A SOURCE-BASED ALGORITHM FOR
DELAY-CONSTRAINED MINIMUM-COST MULTICASTING

Qing Zhu, Mehrdad Parsa, and J.J. Garcia-Luna-Aceves
Department of Computer Engineering
University of California
Santa Cruz, CA 95064
qingz, courant, jj@ece.ucsc.edu

ABSTRACT

A new heuristic algorithm is presented for constructing minimum-cost multicast trees with delay constraints. The new algorithm can set variable delay bounds on destinations and handles two variants of the network cost optimization goal: one minimizing the total cost (total bandwidth utilization) of the tree, and another minimizing the maximal link cost (the most congested link). Instead of the single-pass tree construction approach used in most previous heuristics, the new algorithm is based on a feasible search optimization method which starts with the minimum-delay tree and monotonically decreases the cost by iterative improvement of the delay-bounded tree. The optimality of the costs of the delay-bounded trees obtained with the new algorithm is analyzed by simulation. Depending on how tight the delay bounds are, the costs of the multicast trees obtained with the new algorithm are shown to be very close to the costs of the trees obtained by the Kou, Markowsky and Berman’s algorithm.

1 Introduction

Multicasting consists of concurrently sending the same information to a group of destinations. This is becoming a key requirement of computer networks supporting multimedia applications. To support large numbers of multicast sessions efficiently, the network must transport the information exchanged in these sessions using as few network resources as possible, while meeting the sessions’ service requirements. The current approach to supporting a multicast session efficiently in the network consists of establishing a multicast tree for the session, along which session information is transferred. Algorithms are needed in the network to compute such trees; we call such algorithms multicast routing algorithms.

Different optimization goals can be used in multicast routing algorithms to determine what constitutes a good tree. One such goal is providing minimum delay along the tree, which is important for such multimedia applications as real-time conferencing. Another optimization goal is making use of the network resources as efficiently as possible; two interesting variants of this objective are

- Minimizing the total bandwidth utilization of links; we call this objective utilization driven, because it minimizes the total bandwidth utilization cost of links for a data stream sent from the source to destinations.
- Distributing bandwidth utilization of sessions among links in the tree in order to minimize congestion along links; we call this objective congestion driven, because it minimizes the maximal link cost requirement along the transferring paths.

Previous optimization techniques for multicast routing algorithms have considered the above two optimization objectives, but have treated them as distinct problems. Dijkstra’s shortest path algorithm [6] can be used to generate the shortest paths from the source to destination nodes; this provides the optimal solution for delay optimization. Multicast routing algorithms that perform cost optimization have been based on computing the minimum Steiner tree which is known to be NP-complete problem [8]. Some heuristics for the Steiner tree problem [10, 13, 15, 7] have been developed that take polynomial time and produce near optimum results. In Kou, Markowsky’s and Berman’s (KMB) algorithm [10], a network is abstracted to a complete graph consisting of edges that represent the shortest paths among source node and

---

*This work was supported in part by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175*
destination nodes. The KMB algorithm applies Prim’s algorithm [12] to construct a minimum spanning tree in the complete graph, and the Steiner tree of the original network is obtained by achieving the shortest paths represented by edges in the minimum spanning tree. Waxman [16] examined the dynamic update of the tree if destination nodes join or leave the tree occasionally. Shacham and Meditch [14] investigated the maximum flow distribution of multiple streams along a multicast tree; optimum assignment is based on link capacity and destination requests, but the request is not delay-bounded and a dynamic programming algorithm is devised that may take exponential time complexity.

Bharath-Kumar and Jaffe discuss optimization on both cost and delay [3]. However, they assume that cost and delay functions are identical. Kom-pella, Pasquale and Polyzos [9] propose two heuristics (which we call the KPP algorithm) that address delay-bounded Steiner trees; the KPP algorithm extends the KMB algorithm by taking into account an integer-valued delay bound in the construction of shortest paths.

