An Efficient Delay-Constrained Minimum Spanning Tree Heuristic [*]

H.F. Salama D.S. Reeves Y. Viniotis
Center for Advanced Computing and Communication
ECE Department and CSC Department
Box 7911, North Carolina State University, Raleigh, NC 27695-7911, USA
Phone: (919) 515-5348 Fax: (919) 515-2285.
Email: {`hfsalama,reeves,candice`}`@eos.ncsu.edu`

Abstract

We formulate the problem of constructing broadcast trees for real-time traffic with delay constraints in networks with asymmetric link loads as a delay-constrained minimum spanning tree (DCMST) problem in directed networks. Then we prove that this problem is *NP*-complete, and we propose an efficient heuristic to solve the problem based on Prim's algorithm for the unconstrained minimum spanning tree problem. This is the first heuristic designed specifically for solving the DCMST problem. Simulation results under realistic networking conditions show that our heuristic's performance is close to optimal when the link loads are symmetric as well as when asymmetric link loads are used. Delay-constrained minimum Steiner tree heuristics can be used to solve the DCMST problem. Simulation results indicate that the fastest delay-constrained minimum Steiner tree heuristic, DMCT [1], is not as efficient as the heuristic we propose, while the most efficient delay-constrained minimum Steiner tree heuristic, BSMA [2], is much slower than our proposed heuristic and does not construct cheaper delay-constrained broadcast trees.

1 Introduction

Most distributed real-time applications involve more than two users and hence the need for efficient multicast routing. Several multicast routing protocols have been proposed. These protocols construct multicast trees that can be classified into two categories: source-specific trees (MOSPF [3], DVMRP [4], and PIM [5, 6]) and shared trees (CBT [7] and PIM[1] [5])[2]. All these protocols are

[*]This work was supported in part by the Center for Advanced Computing and Communication at North Carolina State University and in part by an AFOSR grant F49620-92-J-0441DEF.

[1]PIM has two modes: a dense mode that uses source-specific trees, and a sparse mode that allows both source-specific trees and shared trees.

[2]In this paper we consider only source-specific trees.

based on simple multicast routing algorithms: shortest path algorithms [8, 9] and reverse path multicasting [10, 11], and none of them applies cost metrics that are functions of the utilization of the network resources.

Real-time applications, e.g., multimedia and distributed real-time control applications, will be popular applications of high-speed networks in the near future. Real-time traffic is usually bandwidth extensive and requires quality of service (QoS) guarantees from the underlying network. Hence the need for efficient multicast routing algorithms which define cost as a function of the utilized link bandwidth, and are capable of constructing low-cost multicast trees that satisfy the constraints imposed by the QoS requirements. The delay constraint, i.e., the upper bound on end-to-end delay, is an important QoS requirement, because most real-time applications, and the interactive ones in particular, are delay-sensitive. A number of delay-constrained multicast routing algorithms have been proposed during the past few years. A delay-constrained shortest path heuristic was proposed in [12], and several cost-efficient, but quite complex, delay-constrained minimum Steiner tree heuristics were proposed in [13, 1, 14, 2]. A thorough evaluation of different multicast routing algorithms, unconstrained and delay-constrained, can be found in [15]. The destination set of the minimum Steiner tree problem may be any subset of the network nodes. In the special case when the destination set includes all nodes in the network, multicasting reduces to broadcasting, and the minimum Steiner tree problem reduces to the, usually less complex, minimum spanning tree (MST) problem. The delay-constrained minimum Steiner tree problem is *NP*-complete [13], and it remains *NP*-complete even after the delay constraint is removed [16]. We prove in this paper that the delay-constrained MST (DCMST) problem is also *NP*-complete. However, several polynomial time algorithms exist for the unconstrained MST problem [17, 18].

In the future, many real-time applications will involve all nodes in a given network. Some distributed real-time control applications and the broadcasting of critical network state information are just a few examples. For such applications, DCMSTs are needed to broadcast the real-time traffic from the source node to all other nodes in the network. The existing delay-constrained minimum Steiner tree heuristics, can be used to construct delay-constrained broadcast trees, but most of them are too complicated [15]. The comparison given in [15] shows that BSMA [2] is the most efficient delay-constrained minimum Steiner tree heuristic. The distributed delay-constrained minimum Steiner tree heuristic, DMCT, proposed in [1], is the only fast heuristic capable of generating DCMSTs. DMCT was designed for networks with symmetric link loads (undirected networks). In this paper, we propose an efficient DCMST heuristic for the general case of networks with asymmetric link loads (directed networks). Both our heuristic and DMCT are based on Prim's unconstrained MST algorithm [17]. We will show that the heuristic we propose outperforms DMCT, not only in the case of directed networks, but also in the special case of undirected networks. We also compare our heuristic to BSMA.

This paper is organized as follows. In section 2, we formulate the DCMST problem in directed networks and prove that it is *NP*-complete. In section 3, we present our heuristic for solving the problem. Then in section 4, we evaluate the average performance of the heuristic and compare it to optimum, DMCT, and BSMA using simulation. Section 5 concludes the paper.

2 Problem Formulation

A communication network is represented as a directed network $G = (V, E)$, where V is a set of nodes and E is a set of directed links. Any link $e \in E$ has a cost $c(e)$ and a delay $d(e)$ associated with it. $c(e)$ and $d(e)$ may take any positive real values.

A spanning tree $T(s) \subseteq E$ is rooted at a source node $s \in V$ and contains a path from s to any node $v \in (V - \{s\})$. The total cost of a tree $T(s)$ is simply:

$$Cost(T(s)) = \sum_{t \in T(s)} c(t) \tag{1}$$

A path $P(T(s), v) \subseteq T(s)$ is the set of tree links connecting s to $v \in V$. The cost of the path $P(T(s), v)$ is:

$$Cost(P(T(s), v)) = \sum_{t \in P(T(s),v)} c(t) \tag{2}$$

and the end-to-end delay along that path is:

$$Delay(P(T(s), v) = \sum_{t \in P(T(s),v)} d(t) \tag{3}$$

Thus the maximum end-to-end delay of a spanning tree is:

$$Max\_Delay(T(s)) = \max_{v \in V} \left(Delay(P(T(s), v))\right) \tag{4}$$

The DCMST problem in directed networks constructs the spanning tree $T(s)$ rooted at s that has minimum total cost among all possible spanning trees rooted at s which have a maximum end-to-end delay less than or equal to a given delay constraint Δ. The same problem can be expressed as a decision problem as follows.

