Cost, Delay, And Delay Variation Conscious Multicast Routing

Brian K. Haberman   George N. Rouskas

TR-97-03

March 1, 1996

Abstract

We consider a constrained Steiner tree problem in which it is necessary to provide bounded delays between the source and all destinations, as well as bounded variation among these delays. The delay and delay variation bounds are directly related to the quality of service requirements of real-time interactive applications that will use the tree for multicast communication. The main contribution of our work is a new strategy for constructing constrained multicast trees. We present a heuristic that exhibits good average case behavior for the maximum delay variation among the destinations while keeping the overall cost of the multicast tree low. Our heuristic compares favorably with existing multicast algorithms in terms of tree cost, system running time, and the satisfaction of the delay and delay variation constraints.

Department of Computer Science
North Carolina State University
Raleigh, NC 27695

1 Introduction

The recent advances in computer networking technology have led to the development of new network-centric applications. These applications have varying requirements for the underlying network. One of the main requirements is the ability to concurrently send messages to multiple users. Consequently, mechanisms to support *multicast* communication are becoming an important aspect in the design of distributed systems [1, 2]. One important issue in providing such mechanisms is that of routing packets from a user to multiple destinations. Typical approaches to multicast routing require the transmission of data along the branches of a *multicast tree* spanning the source and destination nodes.

The growing interest in multicast communication has led to the development of several algorithms to generate multicast trees. Minimum Steiner tree algorithms attempt to minimize the total cost of the multicast tree, which is taken as the sum of the costs on the links of the tree. The minimum Steiner tree problem is $\mathcal{NP}$-complete [3], but several heuristics have been developed for it [4, 5, 6, 7]. More recently, a heuristic for the Steiner tree problem that includes previous heuristics as special cases was developed in [8]. The unique characteristic of this algorithm is the *control knob* feature that dictates to what degree the algorithm will be concerned about tree cost versus runtime efficiency.

With the advent of real-time interactive applications, attention has been given not only to minimizing total tree cost, but also to providing 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 minimizing tree cost under the constraint that all path delays are within a user-specified delay bound is referred to as the *delay constrained* Steiner tree problem, and it is also $\mathcal{NP}$-complete. Heuristic solutions to this problem can be found in [9, 10, 11].

There are certain classes of applications that would benefit if, in addition to an end-to-end delay bound on individual paths, a bound on the *variation* of delay among paths to different destinations in the multicast tree were also imposed. For interactive applications with humans as the end-users, providing a guarantee on delay variation would be extremely helpful in maintaining the feeling of a face-to-face discussion: with video-conferencing now available on the Internet [12], it is desirable for all of the viewers to see and hear the speaker at (almost) the same time. For other applications, high levels of variability in path delays can severely affect consistency. In an on-line software distribution system, the primary host will want to guarantee that all the remote hosts get a new copy of software at the same time so that the sites never lose integrity, while in virtual world environments [13] the distributed application must maintain consistency among the different users. Finally, minimizing the delay variation may be necessary in order to reduce the competitive advantage a destination gains by being able to process a message sooner than another destination (as in a distributed game scenario where a game server multicasts information to competing users).

An algorithm that takes into account both delay and delay variation when constructing multicast trees is presented in [14]. Given user-supplied values for these parameters, this algorithm attempts to construct *some* tree such that path delays are within these bounds. However, the algorithm is not concerned with optimizing in terms of the overall tree cost, and thus, it may return a tree of high cost. The main contribution of this work is an extension of the basic philosophy of the algorithm in [14] that balances the delay and delay variation requirements of the application against the network requirement for a low cost tree.

It could be argued that it is possible to use buffering (at the source, the routing nodes, the receivers, or a combination of the three) to combat the problem of inter-destination delay variation in a multicast tree. Buffering at the source, however, would require the source to maintain additional information about all destinations. Furthermore, the benefits of using a multicast tree for routing would disappear since each message would have to be buffered a different amount of time for each destination (the sender would incur the additional cost of transmitting the same message multiple times). Buffering at the switching nodes suffers from similar problems. Buffering at the receivers, on the other hand, is a straightforward approach, but it requires that each receiver cooperate with the other members of the multicast group. In the software distribution example above, this approach is feasible, but in a scenario where the receivers are competing against one another, as in a distributed game, the network cannot rely on the receivers to delay messages. Additionally, the amount of buffering needed in this approach is directly proportional to the maximum variation of end-to-end delays. By providing bounds for the delay variation, more efficient usage of buffer space will result. We believe that, when possible, buffering at the receivers may be used along with our routing algorithm to more successfully address the problems caused by the variation in end-to-end delays.

The rest of this paper is organized as follows. Section 2 presents the network model, and formally defines the constrained Steiner tree problem we consider. Section 3 describes the heuristic algorithm we developed for this problem. Numerical results are presented in Section 4 and our conclusions are in Section 5.

