Two Distributed Algorithms for Multicasting Multimedia Information

Vachaspathi P. Kompella, Joseph C. Pasquale, George C. Polyzos*
{kompella, pasquale, polyzos}@cs.ucsd.edu
Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114.

Abstract

This paper addresses the problem of effective multicast tree construction for interactive audiovisual communication to several destinations. We believe that the effectiveness of a multicast tree for such applications is determined by two factors: (i) the end-to-end delay along the individual paths from source to each destination, *and* (ii) the cost of the multicast tree, for example, in terms of network bandwidth utilization. Note that while the cost of the tree should be minimized, it is sufficient simply to bound the delay. The problem of computing the optimal constrained multicast tree is *NP*-complete. We present two distributed algorithms that compute low cost trees with delay-bounded paths from the source to each destination.

Keywords: Multicast, interactive multimedia, Steiner tree, distributed algorithm, constrained optimization.

1 Introduction

Multicasting, the simultaneous transmission of data to multiple destinations, is now being viewed as a very important service in networks especially because video and audio applications can use it effectively. Many of the approaches to multicast route determination can be formulated as graph theoretic problems relating to tree construction. There are two reasons for basing multicast routes on trees: (i) the data can be transmitted in parallel to the various destinations along the branches of the tree, and (ii) a minimum number of copies of the data are transmitted, with duplication of data being necessary only at forks in the tree.

Centralized algorithms for constructing multicast trees have been developed with two optimization goals in mind. The first is the minimum average path delay, $\mathcal{D}_T$, which is the average of the minimum path delays from the source to each of the destinations in the multicast group. A minimum average path delay tree can be constructed in $O(n^2)$ time using Dijkstra's shortest path algorithm [3], where n is the number of nodes in the graph. The second measure of efficiency is in terms of the cost of the multicast tree, $\mathcal{C}_T$, which is the sum of the costs on the edges in the multicast tree. The least cost tree is called a *Steiner tree* [5], and the problem of finding a Steiner tree is known to be *NP*-complete [7].

Several centralized algorithms that construct low-cost multicast routes [2, 6, 13, 14] are based on heuristics for approximate Steiner trees [10, 11, 12]. Empirical observations show that the heuristics produce near optimal trees quickly. These algorithms take polynomial time, ranging from $O(n^3)$ to $O(n^4)$.

The two measures $\mathcal{C}_T$ and $\mathcal{D}_T$ are individually insufficient to characterize a good multicast route for interactive multimedia communication. The performance of such a multicast route is determined by two factors: (i) bounded end-to-end delay along the individual paths from source to each destination, *and* (ii) minimum cost of the multicast tree, for example, in terms of network bandwidth utilization. In our formulation, a multicast tree is a *constrained Steiner tree*, *i.e.*, a delay-bounded minimum cost tree, where the delay bound is specified by the application performing the multicast.

A centralized approach to determining a constrained Steiner tree can be found in [8, 9]. The centralized algorithms perform very well, on the average, generating trees that cost about 5-10% more than the optimal. In dealing with larger networks, where the cost of maintaining consistent information at all nodes regarding the network status may be prohibitive, a distributed approach may prove cheaper.

We present two distributed algorithms using different heuristics for selecting edges in the tree. We compare the performance of the distributed algorithms against the centralized approach. We note that the distributed algorithms come to within 15-30% of the centralized algorithms. Furthermore, we demonstrate that the shortest delay algorithm is inferior to the distributed algorithms. We also posit that, in selecting edges for inclusion in the constrained Steiner tree, it is advantageous to factor in the delay that is incurred on a path from the source through that edge. From earlier results (see [8]), the same is not true for the centralized algorithms that we developed.

*This work was supported by grants from Digital Equipment Corp., IBM, UC Micro, NCR, TRW, and the National Science Foundation.