Delay-Constrained Minimum Spanning Tree (DCMST) Problem: *Given a directed network $G = (V, E)$, a positive cost $c(e)$ for each $e \in E$, a positive delay $d(e)$ for each $e \in E$, a source node $s \in V$, a positive delay constraint Δ, and a positive value B, is there a spanning tree $T(s)$ that satisfies:*

$$Cost(T(s)) \leq B, \tag{5}$$

$$Max\_Delay(T(s)) \leq \Delta? \tag{6}$$

Theorem 1 *DCMST is NP-complete unless all link costs are equal.*

Proof. The DCMST problem in undirected networks (DCMST-undirected) is a restricted version of DCMST. We prove in appendix A that DCMST-undirected is *NP*-complete and therefore DCMST is also *NP*-complete. □

In the next section we propose a simple and efficient heuristic for the DCMST problem to avoid the exponentially growing execution times of the optimal solutions. We call it the bounded delay broadcasting (BDB) heuristic.

3 The BDB Heuristic

The BDB algorithm consists of two phases; phase 1 is executed once, followed by phase 2, which is also executed once. The result of phase 1 is a moderate-cost spanning tree which satisfies the delay constraint. phase 2 attempts to replace expensive links in this tree with cheaper links, without violating the delay constraint, and without introducing loops.

Phase 1 of BDB is outlined below:

```
1. Given $G = (V, E)$, a source node $s$, all link costs, and
   all link delays.  The initial subtree contains the source
   $s$ only.

2. Repeat $\{$

3. Find the cheapest link that connects an unconnected node
   to the already constructed subtree without violating
   the delay constraint.  Add that link to the subtree.

4. If step 3 can not find any link, then select a link $e =
   (u, v)$ which is not included in the subtree and $u$ and $v$
   are already in the subtree.  The link $e$ is chosen such
   that:

     • Replacing the link $l = (w, v)$ (this is the subtree
       link currently used to connect $v$ upstream towards
       $s$) with the selected link $e$ maximizes the delay relaxation.

   If no link that achieves positive delay relaxation can
   be found, the heuristic fails and exits, else remove
   $l$ and add $e$ to the subtree.

5. $\}$ until the tree spans all nodes.
```

The term delay relaxation can be defined as follows. Given are two alternate paths, P_1 and P_2, to connect the source s to some node $n \in V$, and $Delay(P_2) < Delay(P_1)$. If P_1 is the currently used path from s to n, the delay from s to n can be reduced by using the path P_2 instead of P_1, then the delay relaxation at n is the difference between $Delay(P_2)$ and $Delay(P_1)$. Phase 1 of BDB is based on Prim's MST algorithm. It starts with a subtree containing the source node s only, and adds one node at a time to the subtree without violating the delay constraint, as outlined above, until the subtree spans all nodes in the network. If at any point during the first phase, the heuristic can not find any node that can be added without violating the delay constraint, it resorts to delay relaxation. BDB relaxes the delays by choosing a node n that is already in the subtree and replaces the subtree link e_{before} connecting it upstream towards s with another link e_{after} such that n remains connected to s and the end-to-end delay from s to n is reduced. The node n is chosen such that the delay relaxation is maximized. If no more positive delay relaxation can be achieved and BDB fails to add any remaining unconnected nodes to the subtree, the algorithm fails. It is guaranteed, however, that the first phase of BDB will find a delay-constrained tree spanning all nodes if one exists, because in the worst case the heuristic keeps applying the delay relaxation step, and thus reducing the end-to-end delays to the individual nodes until it ends up with the shortest path tree that minimizes the end-to-end delay, the least-delay tree. If the least-delay tree can not span all nodes in the network without violating the delay constraint, no other tree can.

Below is the pseudo-code of phase 2 of BDB:

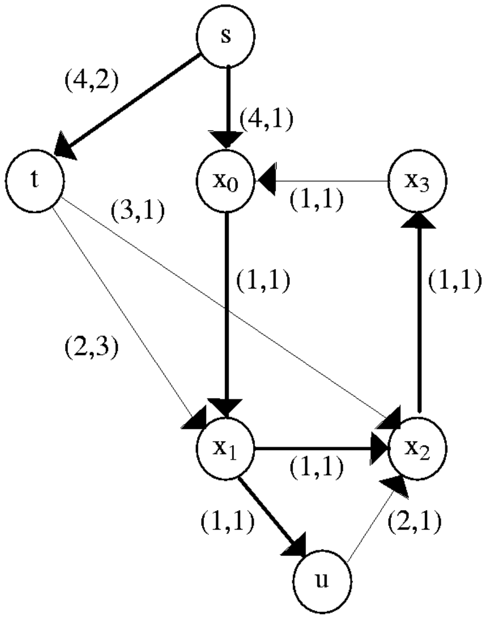
```
1. Given $G = (V, E)$, a source node $s$, all link costs, all
   link delays, and an initial delay-constrained tree spanning
   all nodes in $V$.

2. Repeat $\{$

3. Find the cheapest link $e = (u, v)$, $v \neq s$, which is not
   included in the subtree, and which satisfies the following
   conditions:
     If adding $e$ to the tree does not create a loop:

     • $c(e) < c(l)$ where $l = (w, v)$ is the tree link currently
       used to connect $v$ upstream towards $s$.

     • Replacing $l$ with $e$ does not cause any delay constraint
       violations along the tree.

     else:

     • Call the loop breaking algorithm, LBA. It attempts
       to break the loop such that the cost of the resulting
       tree is less than the cost prior to creating the
       loop and that the resulting tree does not violate
       the delay constraint.  If it returns success memorize
       the values returned for $l_{remove}$ and $l_{add}$.  If the loop
       breaking algorithm fails, the loop created by adding
       $e$ can not be broken.  Thus $e$ can not be used to reduce
       the cost of the tree.

4. If no such link can be found, the heuristic can not achieve
   further cost reduction and stops,
   else:

     • Remove $l$ and add $e$ to the tree.

     • If $e$ creates a loop, break the loop by removing the
       link $l_{remove}$ from the tree and replacing it with the
       link $l_{add}$.

5. $\}$ until no further cost reduction can be achieved.
```