2 Network Model and Constrained Multicast Trees

We consider the problem of generating communication paths through a packet-switched network for multicast traffic. We model the network as a simple [1], connected, directed graph $G = (V, E)$, where V denotes the set of nodes in the network, and E represents the set of edges. These edges correspond to the communication links between the nodes in the network. The existence of a link $\ell = (u, v)$ from u to v implies the existence of a link $\ell' = (v, u)$ for any $u, v \in V$. An edge $\ell \in E$ has two weights associated with it. The non-negative value $\mathcal{D}(\ell)$ represents the delay that data packets experience on link ℓ, and it is the sum of the switching, queueing, transmission, and propagation delays. The non-negative value $\mathcal{C}(\ell)$ represents the cost of using link ℓ to carry multicast traffic; typically, the cost is a measure of the link utilization. The network can be asymmetric, hence it is possible that $\mathcal{C}(\ell) \neq \mathcal{C}(\ell')$ and/or $\mathcal{D}(\ell) \neq \mathcal{D}(\ell')$.

Consider a point-to-multipoint application that requires a *source node* $s \in V$ to transmit data to a *multicast group* $M \subseteq V$. In order for the communication to proceed, paths from the source node s to all members of M must be determined. For efficiency, multicast packets must be routed from s to the destinations in M via the links of a *multicast tree* $T = (V_T, E_T)$, $V_T \subseteq V$ and $E_T \subseteq E$, rooted at s and spanning all nodes $d \in M$. Nodes which are not members of M may be contained in V_T as *relay* nodes. Relay nodes cannot be leaves of the tree, and simply forward packets along adjacent links without actively participating in the multicast communication.

Let s and M be the source and multicast group, respectively, of a certain real-time interactive application, and let T be the tree used for the multicast communication. We define $P_T(s, d) \subseteq E_T$ to be the set of links in the path from s to $d \in M$ in the tree. The total delay along this path is $\sum_{\ell \in P_T(s,d)} \mathcal{D}(\ell)$. The total cost of the tree can be expressed as $\sum_{\ell \in E_T} \mathcal{C}(\ell)$. There are two different, and in many ways conflicting, objectives that have to be taken into account during the construction of a multicast tree T. These objectives correspond to the quality of service requirements of the application, on one hand, and the desire for efficient use of the network resources, on the other.

From the point of view of the application performing the multicast, the delays along source-destination paths in the tree are important in two ways. First, the delay along any individual path cannot be greater than a specified end-to-end *delay tolerance*, Δ. This parameter reflects the fact that a packet delivered more than Δ time units after its transmission at the source is of no value to the receivers (e.g., because it has missed its playout instant). Second, the difference between the end-to-end delays along any two paths $P_T(s, d_1)$ and $P_T(s, d_2)$, $d_1, d_2 \in M$, cannot exceed a specified inter-destination *delay variation tolerance*, δ. Parameter δ defines a synchronization window for

[1]The graph contains no more than one edge between an ordered pair of nodes.

the various receivers. Unless a tree T that meets these two end-to-end delay requirements can be found, the multicast application will not be able to proceed. From the network utilization point of view, however, the tree of minimum cost spanning the source s and the destination nodes in M (Steiner tree) is the best one to use. Our work addresses the problem of balancing the user (application) requirements against the network requirement.

By supplying values for parameters Δ and δ, the application in effect imposes a set of constraints on the paths of the multicast tree. Our objective is to determine a multicast tree of minimum cost such that the delays along all source-destination paths in the tree are within the two tolerances. More formally, we express the resulting *Delay and Delay Variation Constrained Steiner Tree* (DVCST) problem as follows.

Problem 2.1 (DVCST) *Given a network $G = (V, E)$, a multicast group $M \subseteq V$, a source node $s \in V$, a link delay function $\mathcal{D} : E \rightarrow \mathcal{R}^+$, a link cost function $\mathcal{C} : E \rightarrow \mathcal{R}^+$, a delay tolerance Δ, a delay variation tolerance δ, and an overall cost C, does there exist a tree $T = (V_T, E_T)$ spanning s and all the nodes in M, such that :*

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

$$\left| \sum_{\ell \in P_T(s,d_1)} \mathcal{D}(\ell) - \sum_{\ell \in P_T(s,d_2)} \mathcal{D}(\ell) \right| \leq \delta \qquad \forall d_1, d_2 \in M \tag{2}$$

$$\sum_{\ell \in E_T} \mathcal{C}(\ell) \leq \mathsf{C} \tag{3}$$

The DVCST problem can be easily seen to be $\mathcal{NP}$-complete. If constraints (1) and (2) are removed, the problem reduces to the Steiner tree problem, a well-known $\mathcal{NP}$-complete problem [3], while removing constraint (2) yields the delay constrained Steiner tree problem [9, 10]. DVCST also reduces to the DVBMT problem [14] if constraint (3) is removed. Interestingly, the DVBMT problem has also been proven to be $\mathcal{NP}$-complete [14]. To the best of our knowledge, this work is the first that takes all three objectives into account.