This paper presents a new algorithm for multicast tree construction. The algorithm is source based, which means the source of the multicast is assumed to know all the information needed to construct the multicast tree. Although the model of delay-bounded minimum Steiner tree is similar to the one used in the KPP algorithm, we generalize the formulation of the problem to more realistic network settings; the major contributions of the new algorithm, which we call Bounded Shortest Multicast Algorithm (BSMA) \(^1\) can be summarized as follows:

- It considers the congestion-driven and utilization-driven variants of the cost function.
- It can specify different delay bounds for different destinations.
- The delay cost function and delay bound can take arbitrary positive real values.
- Instead of using a single pass as in previous algorithms, it iteratively (in multiple passes) minimizes the cost function of the tree.

The remaining of this paper describes the network model assumed in BSMA, describes BSMA in detail, demonstrates the correctness of BSMA, and analyzes BSMA’s performance by simulation.

\(^1\)The fact that BSMA also spells “Banana Slug Multicast Algorithm” and that the authors’ affiliation is UCSC is entirely coincidental, of course.

2 Network Model for Multicasting

A network is modeled as a weighted graph \(G = (V, E)\), as shown in Fig. 1, with node set \(V\) and edge set \(E\). The edges in \(E\) correspond to the communication links connecting the network nodes. The nodes in set \(V\) can be of the following three types:

- **Source node**: the node connecting to the source that sends out the data stream.
- **Destination node**: A node connecting a destination that receives the data stream. The set of destination nodes in a multicast tree is denoted by \(D \subseteq V\) – \(s\).
- **Relay node**: A node that is an intermediate hop in the path from the source to a destination.

Two positive real-valued functions are defined on \(E\), namely:

- **Link-Cost Function** \((c : E \rightarrow \mathbb{R}^+):\) The cost of a link can be associated with the utilization of the link; a higher utilization is represented by a higher link cost.
- **Link-Delay Function** \((d : E \rightarrow \mathbb{R}^+):\) The delay of a link is the sum of the perceived queueing delay, transmission delay, and propagation delay over that link.

![Figure 1: Network model](image)

Let \(D\) be the set of destinations. For each path from \(s\) to a destination node \(v \in D\), the delay of the path, or path delay, is defined to be the sum of link delays along the path. The following function is defined on the set of destinations \(D\):

**Destination Delay-Bound Function** or DDF \((\delta : D \rightarrow \mathbb{R}^+):\) DDF assigns an upper bound to the delay along the path from the source to each destination in \(D\). Of course, \(\delta(i)\) can be different from \(\delta(j)\)
for destinations \( i \neq j \). For the case in which DDF assigns the same upper-bound delay to each destination, that upper bound is denoted by \( \delta(i) = \Delta \), \( \forall i \in D \).

The multicast tree that we are interested in constructing is a delay-bounded minimum Steiner tree (DMST) and the minimization problem can be formally described as follows:

**DMST Problem**: Given a graph \( G = (V, E) \) with a link-cost function, a link-delay function, a source \( s \), a set of destinations \( D \), and a DDF, then construct a DMST spanning \( D \cup \{s\} \), such that the cost function of the tree is minimized while DDF is satisfied.

The two cost functions addressed in this paper are:
- the sum of link costs in the multicast tree, and
- the maximal link cost in the multicast tree.

The DMST problem based on minimizing the sum of link costs in \( T \) can be reduced to a minimum Steiner tree problem with the delay bound set to \( \infty \), which is known to be an NP-complete problem [8] and only heuristics are of practical interest for its solution; BSMA is a new heuristic designed to solve the DMST problem.

3 Description of BSMA

3.1 Overview

BSMA assumes that the source node has complete information regarding all network links to construct a multicast tree. This requirement can be supported using one of many topology-broadcast algorithms, which can be based on flooding (as is the case in OSPF and IS-IS) or other techniques [2].

BSMA is based on the feasible search optimization method. This method minimizes the objective function constrained inside a feasible region. A good survey on the feasible search optimization method is given in [4]. The feasible region \( R_b \) for the DMST problem consists of the set of all delay-bounded Steiner trees.

BSMA starts with an initial tree \( T_0 \) \( \in R_b \), and iteratively refines the tree for low cost while staying in the feasible region.

BSMA consists of two major steps; a skeleton flowchart is shown in Fig. 2. Step 1 constructs the initial tree \( T_0 \) which is a minimum-delay Steiner tree, with respect to the multicast source, using Dijkstra's shortest path algorithm [6]. Step 2 then iteratively refines \( T_0 \) for low cost. Fig. 3 shows an example network and the corresponding minimum-delay Steiner tree for a multicast group in the network.