In phase 2, expensive tree links are removed and replaced with cheaper links. The algorithm scans all links not already in the tree, and chooses the cheapest link, $e = (u, v)$ such that $c(e) < c(l)$, where $l = (w, v)$ is the link currently used to connect the node v upstream towards s. Another condition for choosing the link e is that using it to replace l in the tree does not cause any delay constraint violations. The chosen link e is then added to the tree, and link l is removed. Phase 2 repeats this operation until no more cost reduction can be achieved.

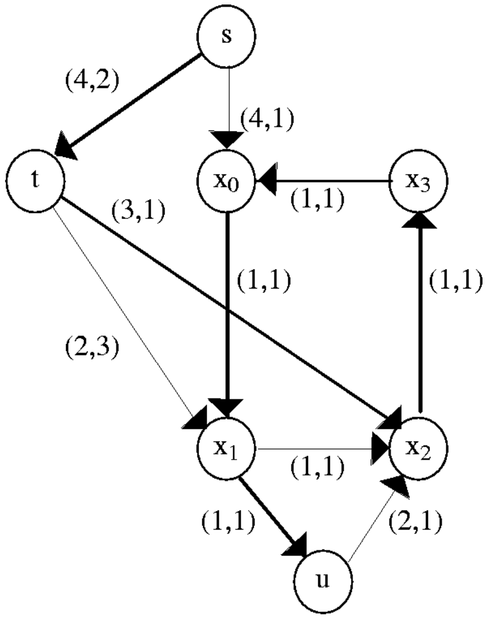
The link replacement operation of phase 2 may result in loops. Before a loop is created, the loop breaking algorithm (LBA) is applied. If LBA succeeds, the loop is created and broken according to its output, else the loop is not created in the first place. The pseudo code for LBA is shown below:

```
1. Given $G = (V, E)$, a tree $T(s) \subset E$, a potential loop path
   consisting of the set of nodes $X = \{x_0, ..., x_k\} \subset V$, and
   the set of links $L = \{l_1 = (x_0, x_1), ..., l_k = (x_{k-1}, x_k)\} \subset T(s)$,
   a link $l'_0 = (x_k, x_0)$ to close the loop if added, and a link
   $l_0 = (y, x_0)$ linking the potential loop path upstream towards
   the source.  $c(l'_0) < c(l_0)$.

2. $improvement := 0$, $l_{remove}$ and $l_{add}$ are not set.

3. Repeat for all nodes $x_i \in (X - x_0)$ $\{$

4. Find the cheapest link $l'_i$, to replace $l_i$ in connecting
   $x_i$ upstream towards the source, such that:

     • $c(l_0) + c(l_i) - c(l'_0) - c(l'_i) > improvement$
     • The resulting tree after removing $l_0$ and $l_i$ and replacing
       them with $l'_0$ and $l'_i$ does not contain any loops and
       does not violate the delay constraint.

   If such a link exists:  $improvement := c(l_0) + c(l_i) - c(l'_0) -$
   $c(l'_i)$, $l_{remove} := l_i$, $l_{add} := l'_i$.

5. $\}$

6. If $l_{remove}$ and $l_{add}$ are not set:  return failure,
   else return success and the values of $l_{remove}$ and $l_{add}$.
```

LBA attempts to select the proper location to break a given loop and to restore the tree structure without violating the delay constraint, and such that the resulting tree after creating and breaking the loop is cheaper. Using the same terminology of LBA's pseudo code, if link $l'_0 = (x_k, x_0)$ is added, a loop results. Removing link $l_0 = (y, x_0)$ disconnects the loop nodes, x_i, $i = 1, ..., k$, from the upstream tree. LBA scans each loop node and attempts to remove the link $l_i = (x_{i-1}, x_i)$ and to replace it with another link $l'_i = (z, x_i)$, $z \in V$. Removing l_i breaks the loop. The link l'_i is selected such that it reattaches the nodes of the broken loop upstream towards the source s. LBA attempts to select l_i and l'_i such that the resulting tree does not violate the delay constraint, and such that its cost is cheaper than the cost of the initial tree given to LBA. If there are multiple alternatives to break the loop, LBA chooses the alternative which results in the cheapest tree. If LBA can not find any links l_i and l'_i to break the loop, it fails, else it outputs the values of l_i and l'_i.



(a) Cost(T(s)) = 12, Max_Delay(T(s)) = 4.



(b) Cost(T(s)) = 11, Max_Delay(T(s)) = 7.

Figure 1: Total cost of a MC tree relative to optimal, unconstrained algorithms, 20 nodes, average degree 4.

Example: Figure 1 shows an example illustrating the loop breaking operation. Figure 1(a) shows the broadcast tree constructed by the first phase of BDB. Its cost is 13. In phase 2, BDB attempts to remove the link $l_0 = (s, x_0)$ and to replace it with the link $l'_0 = (x_3, x_0)$ in order to reduce the tree cost. However, this results in a loop $\{l_1 = (x_0, x_1), l_2 = (x_1, x_2), l_3 = (x_2, x_3), l'_0 = (x_3, x_0)\}$. LBA is called to determine how to break that loop. For the given case, LBA has only three alternatives. The first alternative is to remove the link l_1 and replace it with the link (t, x_1). This breaks the loop and results in a tree cost of 11, but the resulting tree would have a maximum end-to-end delay of $8 > \Delta$. Thus it is rejected. The second alternative is to remove the link l_2 and replace it with the link (u, x_2), thus breaking the current loop, but creating another one consisting of $\{(x_0, x_1), (x_1, u), (u, x_2), x_2, x_3), (x_3, x_0)\}$. This is also unacceptable. The only remaining alternative for the given example is to remove l_2 and replace it with (t, x_2) which results in an acceptable tree with a total cost of 12 and a maximum end-to-end delay of $7 = \Delta$ as shown in figure 1(b).