3 A Multicast Tree Algorithm for DVCST

We now turn our attention to obtaining a solution to the cost optimization problem corresponding to DVCST. We will call a tree that satisfies both constraints (1) and (2) a *feasible tree*. Our objective, then, is to obtain the feasible tree of minimum cost for the given values of Δ and δ. We take a source-based approach, in that we assume that it is the responsibility of the source node s to construct the multicast tree. For this purpose, node s needs information about the network topology which can be collected and updated using one of several topology-broadcast protocols [15].

We note that, if the specified delay and delay variation tolerances are too tight, a feasible tree may not exist. Furthermore, any heuristic for the DVCST problem, including the one to be presented shortly, may fail to find a feasible tree even if one exists (recall that the DVBMT problem is also $\mathcal{NP}$-complete [14]). In this case, the application may either abort or negotiate looser values for Δ and δ. To facilitate the negotiation process, our algorithm has been designed to either return the tree of least cost among the feasible trees found, or, having failed to find a feasible tree, to return the tree that comes closest to meeting the user specified tolerances. More specifically, let Δ_T (respectively, δ_T) be the maximum delay (delay variation) among the source-destination paths in some tree T. The non-feasible tree returned by our algorithm is the one with the smallest Δ_T among the trees considered. If more than one tree with the same Δ_T are found, the one with the smallest δ_T is returned. Finally, if there are multiple trees with the same Δ_T and δ_T values, the tree of least cost among them is returned. As a result, even though a solution to the DVCST problem may not be found, the best tree that can be obtained using our algorithm becomes available to the application. If this tree meets the negotiated tolerances, the source will not need to invoke the routing algorithm for a second time.

Our approach is to have the source s perform the following steps to construct a low cost feasible multicast tree, given Δ and δ:

1. Run the **CCDVMA** heuristic (see Figure 1) to obtain a tree T with maximum delay Δ_T and maximum delay variation δ_T
2. If T is a feasible tree (i.e., $\Delta_T \leq \Delta$ and $\delta_T \leq \delta$), return T and stop
3. If T fails to satisfy either constraint (1) or (2), negotiate the tolerance(s) that T violates
4. If T is a feasible tree for the new (negotiated) values of Δ and/or δ, return T and stop; otherwise, abort the multicast session

The *Cost Conscious Delay and Delay Variation Multicast Algorithm* (**CCDVMA**) is our heuristic for the DVCST problem. **CCDVMA** searches through the space of *candidate* trees (i.e., trees spanning s and the destination nodes in M) for a feasible tree of low cost. A detailed description of

CCDVMA can be found in Figure 1. The operation of **CCDVMA** is explained in the following section.

3.1 The CCDVMA Heuristic

The basic philosophy of **CCDVMA** is to start with a tree of *least cost* paths from the source to the destination nodes, and to incrementally replace paths in the tree to meet the delay and delay variation tolerances. Let T_0 be the tree of least cost paths generated by Dijkstra's algorithm [16]. Let also $d \in M$ be the destination with the highest cost path in T_0 such that (a) the delay along this path does not exceed Δ, and (b) if there are multiple destinations in the path, the delay variation does not exceed δ. (If no feasible path exists, **CCDVMA** selects the one that comes closest to meeting the delay and delay variation tolerances as explained above; we then let d be the destination at the end of this path.) We now note that any solution to the DVCST problem will include *some* path from s to d. Since the path from s to d in T_0 is the least cost one (among all paths from s to d) and also meets (or comes close to meeting) the user constraints, it is reasonable to build the multicast tree around it. Indeed, **CCDVMA** constructs an initial tree containing only this path, and it repeatedly adds more paths to destinations until all nodes in M are included in the tree.

Let us assume that a tree $T = (V_T, E_T)$ spanning s and a subset of M has been determined. Also let $U = M - (M \bigcap V_T)$ be the set of destinations not yet in T. To add a new destination node to T, **CCDVMA** performs the following three steps:

1. Select a destination node $u \in U$
2. Find a "good" path from u to some $v \in V_T$
3. Construct a new tree T' by connecting all nodes and edges in this path to T, and update U to exclude u and any other destination nodes in the path

A "good" path in Step 2 is one that, when connected to tree T in Step 3, the resulting tree T' will be feasible tree for the subset of M it contains. To find such a path **CCDVMA** generates the l least cost paths from u to some node v already in the tree T. These paths are generated on a graph G' created by excluding all nodes in V_T except v and all edges in E_T from the original graph G. Using graph G' guarantees that the addition of any of the l paths into T will not create a cycle. Among these l paths, the path of least cost such that the delay from u to s (through v) is at most Δ is selected. In order to handle the situation where no feasible tree can be found that includes

Cost Conscious Delay and Delay Variation Multicast Algorithm (CCDVMA)