In some cases, the delay bounds given by DDF may be too tight, i.e., they cannot be met even in the minimum-delay tree \( T_0 \). In such cases, some negotiation is required to loose the delay bounds of DDF before any feasible tree can be constructed, as shown in Fig. 2. To guarantee that the output tree is delay-bounded, Step 1 must be invoked with reassigned delay bounds, after the negotiation has been accomplished. The rest of this paper assumes that DDF gives delay bounds that have been met by the minimum-delay tree \( T_0 \).

![Flowchart of Basic Approach Used in BSMA](image)

**Figure 2**: Flowchart of Basic Approach Used in BSMA.

Step 2 iteratively refines \( T_0 \) in a monotonous cost reduction. A tree obtained during each refinement is called a *tree configuration*, and the \( j \)th tree configuration is denoted by \( T_j \).

The refinement from \( T_j \) to \( T_{j+1} \) (initially, \( j = 0 \)) is accomplished by an operation called *path switching*.

3.2 Delay-Bounded Path Switching

Path switching means that a path in \( T_j \) is replaced by a new path that is not in \( T_j \), resulting in a new tree configuration \( T_{j+1} \) [17]. Delay-bounded path switching guarantees that \( T_{j+1} \) is a delay bounded tree. An example is shown in Fig. 4, where the goal is to obtain
a DMST satisfying delay bound $\Delta = 5$; accordingly, a path $AB$ of the tree is switched to a new path $BC$. The cost of the tree is reduced from 32 to 23, while the maximum path delay becomes 4 (Fig. 4(b)) which is still bounded by $\Delta = 5$.

![Diagram](image)

Figure 4: Delay bounded path switching: a path $AB$ in the tree is replaced by a new path $BC$.

Ensuring an effective delay-bounded path switching during the $j$th refinement involves:

- Choosing the path to be taken out of $T_j$.
- Selecting the new path in $G$ not in $T_j$ that replaces the path to be deleted from $T_j$.

To represent the candidate paths chosen in the path switching, we define a new tree denoted by $T'_j$ which is called a **collapsed tree** of $T_j$. The collapsed tree $T'_j$ consists of nodes and **superedges**.

The set of nodes of $T'_j$ consists of the source node and destination nodes of $T_j$, and those nodes of $T_j$ that are connected by more than two tree edges in $T_j$.

A superedge in $T'_j$ is the longest simple path in $T_j$ in which all internal nodes (i.e., excluding the end nodes of the path) are relay nodes and each relay node connects exactly two tree edges. As Fig. 4 shows, path DEF corresponds to a superedge, because the relay node $E$ is connected by exactly two tree edges, namely, DE and EF. Every superedge represents a path in $T_j$ for possible path switching. To reduce the cost of $T_j$, BSMA deletes a superedge from $T'_j$, (which amounts to removing a path $p$ from $T_j$) resulting in two subtrees $T'_j^1$ and $T'_j^2$, where $T_j = T'_j^1 \cup T'_j^2 \cup p$. BSMA then finds a delay-bounded shortest path $p_k$ to reconnect $T'_j^1$ and $T'_j^2$.

Path cost is defined as follows:

- For utilization-driven multicasting, the path cost is the **sum** of link costs along this path.
- For congestion-driven multicasting, the path cost is the **maximal** link cost along the path.

The cost of a superedge in $T'_j$ is defined to be the cost of the corresponding path in $T_j$.

A delay-bounded shortest path $p_k$ between $T_j^1$ and $T_j^2$ is defined as the path with the smallest cost, subject to the constraint that the new tree $T_{j+1} = T_j^1 \cup T_j^2 \cup p_k$ is a delay-bounded tree. As an example in Fig. 5(a), two subtrees of $T_j$ results from removing the superedge (path) AB; and the delay bounded shortest path in Fig. 5(b) is BC. Adding BC makes the delay bounds to be satisfied in the new tree shown in Fig. 5(b) (the maximal path delay is $4 < \Delta = 5$). The cost reduction from switching path AB to path BC is $10 - 2 = 8$.

We can easily add the tie-breaking choice rule for multiple delay-bounded shortest paths: the path chosen is the one providing the maximum slack of destination delays minus delay bounds, so that there is more delay budget for iterative refinement.