Kompella's DMCT algorithm [1] resembles phase 1 of BDB, but its approach to relax the delays is different from BDB's. However, DMCT does not have an equivalent to phase 2 of BDB. DMCT's execution stops as soon as all destination nodes are included in the tree being constructed, because its aim is to construct a Steiner tree and not a spanning tree.

In the next section we compare the performance of BDB to the performances of DMCT, BSMA, and the optimal delay-constrained minimum spanning tree algorithm, OPT. We implemented OPT as a branch and bound algorithm.

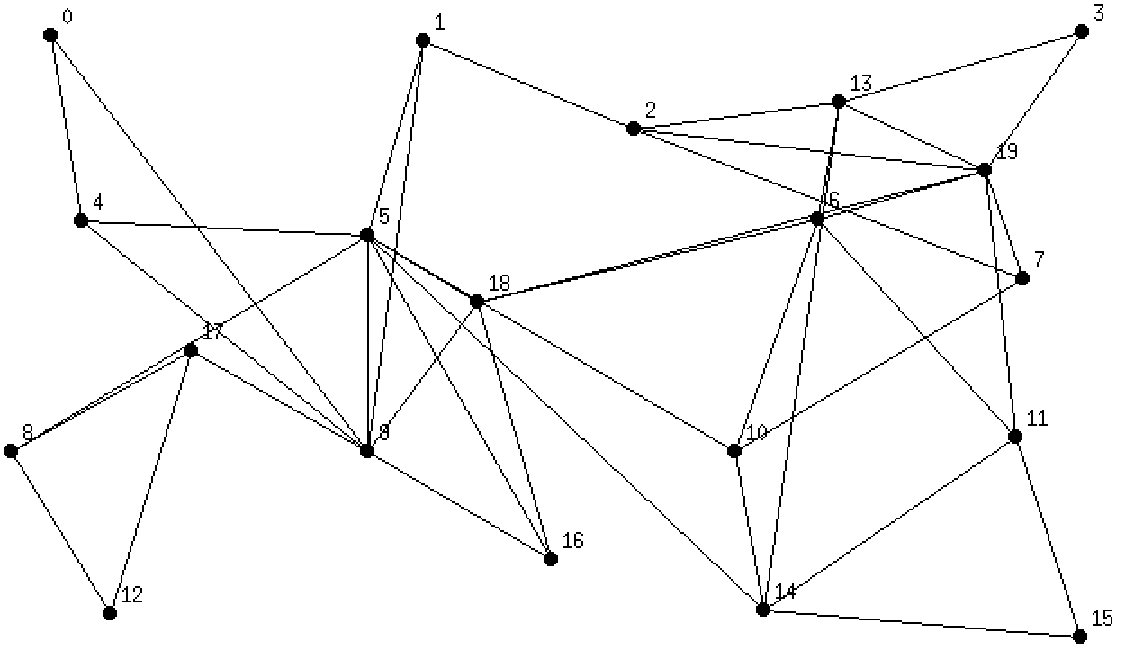


Figure 2: A randomly generated network, 20 nodes, average degree 4.

4 Experimental Results

We used simulation for our experimental investigations to avoid the limiting assumptions of analytical modeling. Full duplex directed networks of different sizes with homogeneous link capacities of 155 Mbps (OC3) were used in the experiments. The positions of the nodes were fixed in a rectangle of size $3000 * 2400$ Km^2, roughly the area of the USA. A random generator (based on Waxman's generator [19] with some modifications) was used to create links interconnecting the nodes. The output of this random generator is always a connected network in which each node's degree is ≥ 2. The probability of a link to exist between any two nodes is a function of the distance between these two nodes [19]. We adjusted the parameters of the random generator to yield networks with an average node degree of 4. Figure 2 shows an example of a randomly generated 20-node network.

The propagation speed through the links was taken to be two thirds the speed of light. The propagation delay was dominant under these conditions, and the queueing component was neglected when calculating link delays and end-to-end delays. Thus the link delays are symmetric, because the link lengths, and hence the propagation delays, are symmetric.

We used variable bit rate (VBR) video for the broadcast sources . These are realistic bursty traffic sources used in real-time applications with delay constraints. A broadcast tree that includes a link e, reserves a fraction of e's bandwidth equal to the equivalent bandwidth[3] of the traffic generated by the corresponding broadcast source. The link cost, $c(e)$, is equal to the reserved bandwidth on that link,i.e., equal to the sum of the equivalent bandwidths of the traffic streams traversing that link.

The experiment we ran, compares the different algorithms when each of them is applied to create

[3]Any other suitable measure could also be used.

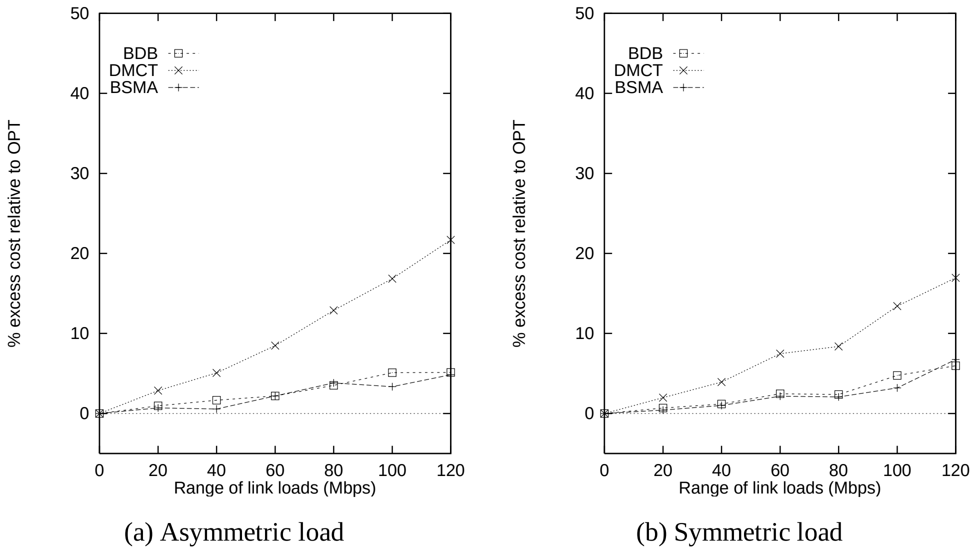


Figure 3: Percentage increase in the cost of a broadcast tree relative to optimal, variable range of link loads, 20 nodes, delay constraint $\Delta = 0.03$ seconds.