M is the set of destinations, T_0 is the tree of least cost paths from the source s to $v \in M$, and $d \in M$ is such that the path from s to d in T_0 is the highest cost path with a delay at most Δ

```
BEGIN
1  $T = T_0$
2  Find the $k$ paths, $p_1, \cdots, p_k$, of least cost from $s$ to $d$
3  for $i = 1$ to $k$ do
4      if the delay from $s$ to $d$ on $p_i$ is $\leq \Delta$ then
5          Initialize $T_i = (V_i, A_i)$ to path $p_i$
6          Let $U = M - (M \bigcap V_i)$ be the set of destinations not yet connected to $T_i$
7          while $U \neq \phi$ do
8              Pick any node $u \in U$
9              $q$ is a fictitious path with infinite delay
10             for each node $v \in V_i$ do
11                 Construct a new graph $G'$ from $G$ by excluding all nodes
                   in $V_i - \{v\}$ and all links in $A_i$
12                 Find the $l$ paths of least cost from $v$ to $u$ in $G'$
13                 Of these $l$ paths, choose the least cost path such that the delay
                   from $u$ to $s$ is at most $\Delta$ (or, if no such path exists, choose
                   the path of least delay) and call it $q_v$
14                 Let $q$ be the best path (as in Section 3.1) between $q$ and $q_v$
15             end of for loop
16             Update $T_i = (V_i, A_i)$ to include all nodes and links in path $q$
17             Update $U$ to remove destinations now in $T_i$
18         end of while loop
19         Let $T$ be the best tree (see Section 3.1) among $T$ and $T_i$
20     end if
21 end of for loop
22 return $T$
END
```

Figure 1: **CCDVMA** heuristic for the DVCST problem

this path from u to v, **CCDVMA** repeats this process for all nodes $v \in V_T$. Once all possible paths from u to nodes in T have been determined, **CCDVMA** uses the following greedy rule to select the path p to include in T, resulting in tree $T'(p)$:

- if one or more feasible (for the subset of destinations they contain) trees $T'(p)$ have been found, select the tree of least cost among them;
- else, select the tree $T'(p)$ with the smallest maximum delay $\Delta_{T'(p)}$, or, if multiple such trees exist, select the one with the smallest maximum delay variation $\delta_{T'(p)}$.

Finally, we observe that the least cost feasible tree for the given multicast application may *not* include the least cost path from s to d, but it will include *some* path from s to d. To account for this possibility, **CCDVMA** builds up to k multicast trees, each starting with one of the k least cost paths from s to d with a delay that does not exceed Δ, and it picks the best tree among them according to the above greedy rule. The details of **CCDVMA** can be found in Figure 1. It can be easily shown that the algorithm always returns a multicast tree. If some of the multicast trees it constructs are feasible, the least cost one among them is returned. If all trees are non-feasible, the tree with the smallest δ_T among the ones with the smallest Δ_T is returned.

Let k be the number of paths generated at line 2 of Figure 1, l be the number of paths generated at line 12 of Figure 1, $m = \mid M \mid$ be the size of the multicast group, and $n = \mid V \mid$ be the number of nodes in the network. The running time of **CCDVMA** is dominated by the loop from line 3 to line 21 in Figure 1. This loop is executed at most k times. During an iteration of this outer loop, the while loop between lines 7 and 18 is executed at most $m - 1$ times. Inside this loop, the computation time is determined by the number of nodes in V_T and the l-shortest path algorithm at line 12. The algorithm at line 12 takes $\mathcal{O}(ln^3)$ [17], and the innermost loop takes time $\mathcal{O}(ln^4)$. Therefore, the overall complexity of **CCDVMA** is $\mathcal{O}(klmn^4)$. Since only paths with a delay less than Δ are considered, we expect the values of parameters k and l to be small constants (unless the delay tolerance specified by the application is very loose). Results to be presented shortly confirm our intuition.

4 Numerical Results

In this section we compare four different algorithms that can be used to construct constrained multicast trees: (a) **CCDVMA**, shown in Figure 1, attempts to find a low cost tree whose paths satisfy the delay and delay variation constraints (1) and (2), respectively; (b) **DVMA**, developed in [14], constructs trees satisfying both constraints (1) and (2) without paying any attention to the overall tree cost; (c) **KPP** [10] is a heuristic for the delay-constrained Steiner tree problem; (d) **BSMA** [9] is also a delay-constrained Steiner tree heuristic. It should be noted that algorithms **KPP** and **BSMA** attempt to minimize the tree cost subject to the delay constraint (1), but they are not concerned with meeting the delay variation constraint (2).

We investigate the average case behavior of the four algorithms in terms of five performance measures: maximum path delay, Δ_T, maximum path delay variation, δ_T, multicast tree cost, failure rate (i.e., fraction of instances for which the algorithm failed to construct a feasible tree), and total system running time. We run our simulations on the multicast routing simulator **MCRSIM**©[2], developed at North Carolina State University by Hussein Salama [18]. The **MCRSIM**© package simulates actual ATM networks with fixed cell sizes and heterogeneous link capacities. **MCRSIM**© also supports multiple traffic types, as well as background traffic on each link. The simulator implements a modified version of the random graph generator described in [4]. The modifications made guarantee that the resulting network is connected and that the minimum degree of any node is 2.