![Diagram](image)

Figure 5: (a) Removing AB from the tree results in two unconnected subtrees $T_j^1$ and $T_j^2$. (b) The shortest delay-bounded path BC connecting $T_j^1$ and $T_j^2$.

To get the delay-bounded shortest path, BSMA incrementally constructs the $k$-shortest paths between two subtrees $T_j^1$ and $T_j^2$. Using a $k$-shortest path algorithm [11], the construction of shortest paths starts from the first shortest path, then the second shortest path, the third shortest path and so on; $k$ is not known a priori, and is determined only after a shortest path is found which has resulted in a delay-bounded tree. Specifically, the $k$-shortest path incremental construction stops when one of the following two conditions is satisfied:

- The shortest path found does not result in the new tree violating the delay bound.
The shortest path found has equal path length to the one just deleted.

The incremental construction of the $k$-shortest path between $T^1_j$ and $T^2_j$ consecutively calls Dijkstra's shortest-path algorithm, eliminating already found shortest paths [11] from consideration. The key step is extending Dijkstra's algorithm to construct the shortest path between two subtrees instead of two nodes. A pseudo source node $s$ is connected to all nodes in $T^1_j$, and a pseudo destination node $t$ connected to all nodes in $T^2_j$. The shortest-path algorithm starts from the pseudo source node $s$ and executes the neighbor node search using Dijkstra’s algorithm, is modified in two ways: (a) The initial queue of nodes contains all nodes in $T^1_j$; (b) the path is backtraced when a node in $T^2_j$ is removed from the queue, since the nodes in $T^2_j$ have been connected to the pseudo destination node $t$.

### 3.3 Path-Switching Heuristic

Initially, all superedges are unmarked. Among all unmarked superedges, BSMA selects the superedge $p_h$ with the highest path cost, and exchanges it with another superedge such that the resulting paths to destinations are delay-bounded. One of two cases must happen:

(a) The delay-bounded shortest path is the same as $p_h$.

(b) The delay-bounded shortest path is a path other than $p_h$.

If case (a) occurs, BSMA marks $p_h$, which means $p_h$ has been examined, and continues to select another superedge with the highest path cost among the remaining unmarked superedges available for path switching. The deleted path $p_h$ is put back in the tree, i.e., $T_{j+1} = T_j$. Note that the delay-bounded shortest path algorithm always terminates, because at least the deleted path is found again.

If case (b) occurs, BSMA unmarks all the marked superedges, and continues to select a superedge with the highest cost for path switching.

BSMA terminates when all superedges have been marked in the tree. That is, all possible superedges in the tree have been examined for path-switching without success in reducing the tree cost. Fig. 6 shows a high-level specification of BSMA, where routine `DelayBoundedShortestPath()` corresponds to the use of a $k$-shortest path algorithm. Figure 7 shows the steps of constructing a multicast tree based on utilization-driven optimization. The total edge cost of $T$ is dropped from 32 (the shortest delay tree) to 21, and the maximal path delay of the final tree is $4 < \Delta = 5$. As a comparison, the minimum-cost tree constructed by the KMB algorithm [10] is shown in Fig. 8(a), which has a cost of 20 and the maximal path delay of 8 which is larger than the delay bound $\Delta = 5$. Figure 8(b) shows the multicast tree obtained in BSMA using congestion-driven optimization, where the most congested link is switched each time. The tree is the same as the one obtained with utilization minimization, as shown in Fig. 7(d).

### 3.4 Greedy Path-Switching

Another heuristic that can be used in BSMA is a greedy selection of path switching at each tree refinement. We define gain $g$ as the cost reduction after a round of path-switching. Let $p$ be a path in tree $T_j$ with cost $c$, and $p^*$ the corresponding delay bounded shortest path to be added into tree $T_{k+1}$ with cost $c^*$. The gain $g$ of this path switching is defined as: $g = c - c^*$.

BSMA computes gains of all pairs of possible path switchings in $T_j$, and then selects the one with the maximum gain. BSMA continues the greedy path switching, and terminates when the maximum gain is zero.