a broadcast tree for a given source node generating VBR traffic with an equivalent bandwidth of 0.5 Mbps, under given network loading conditions. For each run of the experiment, we generated a random set of links to interconnect the fixed nodes, we selected a random source node, and we generated random background traffic for each link. The equivalent bandwidth of each link's background traffic was a random variable uniformly distributed between B_{min} and B_{max}. This represents the initial cost of each link. For networks with symmetric link loads, the initial cost of a link $e = (u, v)$ and the initial cost of the reverse link $e' = (v, u)$ were set to the same random value. For networks with asymmetric link loads, however, the initial cost of $e = (u, v)$ and $e' = (v, u)$ were independent random variables. In that case, the asymmetry of the link loads increased as the range of the link loads, i.e., the difference between B_{max} and B_{min}, increased.

The experiment was repeated with different link loading conditions, different delay constraints, and different network sizes. For each setting of these parameters, we measured the total cost of the broadcast tree. We ran the algorithms repeatedly until confidence intervals of less than 5%, using 95% confidence level, were achieved. On the average, 300 different networks were simulated in each experiment, in order to reach such confidence levels. At least 250 networks were simulated in each case. We simulated the following algorithms: BDB, DMCT, BSMA, and OPT. We could not apply OPT to networks with more than 20 nodes due to its excessive execution time. The other algorithms were applied to networks with up to 200 nodes. Therefore, we show the percentage excess cost of each algorithm relative to OPT in case of 20-node networks only. When discussing results for networks with more than 20-nodes, we show the percentage excess cost of each algorithm relative to BSMA which was previously shown in [15] to be the best performing minimum Steiner tree heuristic.

Figure 3 shows the cost of a broadcast tree versus the range of link loads, $B_{max} - B_{min}$, for 20-node networks and a tight delay constraint of 0.03 seconds. We increased the range of link loads, $B_{max} - B_{min}$, from 0 to 120 Mbps while keeping the average link load, $(B_{max} + B_{min})/2$, fixed

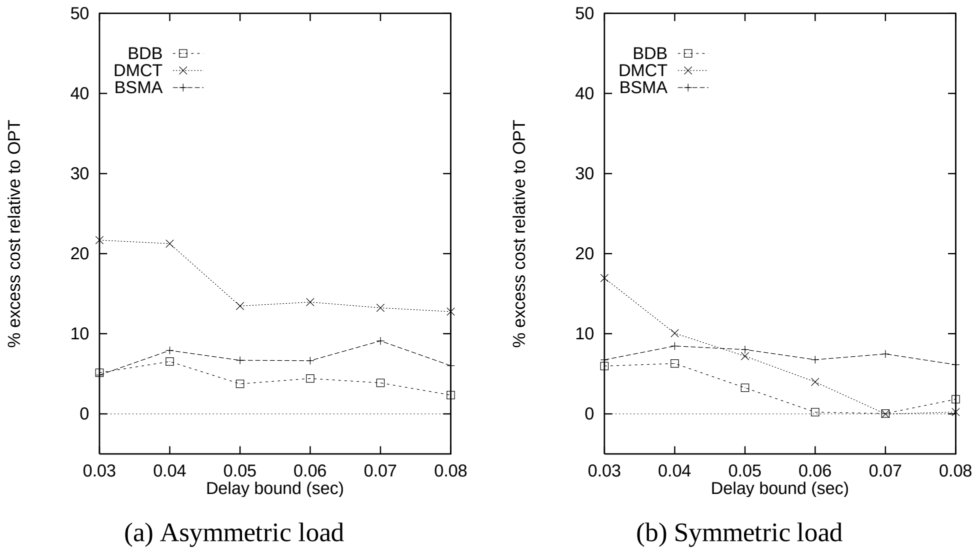
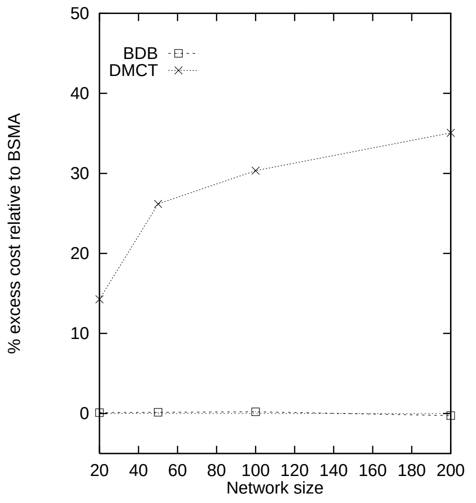


Figure 4: Percentage increase in the cost of a broadcast tree relative to optimal, variable delay constraint Δ, 20 nodes, B_{min} = 5 Mbps, B_{max} = 125 Mbps.

