic algorithm may fail to discover a feasible tree, either because no such tree exists or because of the ineffectiveness of the search strategy employed. Other than abandoning the connection altogether, the only course of action available at that point would be to determine a new value for the delay tolerance δ that would be acceptable to all parties involved in the multicast session. If such a value can be agreed upon the source would go through another iteration in the flowchart of Figure 1, otherwise the multicast session would have to be abandoned.

An alternative that would result in a considerable speed-up of the negotiation process would be to design the search algorithm so that it always returns, among the trees considered, the one with the smallest value of δ_T in (3). Indeed, regardless of whether a solution to the given instance of *DVBMT* problem exists or not, the tree corresponding to the smallest value of δ_T is the best tree that can be obtained with the search algorithm at hand. If this tree is available at the termination of the algorithm, all that has to be determined during the negotiation process is whether an acceptable level of quality of service can be sustained for the given value of δ_T and there is no need to repeat the route determination process; this is shown in Figure 1.

The following subsection presents a new multicast tree heuristic designed to solve the *DVBMT* problem. Following that, we show how the to develop a solution to the *dynamic* problem of updating the tree in response to receiver requests for joining or leaving an ongoing multicast session.

4.1 Delay Variation Multicast Algorithm

Let T_0 be the tree of shortest paths from source s to the nodes in the destination set M for the multicast connection under consideration. Let us also assume that T_0 meets the delay requirement (1), but that it does not meet the delay variation requirement (2). The *Delay Variation Multicast Algorithm (DVMA)*, described in detail in Figure 2, can then be used

to search through the space 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 (a) satisfies the delay constraint (1) and (b) has the least value of δ_T *among the trees considered by the algorithm.* The basic idea behind the operation of *DVMA* is now described.

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. *DVMA* operates by appropriately augmenting tree T to eventually include all nodes in U; to this end, it 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 Step 2 above is one which, if connected to T in Step 3, the resulting tree T' would be a feasible tree for the subset of the set of destination nodes it contains. In order 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, in order to guarantee 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 inter-destination 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 for which (a) the total delay from s to u is at most Δ, and (b) 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.

To see how an initial tree T is constructed, consider T_0, the tree of shortest paths, and 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, the only alternative to constructing a feasible tree is to find longer paths from s to some or all of the other destination nodes. Hence, our approach is to start with an initial 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 *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. Therefore, 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. In the latter case, the algorithm will return the tree with the smallest value of δ_T in (3). The details of *DVMA* can be found in Figure 2.

The correctness of *DVMA* is provided by the following lemma. Note, however, that although the algorithm returns the best tree, in terms of maximum delay variation, that it can find, because of its heuristic nature it may fail to discover a feasible tree for the given value of δ even if one exists.

Lemma 4.1 (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 in 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 as 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 destinations in M are in T, our first claim is true.

That the delay constraint (1) is satisfied by the final tree T is now is easy to see. If $T = T_0$ this is true by hypothesis; if $T \neq T_0$ this is also true as no path is ever added to any tree T_i unless the delay constraint 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 by the algorithm. □

The next lemma determines the complexity of *DVMA*.

Lemma 4.2 *The worst-case complexity of* DVMA *is $\mathcal{O}(klmn^4)$, where k is the number of paths at line 3 of Figure*

Delay Variation Multicast Algorithm (DVMA)