In our simulations we varied the number of nodes from 40 to 100. The multicast group size was set to 5% of the total number of nodes in the network, and the background traffic load at each link was set to 15% of the link capacity. The average node degree was set to 4, and the delay and delay variation tolerances were fixed at $\Delta = 0.025$ seconds and $\delta = 0.01$ seconds, respectively. Each data point plotted in Figures 2 - 7 represents the average of 100 different simulation runs for the stated values of the various parameters. Similar results for other values of parameters Δ and δ, the average nodal degree, the multicast group size, and the background traffic load can be found in [19].

In Figure 2 we plot the average *maximum* delay Δ_T of the trees generated by each of the four algorithms. In computing the average we did *not* include simulations for which an algorithm failed to construct a tree satisfying the delay constraint (1). As a result, the average is below the specified tolerance $\Delta = 0.025$ seconds, as expected. From the figure we can see that the best trees in terms of maximum delay are those generated by **DVMA**. This can be explained by noting that **DVMA** will not reject a tree (or path) because of high cost, as it is not concerned with keeping the overall tree cost low. Of the other three algorithms, **KPP** appears to have the best performance and **BSMA** the worst, with **CCDVMA** in between these two. The fraction of time the algorithms fail to construct a tree that meets the delay constraint is shown in Figure 3. Overall, the results in Figures 2 and 3 indicate that all four algorithms perform well with respect to the end-to-end delay of the final tree.

[2]**MCRSIM**© can be downloaded from ftp://ftp.csc.ncsu.edu/pub/rtcomm/mcrsim.html

Figure 4 compares the performance of the various algorithms in terms of the average maximum delay variation δ_T. The numbers shown are the averages of all trees that met the delay tolerance, regardless of whether or not they also met the delay variation tolerance. Therefore, the average is sometimes larger than the specified tolerance $\delta = 0.01$ seconds. Also, in Figure 5 we plot the percentage of time that the algorithms fail to generate a tree with a maximum delay variation of at most 0.01 seconds. We see that **DVMA** has the best performance in both measures, for the same reasons as before. **CCDVMA** also performs well, as expected. On the other hand, when the network size is 80 or 100 nodes, the average maximum delay variation for **BSMA** and **KPP** is higher than 0.01, and their failure rate is extremely high. These results are not surprising given that algorithms **KPP** and **BSMA** were not designed to satisfy the delay variation constraint (2), and reflect the fact that they are not appropriate for applications requiring path delay variation bounds. **CCDVMA** performs significantly better than **KPP** and **BSMA** in all cases except for the delay variation measure when the network size is 100 nodes. But for networks with 100 nodes, **CCDVMA** has a failure rate about half that of **KPP** and **BSMA**, meaning that the relatively few bad cases account for the increase in maximum delay variation for **CCDVMA**.

Let us now turn our attention to Figure 6 that compares the algorithms in terms of the overall cost of the generated trees. As expected, the trees generated by **DVMA** have the highest cost, but the other three algorithms exhibit very similar behavior. The results show that **CCDVMA** is performing as well as or better than **KPP** and **BSMA** in reducing the overall tree cost, while at the same time meeting the delay variation constraint. This an encouraging result, and is an indication that our algorithm can strike a good balance between the application and network requirements.

Finally, Figure 7 plots the total *system* over 100 simulation runs for each of the four algorithms (the system time only includes the time actually spent computing the various trees). We should emphasize, however, that **BSMA** and **KPP** were implemented in C++ (along with the **MCRSIM**© package) while **DVMA** and **CCDVMA** were implemented in C and then integrated into the simulator. As a result, the CPU times shown are not comparable across the two pairs of algorithms due to the added overhead of C++. Studies have shown that an implementation in C++ adds about a 20-40% overhead to a C implementation on the average, but it is not clear what the overhead is in this specific case. Therefore, the CPU times in Figure 7 are only indicative of the behavior of the individual algorithms with increasing network size. We can see from Figure 7 that, as the number of nodes increases, the average execution times for **KPP** and **DVMA** grow faster than those of **BSMA** and **CCDVMA**, and that **CCDVMA** is performing well even if we account for the added overhead of C++ in **KPP** and **BSMA**. The low running times for **CCDVMA** can be explained by Table 1 where we show the maximum (over all simulation runs) and average

Parameter	40 Nodes	60 Nodes	80 Nodes	100 Nodes
k	*Max* 5.0	*Max* 4.0	*Max* 4.0	*Max* 2.0
	Avg 4.2	*Avg* 3.5	*Avg* 2.7	*Avg* 1.9
l	*Max* 5.0	*Max* 11.0	*Max* 14.0	*Max* 20.0
	Avg 3.2	*Avg* 10.3	*Avg* 9.8	*Avg* 13.6

Table 1: Maximum and average values of parameters k and l for 5% multicast group membership and 15% background traffic load

values of parameters k and l (recall that the worst-case complexity of **CCDVMA** is $\mathcal{O}(klmn^4)$). Although in our implementation we set the values of k and l to $3n$ and n, respectively (where n is the number of nodes in the network), because of the imposed delay bound, only a small number of paths (relative to the number of nodes) is considered by the algorithm. These results confirm our intuition that the actual values of k and l are small.