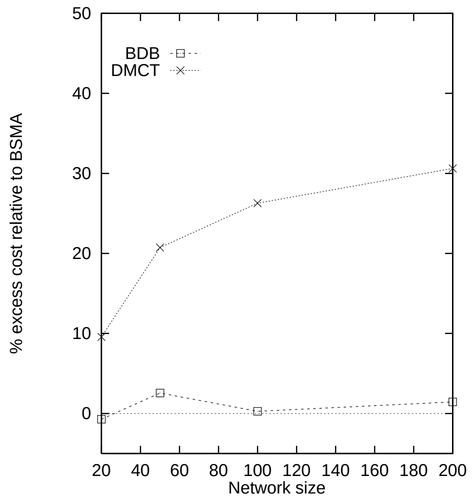
at 65 Mbps at all times. All algorithms perform equally well in case of zero range of link loads, because all link costs are equal and there is nothing to be minimized in that case. DMCT's costs also deviate further from optimal as the range of link loads increases. It is up to 23% worse than OPT when the link loads are asymmetric, but it performs slightly better in case of symmetric link loads and yield tree cost that are within 15% from OPT. BDB and BSMA remain close to optimal even when the range of link loads is large. The tree costs of BDB and BSMA are equal throughout the entire range of link loads simulated in this experiment. BDB's tree costs are within 7.5% from OPT when asymmetric link loads are used and only 5% off optimal in case of using symmetric link loads. We found that DMCT's tree costs are comparable to the costs of the intermediate trees resulting from phase 1 of BDB. Thus the phase 2 of BDB is capable of reducing the cost of its initial tree by up to 15% in some cases.

We repeated the experiment with a fixed range of link loads and a variable delay constraint. The results are shown in figure 4 for 20-node networks. When the delay constraint is tight, it limits the algorithms' ability to construct cheap trees, because there aren't many possible solutions for the problem. As the delay constraint increases, its effect on restricting the algorithms' efficiency in constructing cheap trees diminishes, and the algorithms are capable of constructing cheaper trees. When the delay constraint increases further, the effect on the resulting tree costs is minimal, because the solution of the DCMST problem approaches the solution of the unconstrained MST problem. This is evident in figure 4(b), where BDB and DMCT are almost optimal when the delay constraint exceeds 0.06 seconds. This happens because DMCT and phase 1 of BDB are based on Prim's algorithm which is optimal for the unconstrained MST problem in undirected networks. BDB performs slightly better than BSMA when the delay constraint is relaxed, both when the link loads are asymmetric and when they are symmetric.

Figure 5 shows the percentage excess costs of BDB and DMCT relative to BSMA when the network size varies from 20 nodes to 200 nodes while keeping the delay constraint and the range



(a) Asymmetric load



(b) Symmetric load

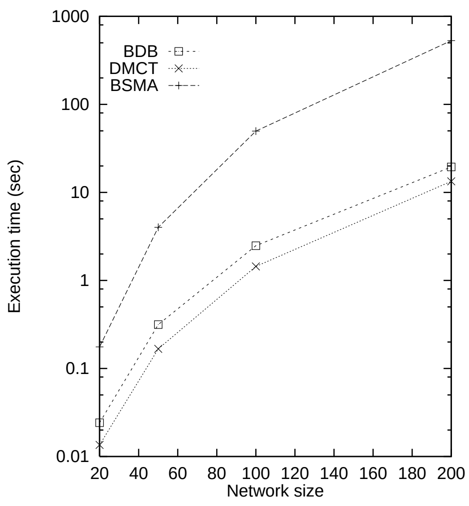
Figure 5: Percentage increase in the cost of a broadcast tree relative to BSMA's tree cost, variable network size, delay constraint $\Delta = 0.03$ seconds, B_{min} = 5 Mbps, B_{max} = 125 Mbps.

of link loads fixed. BDB performs as well as BSMA does throughout the entire range of network sizes, while DMCT's performance relative to BSMA deteriorates as the network size increases. For 200-node networks, DMCT's trees are 30% and 35% more expensive than BSMA and BDB depending on whether the link loads are symmetric and asymmetric respectively.

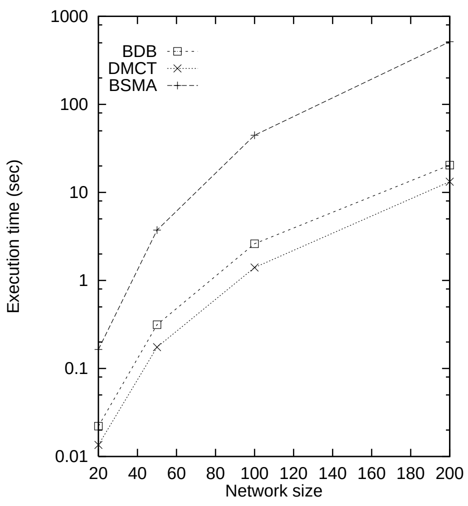
Finally, in figure 6, we present the average execution times of BDB, DMCT, and BSMA versus the network size. Note, however, that our code for these algorithms was not optimized for speed. Figure 6 shows that the average execution times of both heuristics grow at the same rate, and are always within the same order of magnitude, with BDB being constantly slower than DMCT by approximately 60%. BSMA is at least one order of magnitude slower than BDB and DMCT and its execution time grows at a faster rate.

5 Conclusions

Distributed real-time applications with QoS constraints will extensively utilize the resources of high-speed networks in the near future. We studied the problem of constructing broadcast trees for real-time traffic with delay constraints in networks with asymmetric link loads. We formulated the problem as a DCMST problem in directed networks, and then we proved that this problem is *NP*-complete. We proposed a bounded delay broadcast (BDB) heuristic to solve the DCMST problem. The heuristic consists of two phases. The first phase is based on Prim's algorithm [17] and constructs a moderate-cost delay-constrained spanning tree. The second phase reduces the cost of that tree by replacing tree link with cheaper links not in the tree without violating the imposed delay constraint. Simulation results show that BDB's performance is close to optimal in case of networks with asymmetric link loads as well as in the special case of networks with symmetric link loads. Delay-constrained minimum Steiner tree heuristics can be used to solve the DCMST



(a) Asymmetric load



(b) Symmetric load

Figure 6: Average Execution time, variable network size, delay constraint $\Delta = 0.03$ seconds, B_{min} = 5 Mbps, B_{max} = 125 Mbps.

problem. We compared BDB to the best delay-constrained Steiner heuristic with respect to tree cost, BSMA, and the fastest delay-constrained Steiner heuristic, DMCT. BDB is as efficient as BSMA in constructing cheap broadcast trees. DMCT is not as efficient as BDB. As the network size increases the difference between BDB and DMCT increases. DMCT's trees are up to 35% more expensive than BDB's trees when the network size is 200 nodes. DMCT was designed for networks with symmetric link loads, but even in that case BDB performs better. The average execution times of both BDB and DMCT grow at the same rate with DMCT being faster than BDB. BSMA's execution times are larger than BDB's execution times, and grow at a faster rate. In summary, our experimental results indicate that BDB is a fast and efficient DCMST heuristic.