(T_0 is the tree of shortest paths, and $w \in M$ is 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, ..., 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 ∈ V_i
6.        Let U = M − (M ∩ V_i) be the set of destinations not yet connected to the tree T_i
7.        while U ≠ φ do
8.           Pick any node u ∈ U          // will connect u to the tree T_i
9.           for each node v ∈ V_i do          // find a path from v to u
10.             Construct a new graph G' from the initial graph G by excluding all nodes in V_i − {v} and all links in A_i
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 ∈ V_i loop
14.          Select the best path q among all paths q_v, v ∈ 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 ∩ 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
```

Figure 2: Heuristic algorithm for the *DVBMT* problem

2, l the number of paths at line 11, m is the size of the multicast group M, and n 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 j-th iteration of the "while" loop. Then, the innermost loop starting at line 9 will iterate t_j times; inside this loop the complexity is determined by the l-shortest path algorithm at line 11, which takes time $\mathcal{O}(lN^3)$ [13] 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 easy 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, we conclude that the complexity of the algorithm is, in the worst case, $\mathcal{O}(klmn^4)$. □

Regarding parameters k and l, note that the maximum value they 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 loose, 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.

4.2 Reorganization of the Multicast Tree

During connection establishment, *DVMA* can be used to construct a feasible tree for a given destination set. For certain applications, however, 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*. Similarly, 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 to ensure that constraints (1) and (2) are satisfied at all times.

Let T be the initial tree for 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. Note that 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'. 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 adopt a different strategy, one that attempts to minimize both the cost incurred during the transition period, and the disruption caused to the receivers. More specifically, the multicast tree is never modified unless it is absolutely necessary to do so. Even then, the new tree is not computed from scratch, rather, a feasible tree for the new multicast group is constructed by making incremental and localized changes to the old tree. We now describe in detail how the join and leave requests are handled under our approach.

Let us first consider leave requests, and 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 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 in order to avoid wasting bandwidth, tree T has to be pruned to exclude v and, possibly, relay nodes and links 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. We conclude that leave requests are easy to handle, and no destination node (other than v) needs to be disrupted.

Let us now turn our attention to the actions taken whenever a node $u \notin M$ announces its intention to join the multicast group. We distinguish three cases, as follows. First, suppose that $u \notin V_T$, i.e., the new node is not part of the multicast tree T. Our approach is to augment T to include a path from a node $V \in V_T$ to the new node u. This can be easily accomplished by letting $T_i = T$ and $U = \{u\}$ at lines 5 and 6, respectively, of *DVMA* (see Figure 2), 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 new path and does not affect any of the paths from the source to nodes already in the multicast group [1].

Now suppose that $u \in V_T$, 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 $M' = M \cup \{u\}$ [2]. Tree T is then a feasible tree for the 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 downstream nodes.

Finally, let $u \in V_T$, but the path from s to u be such that the delay variation 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). Finding a new path from s to u will definitely affect the paths to these nodes, however, the paths to nodes in $M - W$ need not be affected. 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* in Figure 2. We then execute the code between lines 7 and 17 to connect the destination nodes in U into tree T_1. As a result, packets will be routed from s to the nodes in W over new paths in the final tree T', but none of the paths to nodes in $M - W$ will change.

As a final observation, besides being minimally disruptive, this approach has the additional advantage that the algorithm used during set-up time to construct an initial tree for the multicast connection, can also be used to reorganize the tree during the lifetime of the session.

5 Numerical Results