Overall, although **CCDVMA** did not yield the best performance in any given measure, it consistently performed well in all measures. Our results indicate that **CCDVMA** is a reasonably fast heuristic for solving the intractable DVCST problem: it scales well to larger network topologies, its system running time is encouraging, and it provides good multicast trees. None of the other three algorithms studied gave better overall results.

5 Concluding Remarks

We considered a variant of the Steiner tree problem in which a set of constraints are imposed on the delay and delay variation along the paths of the final tree. We presented a heuristic for this problem that compares favorably with existing multicast algorithms in terms of tree cost, scalability, and system running time. Our results indicate that our algorithm is adept at balancing the user constraints (regarding the end-to-end path delay and delay variation) with the underlying network demands for low cost trees. With the proliferation of real-time interactive applications, this ability will become increasingly more important in future high-speed networks.

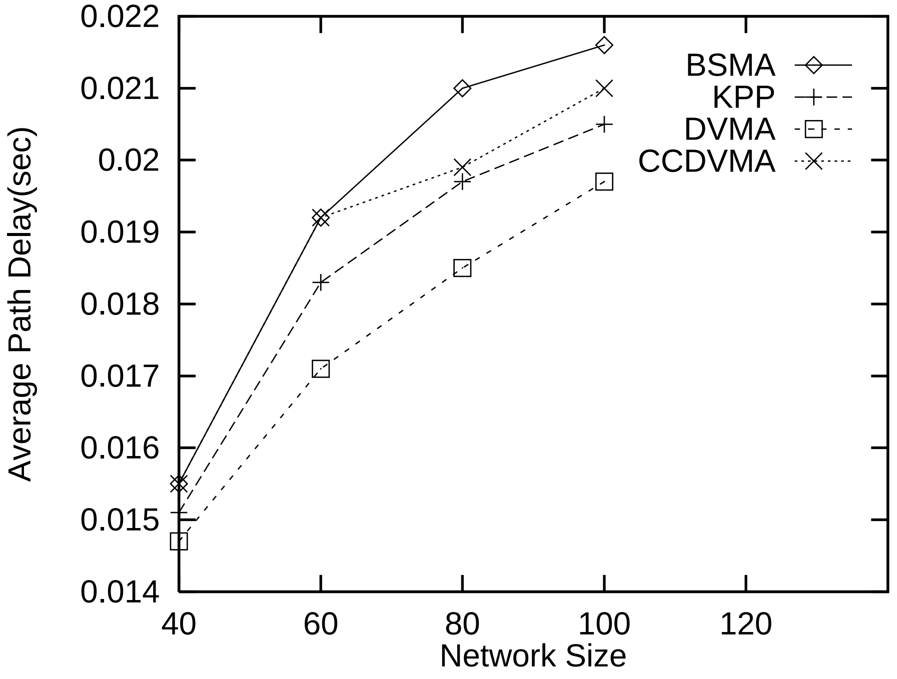

Figure 2: Average maximum path delay for 5% multicast group membership and 15% background traffic load

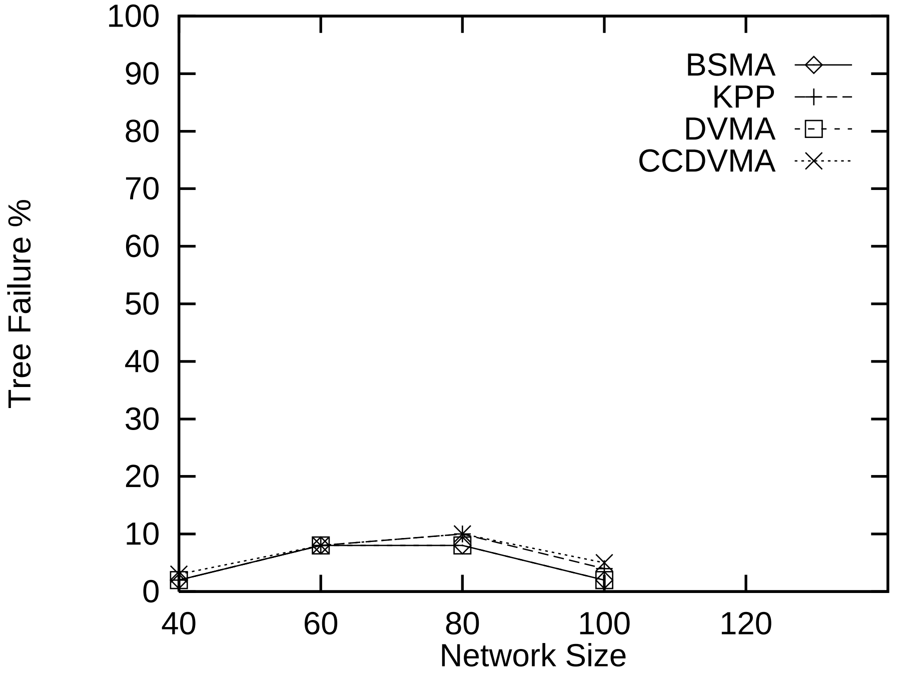

Figure 3: Percentage of tree generation failure due to violation of the delay tolerance for 5% multicast group membership and 15% background traffic load