Appendix A *NP*-Completeness of DCMST-undirected

Delay-Constrained Minimum Spanning Tree in Undirected Networks (DCMST-undirected) Problem: *Given an undirected network $G = (V, E)$, a positive cost $c(e)$ for each $e \in E$, a positive delay $d(e)$ for each $e \in E$, a source node $s \in V$, a positive delay constraint Δ, and a positive value B, is there a spanning tree $T(s)$ that satisfies:*

$$Cost(T(s)) = \sum_{t \in T(s)} c(t) \leq B, \tag{7}$$

$$Max\_Delay(T(s)) = \max_{v \in V} Delay(P(T(s), v)) \leq \Delta? \tag{8}$$

Theorem 2 *DCMST-undirected is NP-complete unless all link costs are equal.*

Proof. When all link costs are equal, the solution to the polynomial time least-delay tree problem (the shortest path tree in which link delays are used as a cost function) can be used to answer the DCMST-undirected decision problem. The DCMST-undirected problem is in *NP*, since a nondeterministic algorithm can guess a set of links to form the tree, then it is possible to verify in polynomial time that these links do form a tree, that this tree spans all nodes in the network, that the total cost of the tree is less than B, and that the maximum end-to-end delay from the source node s to any node $v \in V$ is no more than Δ.

The next step is to transform a known *NP*-complete problem to DCMST-undirected. We will use the Exact Cover by 3-Sets (X3C) problem which has been shown to be *NP*-complete in [20]. It can be stated as follows.

Exact Cover by 3-Sets (X3C) Problem: *Given a finite set $X = \{x_1, ..., x_{3p}\}$ and a collection $Y = \{y_1, ..., y_q\}$, $q \geq p$, of 3-element subsets of X, is there a subcollection $Y' \subseteq Y$ such that every element of X occurs in exactly one member of Y', i.e., the members of Y' are pairwise disjoint, and $\bigcup_{y \in Y'} y = X$?*

Given an arbitrary instance of X3C, we construct the network $G = (V, E)$ for the corresponding instance of DCMST-undirected as follows:

- Every element of X is represented by a node in the network, and also every member of Y is represented by a node. Two additional nodes are introduced: a source node s and another node t. Therefore:
$$V = s \cup t \cup X \cup Y \tag{9}$$
- Add the following set of undirected links E to interconnect the nodes:
$$\begin{aligned} E \;=\; & (s,t) \cup \{(s,y_i) : i = 1, ..., q\} \cup \{(t, y_i) : i = 1, ..., q\} \cup \\ & \{(y_i, x_j) : x_j \in y_i, i = 1, ..., q, j = 1, ..., 3p\} \end{aligned} \tag{10}$$
- Assign the following costs to the links:
$$c(e) = \begin{cases} 3 & e \in \{(s, y_i) : i = 1, ..., q\} \\ 1 & otherwise \end{cases} \tag{11}$$
- Unit delay is assigned to all links:
$$d(e) = 1 \quad \forall e \in E \tag{12}$$
- Set the delay constraint Δ of DCMST-undirected to:
$$\Delta = 2 \tag{13}$$
- Set the positive value B to:
$$B = 5p + q + 1 \tag{14}$$

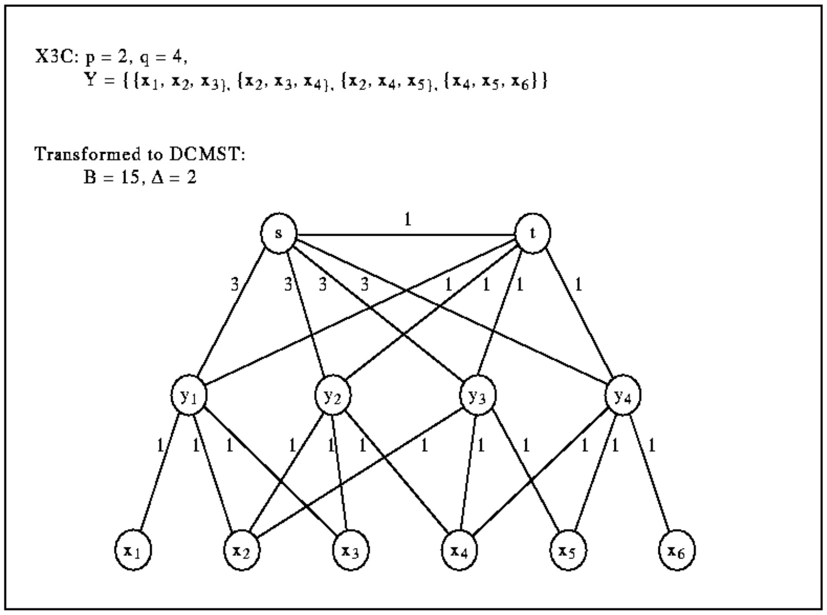


Figure 7: Equivalent instances of X3C and DCMST-undirected. Link costs are shown. All link delays are set to 1 (not shown).

Figure 7 illustrates this transformation. It is clear that it can be done in polynomial time. The final step is to show that a feasible spanning tree exists for the above instance of DCMST-undirected if and only if the 3-set collection Y has an exact cover Y'. If for a given instance of X3C, any element $x \in X$ is not in any member y of the collection Y then that instance does not have an exact cover. The above transformation, when applied to such an instance, will result in an unconnected network, thus no spanning tree can be constructed.