We now consider five different algorithms that can be used to construct multicast trees for a given source and destination set, and compare their performance in terms of the maximum delay variation δ_T among the source-destination paths in the final tree T, as defined in (3). The five algorithms studied are: (1) *DVMA*, the algorithm described in Figure 2. We run this algorithm with $\Delta = 0.05s$ and $\delta = 0$. This value of δ was used in order to force the algorithm to go through all possible iterations of the outer for loop and return the tree with the smallest value of δ_T it can find; (2) *DVMA2*, an algorithm very similar to *DVMA*; it differs from the latter in the way the graph G' is constructed at line 10 of Figure 2. More specifically, in addition to excluding all nodes in $V_i - \{v\}$ and all links in A_i, all the nodes in $U - \{u\}$ and their adjacent links are also excluded from the initial graph G. The values of parameters Δ and δ used are the same as for *DVMA* above; (3) Dijkstra's algorithm [12] which constructs the tree of shortest paths (*SPT*) from the source to any node in the network; (4) Prim's algorithm [14] which constructs a tree of minimum weight (MST) spanning all nodes in the network; the weight of each link is set to the delay incurred along the link; (5) The *tradeoff* (TDF) algorithm [5] between the minimum spanning tree *heuristic* [7] and *SPT*, considered here because it was conjectured in [10] that it may yield good performance in terms of δ_T.

We have studied the *average case* behavior of the five algorithms by generating random graphs for a wide range of values for the total number n of nodes, the average degree of each node, and the number m of destinations in the multicast group as a percentage of n. The graphs were constructed to resemble real-world networks using the method described in [4]; the nodes graphs were placed in a grid of dimensions 4900 × 4900 Km, and the delay for each link was set to the propagation delay of light along that link. Figures 3 - 5 plot δ_T against the number of nodes n in the network, for the five algorithms discussed above (other results can be found in [15]). Each point plotted represents the average over three hundred different graphs for the stated values of n, m, and the average degree of each node.

[1] If this fails to discover such a path, there are two possible courses of action: (a) run *DVMA* from scratch for the new multicast group, or (b) deny node u its participation in the multicast session; which course of action to be taken may depend on several factors, such as the nature of the application, the cost of rerouting the connection, etc.

[2] The path from s to u will satisfy (1), as u cannot be a leaf in T.

Our results suggest that *DVMA* and *DVMA2* achieve the best performance among the five algorithms (with *DVMA2* outperforming *DVMA* in most cases), and achieve an improvement of up to an order of magnitude over the tree of shortest paths *SPT* which exhibits the next best performance. Contrary to the expectations expressed in [10], the tradeoff algorithms constructs trees with maximum delay variation larger than that of *SPT*. The *MST* is by far the worst tree in terms of δ_T, but this should be expected as Prim's algorithm minimizes the *total* weight of the tree, without paying any attention to individual paths. As the size m of the multicast group increases as a percentage of the size n of the network (compare Figures 3 and 4), the improvement over the *SPT* achieved by our algorithms decreases; results in [15] show that when m is larger than 25-30% of n, it is preferable to simply use *SPT* rather than running *DVMA* or *DVMA2*. On the other hand, the larger the average nodal degree, the better the performance of our algorithms, as Figure 5 illustrates.

Overall, our algorithms achieve their best performance under conditions that are typical of multicast applications running in high speed networks, namely, when (a) the size of the multicast group is relatively small compared to the total number of nodes in the network, and/or (b) the number of incoming/outgoing links at each node is relatively large.

6 Concluding Remarks

We have considered the problem of determining multicast trees that guarantee certain bounds on the end-to-end delays from the source to the each of the destination nodes, as well as on the variation among these delays. After establishing that the problem of constructing such constrained trees is $\mathcal{NP}$-complete, we developed heuristics that exhibit good average case behavior, especially under conditions typical of multicast scenarios in high-speed networks. We have also shown that the strategy employed by the heuristic is applicable to the problem of reorganizing the tree in response to changes in multicast group membership.

References