In this greedy heuristic, each decision of path
switching can be made after all superedges are tentatively switched to their delay-bounded shortest paths. On the other hand, the heuristic described in Section 3.3 decides to make a path switching whenever it succeeds in reducing the cost of the tree.

We compared the greedy heuristic with the heuristic of Section 3.3, and found a small difference of costs for final trees. The cost difference for all tested examples was less than 2%. Although the greedy heuristic may make the algorithm description clearer, it increases the time complexity. The time complexity of the greedy path selection per path-switching is higher by a factor $O(|U|)$, where $U$ is the set of superedges and $|U| = O(n)$.

4 Correctness of BSMA

Theorem 1: BSMA always constructs a delay-bounded multicast tree, if such a tree exists.
Proof: The proof is done by induction on $j$, the tree configuration number. The basis for induction is $T_0$, which is a shortest-delay tree. If $T_0$ cannot satisfy the delay bounds of DDF, there can be no tree satisfying the given delay bounds. Assume that the $T_j$ is delay bounded. $T_{j+1}$ is obtained from $T_j$ by path-switching, which by definition exchanges highest cost superedge with one that does not result in paths to destinations exceeding their delay bounds. □

Furthermore, BSMA is shown to be sub-optimal in the cost reduction.

Theorem 2: BSMA monotonically decreases the cost of the delay-bounded tree.
Proof: Consider the refinement from $T_j$ to $T_{j+1}$. Let $c$ be the cost of the superedge to be removed, and $c^*$ be the cost of the delay-bounded shortest path to be added. Obviously, $c^* \leq c$, since $c^*$ corresponds to the cost of the shortest one among all delay bounded paths, that in the worst case will be the cost $c$ of the deleted path. □

5 Performance of BSMA

5.1 Time Complexity

Consider a network of $n$ nodes and denote by $k$ the average number of the $k$-shortest paths constructed to obtain the delay-bounded shortest path.

Theorem 3: The expected time complexity of BSMA is $O(kn^3 \log(n))$.
Proof: To analyze the time complexity of BSMA, examine the behavior of an idealized algorithm, call it IA, which can find the optimal solution of the minimum Steiner tree. IA constructs an initial Steiner tree and iteratively transforms the Steiner tree to get to the global minimum. IA may stay at the same tree for a maximum of $n$ iterations.
By Cayley’s Theorem [5], there are \( n^{n-2} \) possible spanning trees on \( n \) nodes; thus the number of Steiner trees spanning \( n_{j} \) (\( n_{j} \leq n \)) is then bounded by \( n^{n-2} \). The number of Steiner trees spanning \( n \) nodes is then bounded by \( n^{n-2} \) since \( n_{j} \leq n \). Construct a Markov chain of \( n^{n-2} \) states, where each state corresponds to a spanning tree. Sort these states in a nonincreasing order from left to right with respect to the cost of the Steiner tree (not the spanning tree) breaking ties arbitrarily. Replace each state in the sorted list with \( n \) copies of itself to get a total of \( n^{n-1} \) states. Each of these \( n \) replicated states corresponds to the same tree. In this Markov chain, transition edges from a state \( S_{i} \) go only to a state to the right of \( S_{i} \). Assume that each of the possible transitions is equally likely. Thus, the probability of transitioning from \( S_{i} \) to \( S_{j} \) is

\[
P_{ij} = \frac{1}{i-1} \text{ for } 1 \leq j < i
\]

and \( P_{11} = 1 \). Let \( T_{i} \) be the number of transitions needed to go from state \( i \) to state 1. The expected value can be found by conditioning on the first transition. Let \( Y \) be the random variable of the next state of the first transition.

\[
E[T_{i}] = E[E[T_{i}|Y]] = \sum_{y} E[T_{i}|Y = y] P\{Y = y\} = \sum_{y} E[T_{i}|Y = y] \frac{1}{i-1}
\]

\[
= \frac{1}{i-1} \sum_{y=1}^{i-1} (1 + E[T_{y}]) = 1 + \frac{1}{i-1} \sum_{y=1}^{i-1} E[T_{y}]
\]

Using induction with \( E[T_{1}] = 0 \), it can be shown that \( E[T_{i}] = \sum_{y=1}^{i-1} 1/y \approx \log(i) \).