2 The Constrained Steiner Tree

The constrained Steiner tree (CST) problem can be formulated as follows. Given a graph $G = (V, E)$ with node set V and edge set E, we define the following weight functions on the edges:

$$\mathcal{C} : E \rightarrow \mathbb{R}^+, \text{a positive real edge cost function}$$

and

$$\mathcal{D} : E \rightarrow \mathbb{Z}^+, \text{a positive integer edge delay function}$$

On this graph, we designate a source node s and a set of destination nodes S, called the multicast group. Given a delay tolerance Δ, a *constrained spanning tree* T is a tree rooted at s, that spans the nodes in S, such that for each node v in S, the delay on the path from s to v is bounded above by Δ. We can now describe the constrained spanning tree formally as follows. For each $v \in S$, if $P(s, v)$ is the path in T from s to v,

$$\sum_{e \in P(s,v)} \mathcal{D}(e) < \Delta$$

The constrained Steiner tree can now be described as a constrained spanning tree such that

$$\sum_{e \in T} \mathcal{C}(e) \text{ is minimized}$$

We note that this problem is *NP*-complete, as shown in [9].

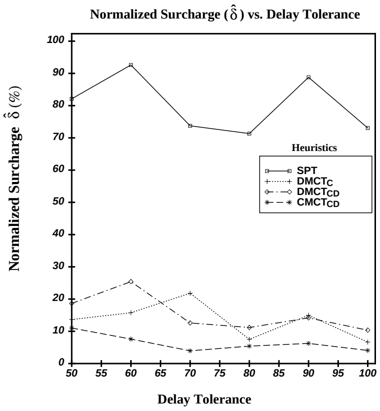

Figure 1: The graph used in explaining the heuristics.

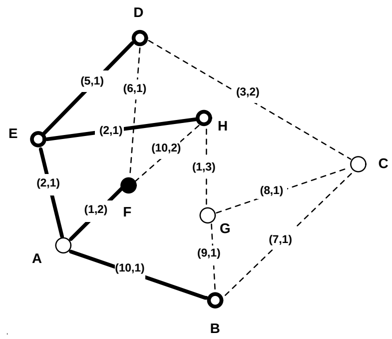

Figure 2: The optimal constrained Steiner tree on $S = \{$B,D,E,H$\}$ with $\Delta = 5$, and $s =$ F.

3 Distributed Algorithms for the Constrained Steiner Tree

When a network is small, it is possible to maintain a consistent view of the network status by broadcasting this information. As the network gets larger, this process becomes less reliable, less accurate, and more expensive. We will present two distributed algorithms that attempt to use only information that is passed on by neighboring nodes. We will also compare the performance with the tree of shortest delay paths. We will demonstrate the working of the two distributed algorithms on the graph in Fig. 1. In the graph, the numbers along each edge represent the cost and delay for that edge. The source node is F, and the multicast group is {B, D, E, H}. The delay constraint is 5. Thus, all paths must have a delay between 1 and 4. Fig. 2 shows the optimal tree which has a cost of 20.

3.1 Centralized Routing Algorithms

We briefly describe the centralized algorithms, but the reader is referred to [8] for further details. The basic strategy is to construct a closure graph on the source and destination nodes only. The closure graph encapsulates the costs of the cheapest delay-bounded paths between any two nodes, and the corresponding delays. A minimum spanning tree rooted at the source could then produce an approximate constrained Steiner tree. By choosing edges carefully, the centralized algorithms can guarantee a solution, which appears, from experimental results, to be close to optimal. We now examine three approaches that use only a small amount of local information, *i.e.*, information passed on to nodes by neighbors.

3.2 Distributed Routing Algorithms

The first approach, *SPT*, simply constructs the tree of minimum delay paths that connect the source to each destination. This is a distributed version of the Bellman-Ford shortest path algorithm [1]. Obviously, the tree has to be pruned to remove unnecessary branches. It is guaranteed to be successful in finding a constrained spanning tree (if one exists), but does not attempt to optimize on the cost of the solution. Fig. 3 shows the solution produced by *SPT* when run on our example graph. The tree produced by *SPT* has a cost of 32.

We now present two distributed algorithms using heuristics f_C and f_{CD} to select edges in the tree, which are described in detail below. The general form of the two dis-

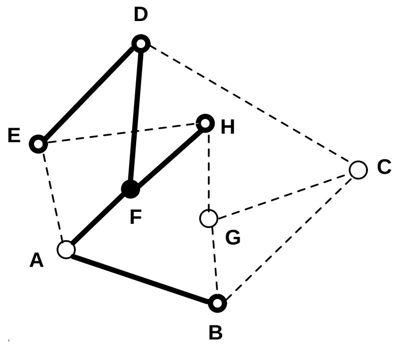

Figure 3: The tree produced by using the shortest delay paths from source to destinations.

tributed algorithms is the same, differing only in the criterion they apply to select edges. These two algorithms are based on a distributed minimum spanning tree (MST) algorithm. Essentially, the algorithm mimics Prim's MST algorithm. The reader can find more details on the distributed MST algorithm in [4].