[1] J. S. Turner. New directions in communications (or which way to the information age?). *IEEE Communications Magazine*, 24(10):8–15, October 1986.

[2] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[3] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing steiner minimal trees. *SIAM J. of Applied Mathematics*, 32(4):835–859, June 1977.

[4] B. W. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[5] K. Bharath-Kumar and J. M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Transactions on Communications*, COM-31(3):343–351, March 1983.

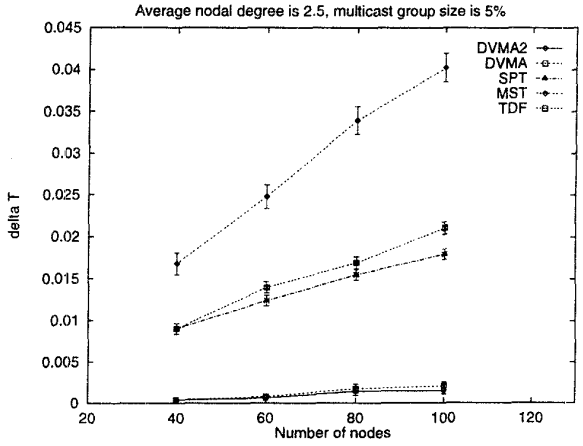

Figure 3: Algorithm comparison (first set of graphs)

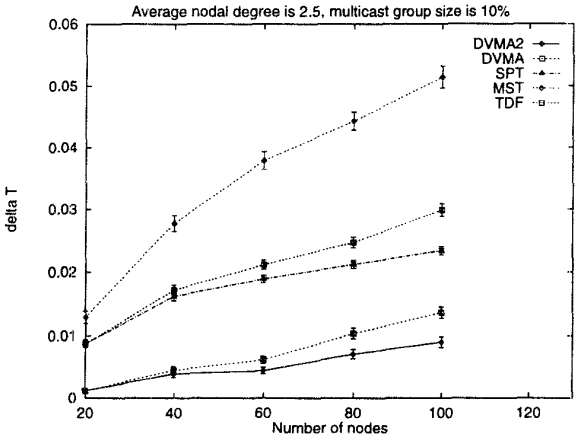

Figure 4: Algorithm comparison (second set of graphs)

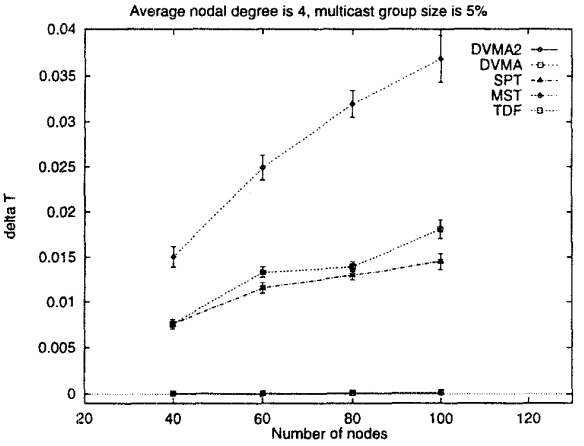

Figure 5: Algorithm comparison (third set of graphs)

[6] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15:141–145, 1981.

[7] E. N. Gilbert and H. O. Pollak. Steiner minimal tree. *SIAM Journal on Applied Mathematics*, 16, 1968.

[8] V. Kompella, J. Pasquale, and G. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Trans. Networking*, 1(3):286–292, June 1993.

[9] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for near-optimum delay-constrained multicasting. *IEEE Infocom*, March 1995.

[10] H. Salama, D. Reeves, Y. Viniotis, and T. Sheu. Evaluation of multicast routing algorithms for distributed real-time applications in high-speed networks. In *Proc. of 6th IFIP Conf. on High Speed Networks*, Sep. 1995.

[11] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, Inc., 1992.

[12] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[13] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[14] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389–1401, Nov 1957.

[15] G. N. Rouskas and I. Baldine. Multicast routing with inter-destination delay variation constraints. Technical Report TR-95-09, NCSU, Raleigh, NC, 1995.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., New York, 1979.

A Proof that *DVBMT* is $\mathcal{NP}$-complete

We now show that problem *DVBMT* is $\mathcal{NP}$-complete. The proof uses a transformation from *PARTITION* [16], an $\mathcal{NP}$-complete problem repeated here for the sake of completeness.

Problem A.1 (PARTITION) *Given a set of k elements $S = \{1, 2, \ldots, k\}$ with a_i the weight of element i, and $A = \sum_{i=1}^{k} a_i$, does there exist a partition of S into two sets, S_1 and S_2, such that $\sum_{i \in S_1} a_i = \sum_{j \in S_2} a_j = \frac{A}{2}$?*

Proof (of Theorem 3.1). *DVBMT* is in the class $\mathcal{NP}$, since a nondeterministic algorithm need only guess a tree spanning s and the nodes in the destination set M, and verify in polynomial time that the tree satisfies both (1) and (2).

We now transform *PARTITION* to *DVBMT*; note that it is sufficient to find a transformation for the case $\mid M \mid = 2$. Let $S = \{1, 2, \ldots, k\}$ be the set of elements of weights $a_i, i = 1, \ldots, k$, 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 Figure 6). The network $G = (V, A)$ has $n =$

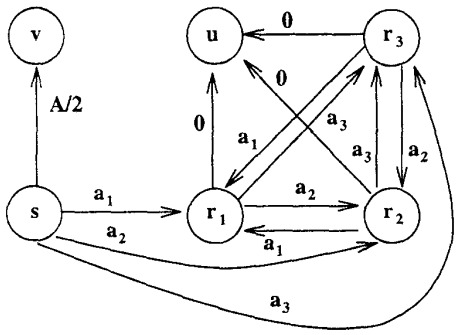

Figure 6: Instance of *DVBMT* corresponding to an instance of *PARTITION* with $S = \{1, 2, 3\}$

$k + 3$ nodes and $V = \{s, v, u, r_1, r_2, \ldots, r_k\}$ (s is the source and $M = \{v, u\}$ is the destination set). The set A of links is:

$$\begin{aligned} A \;=\; & \{(s,v),(s,r_1),\ldots,(s,r_k),(r_1,u),\ldots,(r_k,u) \\ & (r_1,r_2),\ldots,(r_1,r_k),\ldots,(r_k,r_1),\ldots,(r_k,r_{k-1})\} \end{aligned} \quad (4)$$

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 $r_j, i, j = 1, \ldots, k, i \neq j$ (i.e., the subgraph of G containing only nodes $r_i, i = 1, \ldots, k$, is a complete graph). There is only one path from s to destination node v consisting of the single link (s, v); a path from s to the other destination u may contain any number of the nodes $r_i, i = 1, \ldots, k$, and in any order (see Figure 6). The link-delay function $\mathcal{D}$ is now 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 $S_1 = \{a_{\pi_1}, \ldots, a_{\pi_l}\}$ for some $l < k$. The tree consisting of path (s, v) and path $(s, r_{\pi_1}), (r_{\pi_1}, r_{\pi_2}), \ldots, (r_{\pi_{l-1}}, r_{\pi_l}), (r_{\pi_l}, u)$, is then a feasible tree as the delay along both paths is equal to $\frac{A}{2}$.

Conversely, let T be a feasible tree for *DVBMT*. T includes the path (s, v) of delay $\frac{A}{2}$, as this is the only path from the source to v. Let $(s, r_{\pi_1}), (r_{\pi_1}, r_{\pi_2}), \ldots, (r_{\pi_{l-1}}, r_{\pi_l}), (r_{\pi_l}, u)$, 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 $\frac{A}{2}$, and $l < k$ (for if $l = k$, the path from s to u would include all $r_i, i = 1, \ldots, k$, and the delay along the path would equal A, contradicting our hypothesis that T is a feasible tree). Then, $\sum_{i=1}^{l} a_{\pi_i} = \frac{A}{2}$, and $S_1 = \{a_{\pi_1}, \ldots, a_{\pi_l}\}$, $S_2 = S - S_1 \neq \phi$, is a partition of S. □