Therefore, if IA starts in the most expensive state, i.e., \( i = n^{n-1} \), then the expected number of transitions, i.e., iterations, is \( O(\log(n^{n-1})) = O(n \log(n)) \). This is for an ideal algorithm that finds the global minimum. BSMA is likely to terminate earlier in a local minimum, i.e., when path-switching fails consecutively for every superedge. An unsuccessful path-switching corresponds to IA making a transition from a state \( S_{i} \) to a replicated state of \( S_{i} \) to the right of \( S_{i} \). Thus, the maximum expected number of path-switchings for BSMA is \( O(n \log(n)) \). A path-switching is dominated by computing the \( k \)-shortest path. This can be done in \( O(ks(n)) \), where \( s(n) \) is the complexity of doing a (single-source) shortest-path algorithm. A simple implementation of Dijkstra’s shortest-path algorithm takes \( O(n^2) \). The expected time complexity of BSMA is thus \( O(ks(n) \cdot n \log(n)) = O((kn^2 \cdot n \log(n)) = O(kn^3 \log(n)) \). □

In using Cayley’s Theorem, we have allowed the underlying graph on \( n \) nodes to be complete. In practice, we are interested in degree-bounded networks where the maximum degree is upper-bounded.

**Corollary 1:** The expected time complexity of BSMA is \( O(kn^3) \) for \( n \) nodes in a degree-bounded network.

Proof: The edge number is bounded by \( dn/2 \) for a degree-bounded graph with the maximum degree \( d \) and \( n \) nodes. Therefore, the maximum number of possible graphs is \( 2^{dn/2} \), which gives an upper bound on the number of spanning trees. Following the same line of proof in Theorem 3, we get the expected number of path-switchings to be \( O(\log(n^{2dn/2})) = O(n) \). The expected time complexity of BSMA is thus \( O(kn^2 \cdot n) = O(kn^3) \). □

The value of \( k \) for the delay bounded shortest path construction depends on the delay bound. For a not-too-tight delay bound, by empirical simulation, \( k \) was found to be a small number on average.

### 5.2 Simulation Results

BSMA was implemented in C++. The following results are obtained based on BSMA’s heuristic of Section 3.3. The greedy heuristic described in Section 3.4 gets the costs of trees within 2% difference but much longer running time. The performance is evaluated using random graphs as the network model. We attained qualitatively similar results for different random graphs with \( |V| \) nodes. For a given number of nodes, the results presented here are for a fixed random graph as the network. The graphs used were designed to be sparse with the average degree being less than five to capture the flavor of realistic network topologies. Group members are picked uniformly from the set of nodes in the graph (excluding the nodes already selected for the group).

The random graphs used in the simulation are constructed using the method proposed in [16]. The \( n \) nodes of a graph are randomly placed on a Cartesian coordinate grid with unit spacing. The \((x, y)\) coordinates of each node was selected uniformly from integers in \([0, n]\). Considering all possible pairs of node, edges are placed connecting nodes with the probability

\[
P(u, v) = \beta \exp \left( \frac{-d(u, v)}{\alpha L} \right)
\]

where \( d(u, v) \) is the Manhattan distance between
nodes \( n \) and \( v \), and \( L \) is the maximum possible distance between two nodes. The parameters \( \alpha \) and \( \beta \) are in the range \([0,1]\) and can be selected to obtain desired characteristics in the graph. For example, a large \( \beta \) gives nodes with a high average degree, and a small \( \alpha \) gives long connections. It has been observed that, with appropriate parameters, this method gives networks that resemble “real world” networks. The parameters \( \alpha \) and \( \beta \) are varied to obtain appropriately sparse networks, i.e., average degree of node is less than or equal to five.

The cost of each edge was set to the Manhattan distance between its endpoints plus one. By adding one to the Manhattan distance, the interesting case of zero edge cost is eliminated. The delay \( D \) of an edge is set to a uniform random number in \([0,1]\) time its cost plus one. This definition of delay is used to eliminate the unrealistic possibility of zero delay. The graphs obtained for the simulation runs have the average degree listed in Table 1.

BSMA was run on 25-, 50-, 75-, and 100-node graphs. The number of destinations in a group is 4 in one set of runs and 6 in another set. For the sake of simplicity and comparison with other heuristics, the delay bound is the same for all destinations. The cost of the resulting multicast tree is averaged over the number of groups. The total number of groups is 2500 and 3500 for the set of runs with 4 destinations and 6 destinations, respectively.