If, however, for a given instance of X3C, each element $x \in X$ appears in at least one member y of Y, the resulting network will be connected. Since all link delays are set to 1 and the delay constraint Δ is set to 2, the number of hops along any path starting at s in any feasible solution $T(s)$ can not exceed 2. As a result all nodes x_i, $i = 1, ..., 3p$, must be leaf nodes in any feasible solution. Each node x_i will be attached to some node $y_j \in Y$ via a unit cost link. The y_js used to reach the x_is must therefore be directly attached to s via the expensive links of cost 3. Cheaper but longer paths, via the node t, can be used from s to the y_js which are not used to reach the x_is. Let there be m nonleaf y_j nodes (y_js that are used to reach the x_is) and n leaf y_j nodes, where $q = m + n$. Thus the total cost of any spanning tree, $T(s)_{sat\_\Delta}$, that satisfies the delay constraint Δ is:

$$\begin{aligned} Cost(T(s)_{sat\_\Delta}) &= \overbrace{3p * 1}^{x_i s} + \overbrace{m * 3}^{\text{nonleaf } y_j s} + \overbrace{n * 1}^{\text{leaf } y_j s} + \overbrace{1 * 1}^{t} \\ &= 3p + 3m + (q - m) + 1 = 3p + 2m + q + 1 \qquad (15) \end{aligned}$$

Each component in equation 15 represents the cost of the links used to attach a set of nodes, indicated as the label of that component, upstream towards s. From equations 7, 14 and 15 it follows that the condition

$$m \leq p \tag{16}$$

must be satisfied for the total cost of the tree to be less than B. On the other hand, each of the m nonleaf y_js can be used to reach at most 3 x_is. Therefore

$$m \geq p \tag{17}$$

nonleaf y_j nodes are needed to reach the $3p$ x_is. Combining conditions 16 and 17, we get

$$m = p \tag{18}$$

as the only possible number of nonleaf y_js that results in a feasible solution to that instance of DCMST-undirected. For such a feasible solution to exist, each of the p nonleaf y_js must be used to reach exactly 3 different x_is, or in terms of the X3C problem: the members of the collection Y corresponding to the p nonleaf y_j nodes must be pairwise disjoint. p pairwise disjoint 3-sets cover $3p$ elements, and thus form an exact cover of the set X. This means that the existence of a feasible solution of an instance of DCMST-undirected implies the existence of a feasible solution for the corresponding instance of X3C.

Conversely, if an instance of X3C has an exact cover of X, that exact cover $Y' \subseteq Y$ must have exactly $|Y'| = p$ members to cover the $3p$ elements of X. The corresponding instance of DCMST-undirected will have a feasible solution with $|Y'|$ nonleaf y_js, a maximum end-to-end delay of $2 = \Delta$, and a total cost of $3p + |Y'| * 3 + |Y - Y'| * 1 + 1 = 3p + 3p + q - p + 1 = 5p + q + 1 = B$. This completes the proof. □

References

[1] V. Kompella, J. Pasquale, and G. Polyzos, "Two Distributed Algorithms for the Constrained Steiner Tree Problem," in *Proceedings of the Second International Conference on Computer Communications and Networking*, pp. 343–349, 1993.

[2] Q. Zhu, M. Parsa, and J. Garcia-Luna-Aceves, "A Source-Based Algorithm for Delay-Constrained Minimum-Cost Multicasting," in *Proceedings of IEEE INFOCOM '95*, pp. 377–385, 1995.

[3] J. Moy, "MOSPF, Analysis and Experience." Internet RFC 1585, May 1994.

[4] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol." Internet RFC 1175, November 1988.

[5] S. Deering *et al.*, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification." Internet Draft, September 1995. Available at ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-idmr-pim-spec-02.txt.

[6] S. Deering *et al.*, "Protocol Independent Multicast-Dense Mode (PIM-DM): Protocol Specification." Internet Draft, January 1996. Available at ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-idmr-pim-dm-spec-01.txt.

[7] A. Ballardie, “Core Based Tree (CBT) Multicast: Protocol Specification.” Internet Draft, February 1996. Available at ftp://ftp.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-idmr-cbt-spec-04.txt.

[8] R. Bellman, *Dynamic Programming.* Princeton University Press, 1957.

[9] E. Dijkstra, “Two Problems in Connectionwith Graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[10] Y. Dalal and R. Metcalfe, “Reverse Path Forwarding of Broadcast Packets,” *Communications of the ACM*, vol. 21, pp. 1040–1048, December 1978.

[11] S. Deering and D. Cheriton, “Multicast Routing in Datagram Internetworks and Extended LANs,” *ACM Transactions on Computer Systems*, vol. 8, no. 2, pp. 85–110, May 1990.

[12] Q. Sun and H. Langendoerfer, “Efficient Multicast Routing for Delay-Sensitive Applications,” in *Proceedings of the Second Workshop on Protocols for Multimedia Systems (PROMS)*, (Salzburg, Austria), October 1995.

[13] V. Kompella, J. Pasquale, , and G. Polyzos, “Multicasting for Multimedia Applications,” in *Proceedings of IEEE INFOCOM ’92*, pp. 2078–2085, 1992.

[14] R. Widyono, “The Design and Evaluation of Routing Algorithms for Real-Time Channels,” Tech. Rep. ICSI TR-94-024, University of California at Berkeley, International Computer Science Institute, June 1994.

[15] H. Salama, D. Reeves, Y. Viniotis, and T.-L. Sheu, “Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks,” in *Proceedings of the Sixth IFIP Conference on High Performance Networking (HPN ’95)*, pp. 27–42, Chapman and Hall, September 1995.

[16] R. Karp, “Reducibility among Combinatorial Problems,” in *Complexity of Computer Computations* (R. Miller and J. Thatcher, eds.), pp. 85–103, Plenum Press, 1972.

[17] R. Prim, “Shortest Connection Networks and Some Generalizations,” *The Bell Systems Technical Journal*, pp. 1389–1401, November 1957.

[18] H. Gabow, Z. Galil, T. Spencer, and R. Tarjan, “Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs,” *Combinatorica*, vol. 6, no. 2, pp. 109–122, 1986.

[19] B. Waxman, “Routing of Multipoint Connections,” *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, December 1988.

[20] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: W.H. Freeman and Co., 1979.