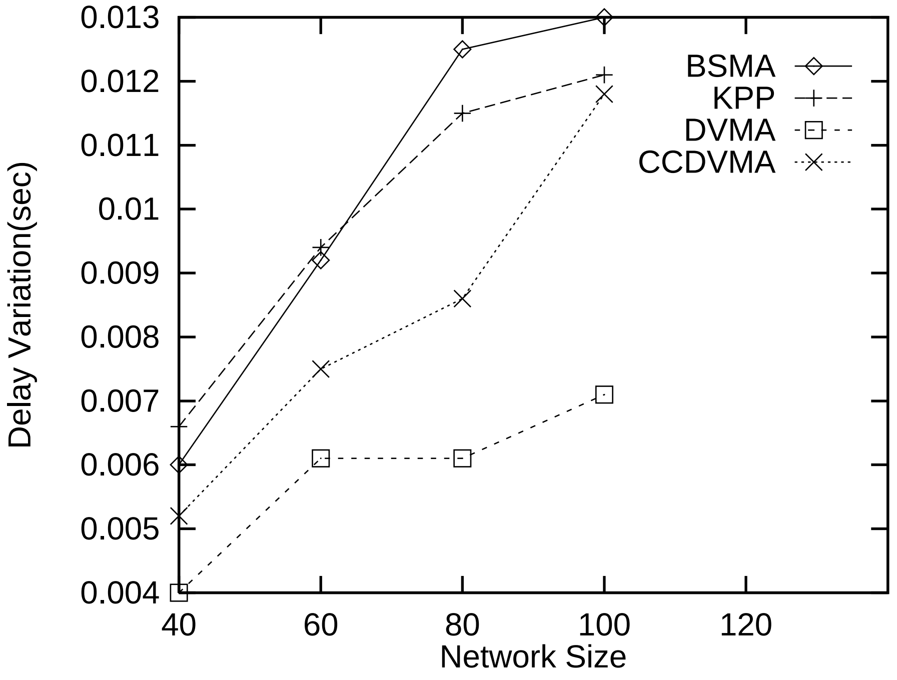

Figure 4: Delay variation comparison for 5% multicast group membership and 15% background traffic load

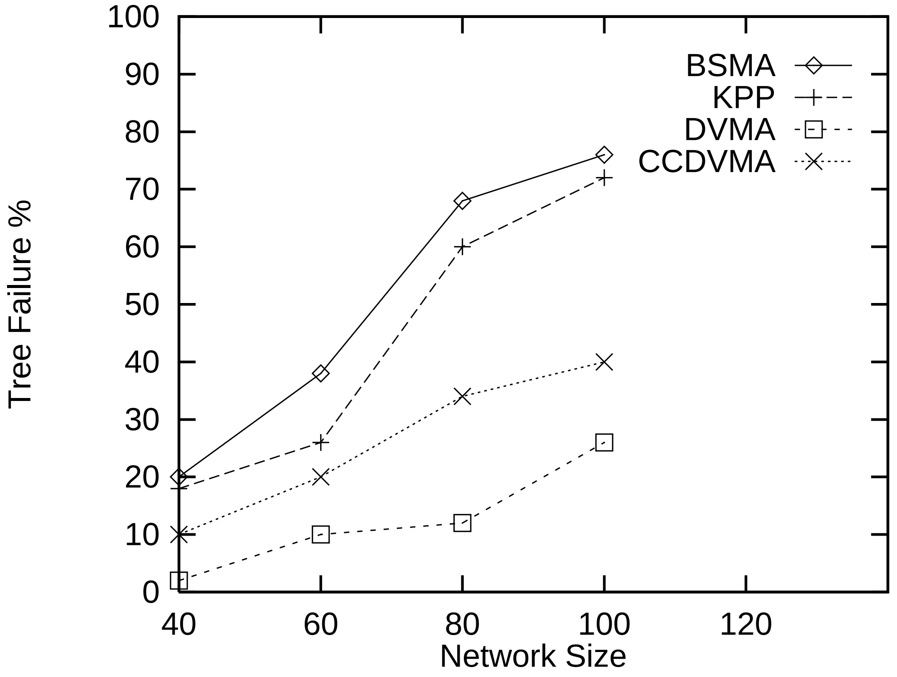

Figure 5: Percentage of tree generation failure due to violation of the delay variation tolerance for 5% multicast group membership and 15% background traffic load