The cost of the solution when the delay bound is the same as that of the KMB solution is very nearly the same as the unconstrained KMB solution. The KMB solution has been shown to yield a worst-case cost performance of \( 2(1 - \frac{1}{|S|}) \), where \( Z \) the set of leaves in the optimal Steiner tree [10]. Therefore, we can conclude that the quality of the solution found by BSMA is minimal. Using the tightest possible delay bound, as determined by the minimum delay tree, the cost of the BSMA tree is substantially better than the cost of the minimum delay tree. By controlling the delay bound between the two extremes of the KMB and minimum delay solution, a range of minimum cost solutions can be obtained.

Another set of runs was done with all the nodes in the network belonging in one group and each node taking turn as the source and the rest of the network

| \( |V| \) | average degree |
|-------|--------------|
| 25    | 3.76         |
| 75    | 6.06         |
| 100   | 6.00         |

Table 1: Average degree for the networks in the simulations.
being the set of destinations. In this case, the Steiner tree problem reduces to the minimum weight spanning tree for the unconstrained case. The results are shown in Figure 10. Again, the cost of the BSMA solution with the delay bound set to that of the minimum-weight spanning tree is nearly identical to that of the minimum-weight spanning tree solution, which is the optimal solution. By tightening the delay bound, a range of solutions can be obtained between the optimal cost solution and the minimum delay solution. Note that, compared to KMB, the relative quality of the results obtained with BSMA improve with the number of destinations in the multicast group (this is apparent by comparing Fig. 9 and Fig. 10). Interestingly, it is not clear that the maximum k necessarily increases with the multicast membership for a given network size.

The delay bound given has a strong influence on the the maximum k-shortest-path computed by BSMA and thus on the execution time of the algorithm. A tighter delay bound results in larger k. Figure 11 shows the maximum value of k for different delay bounds. The results for the maximum k in the case when the whole network is one group were qualitatively the same and are omitted here.

It can be seen that a slight relaxation of the delay bound often results in considerably fewer computations. This is especially true for larger networks. A possible heuristic for setting the delay bound is the half way between the two extremes obtained by the minimum-cost Steiner trees (KMB) and the minimum-delay trees. Perhaps a good heuristic for the upper bound for the minimum delay is the delay of the shortest cost tree to each destination found quickly by Dijkstra’s algorithm. The lower bound to minimum delay can be quickly found also using Dijkstra’s algorithm with delay cost. Then a possible strategy for setting the delay bound is half way between these two extremes.

The running time of BSMA and KMB on tested examples is shown in Table 2. The results are very encouraging, considering that the times are for execution on Sun Sparcstation 1+ and that the code is not optimized for speed.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Average execution time per group, given in seconds, for BSMA and KMB on Sun Sparc1+ workstations.

6 Conclusions

Multicast tree construction is becoming an integral part of multimedia application support. This paper proposed a source-based algorithm for the construction of delay-bounded minimum-cost multicast trees. The contribution of our work lies both in the formulation of the problem and the novelty of the algorithm used to solve it. We allow variable delay bounds set for different destinations or different media. This simulates timing requirements of realistic networks supporting multimedia applications. The algorithm can handle two variants of the cost function: one minimizing the total link cost of the tree, and the other minimizing the most congested (maximal) link cost. Instead of using the one-pass growing of the multicast tree used in most previous works, we propose an iterative optimization process to further minimize the tree cost. The cost minimization is monotonically achieved after a series of tree refinement. The simulation results show that BSMA produces delay-bounded multicast trees that, depending on the delay bounds imposed, can be very close to having minimum cost.

Work continues to extend BSMA in a few directions. It can be extended to handle a dynamic delay-bounded minimum Steiner tree; the key to accomplish this is to update the existing multicast tree when destinations need to be added or deleted, without regenerating the tree from scratch. Another extension consists of allowing each source node to know only partial topology information, using an approach similar to the one proposed in [1]. Finally, the heuristic described in this paper needs to be incorporated into an appropriate multicast routing protocol.
Figure 11: The maximum k-shortest path computed. As the delay bound is tightened larger values of k are computed.

References