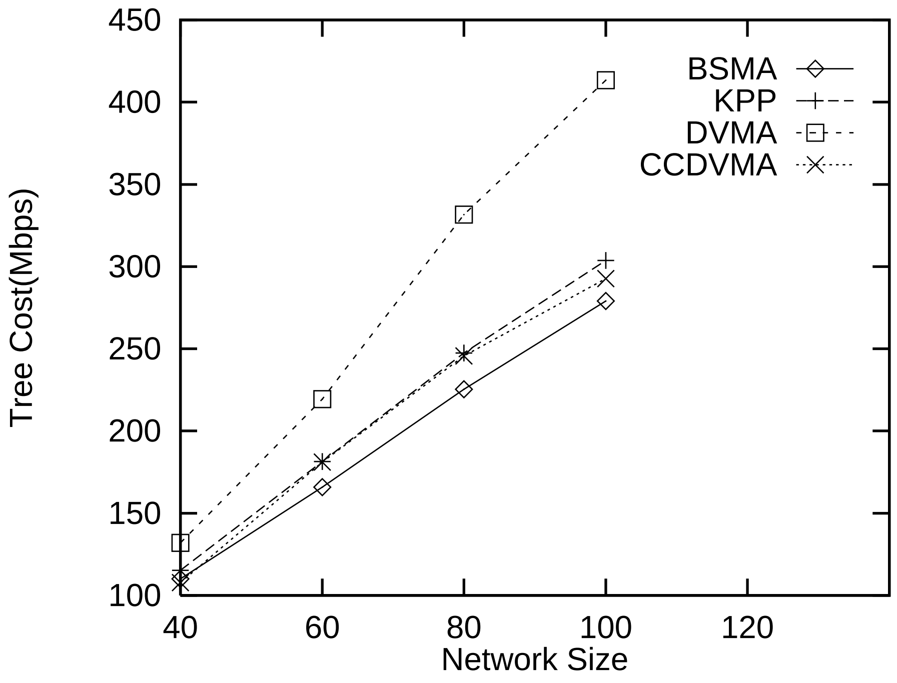

Figure 6: Cost comparison for 5% multicast group membership and 15% background traffic load

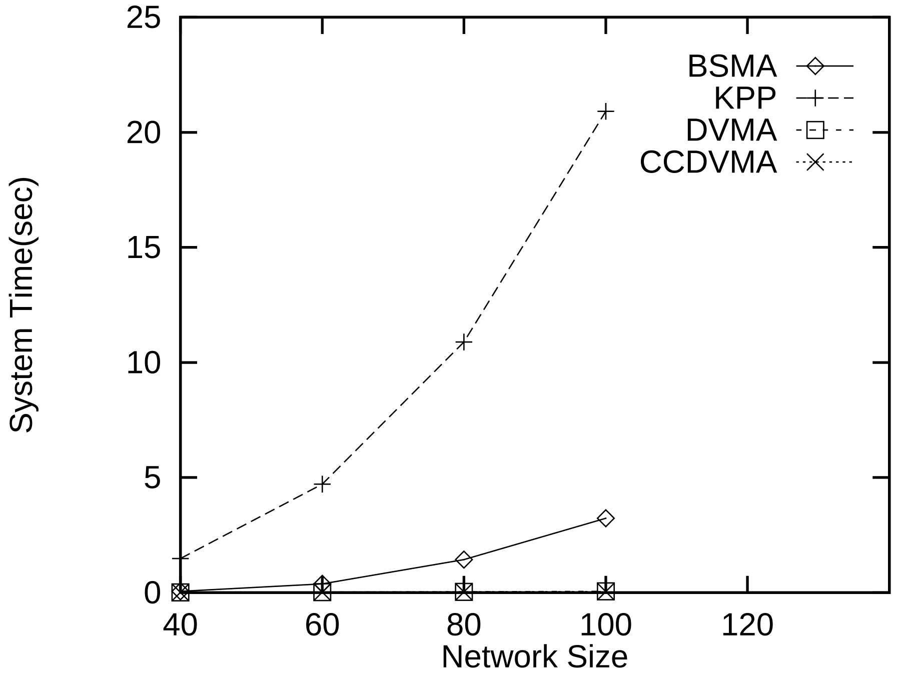

Figure 7: Average running time for 5% multicast group membership and 15% background traffic load

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] M. Ammar, G. Polyzos, and S. Tripathi (Eds.). Special issue on network support for multipoint communication. *IEEE Journal Selected Areas in Communications*, 15(3), April 1997.

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

[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.

[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] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking*, 4(4):558–568, August 1996.

[9] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for near-optimum delay-constrained multicasting. In *Proceedings of IEEE Infocom '95*, March 1995.

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

[11] R. Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, University of California at Berkeley, Department of EECS, June 1994.

[12] M. R. Macedonia and D. P. Brutzman. Mbone provides audio and video across the internet. *IEEE Computer*, 27:30–36, April 1994.

[13] B. Roehle. Channeling the data flood. *IEEE Spectrum*, 34(3):32–38, March 1997.

[14] G. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communications*, 15(3):346–356, April 1997.

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

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

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

[18] H. Salama. *Multicast Routing for Real-time Communication on High-Speed Networks.* PhD thesis, North Carolina State University, Department of Electrical and Computer Engineering, December 1996.

[19] B. K. Haberman. Cost, delay, and delay variation conscious multicast routing. Master's thesis, North Carolina State University, Raleigh, NC, March 1997.