The most significant problem with using the distributed MST algorithm to construct an approximate constrained Steiner tree is that there is no way to know *a priori* whether the tree is growing in the right direction. Since the distributed MST algorithm does not place any restrictions on delay, it is possible that a tree grown by such an MST algorithm fails to meet delay bounds. To avoid this, a first approach could be to discard edges that cause a delay bound violation. However, this is not a strong enough condition to guarantee that the distributed MST algorithm will always produce a delay constrained tree, when one does exist.

To illustrate this point, consider Fig. 4, an instance where A is the source, D is the destination and $\Delta = 3$, so that the path delay from A to D should not exceed 2. If edges were chosen based on cost, then the edges AB and BC would be selected over AC because they are cheaper. The algorithm would then terminate declaring no solution exists because selecting the edge CD would cause a delay bound violation. However, $\{\mathrm{AC}, \mathrm{CD}\}$ form a solution.

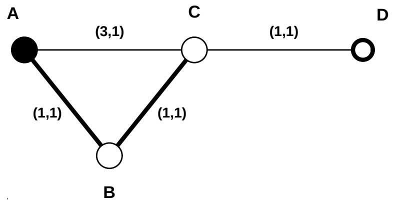

Figure 4: Example where $DMCT_C$ and $DMCT_{CD}$ fail. After AB and BC have been selected, D cannot be reached. $\Delta = 3$.

One way to resolve the problem above is to choose an edge that causes a cycle, and then break the cycle by deleting some edge (or edges) from the tree. In the above case, A has very few choices, so the resolution works. However, in general, there are no guidelines or hints as to which edge to add to form the cycle, and which to delete to break the cycle.

3.2.1 The Problem of Delay Bounds

The problem with nodes having no delay information other than the adjacent link delays is that this very limited horizon does not permit them to decide whether the direction being explored is delay-bounded or not. This problem can be obviated if each node has available the minimum delay to every other node in the graph (*e.g.* by running the Bellman-Ford distributed algorithm on the delay metric). Using this, the above incorrect termination can be circumvented in the following way. At the first stage, A would have to decide between edges AB and AC. When B responds to a query from A regarding the minimum delay from B to D, A can easily conclude that with a delay of 2 from B to D, no solution exists that includes AB. A can then turn to C, and build the correct solution.

We will return to the problem of possible termination without a solution again, as this can happen even with our algorithm. However, this time, we shall demonstrate that there is a convenient procedure to decide how to resolve this problem, by choosing to make and break cycles in such a way that the number of reachable destinations increases. As a consequence, we know that with k destinations, the maximum number of cycles that have to be resolved cannot exceed $k-1$. The algorithm is given in Appendix 1.

Every edge (v, w) is initially marked *Unknown*. As nodes receive responses to TEST and ADJUST messages, they change the status of edges to *Usable*, *Cycle* or *Unusable*. Before sending an ADD message for edge (x, y), node x marks (x, y) as *Tree*. On receiving the ADD message, node y marks edge (y, x) as *Reverse*.

Every node is initially marked *Unseen*. As it is added to the tree, it marks itself *Seen*. The set of unreached destinations S' is initially the set of all destinations S. As destination nodes are added to the tree, s removes them from S', and when necessary, reports (in the FIND message) the newest destination node added. In that way, nodes get to know what S' is. Every new node in the tree is communicated the set S' in the ADD message. Finally, when the source determines that all the nodes to be covered are in the tree, the source can propagate a message up the tree, to eliminate unnecessary branches that lead to nodes not in the destination set.

3.2.2 Edge Selection Functions

The important question to determine here is how to choose between competing edges. Several heuristics can be used to select the best edge to extend the subtree. We have looked at two edge selection functions. Firstly, it could be argued that cost is the over-riding determinant of the route. Thus, a first choice could be: "Select the cheapest edge out-going from the subtree constructed so far. Ensure that all destinations are either reachable or have been reached within the delay bound." We call this selection function f_C, and the consequent variant of the distributed algorithm $DMCT_C$.

$$f_C = \begin{cases} \mathcal{C}(v,w) & \text{if } \mathcal{P}(v) + \mathcal{D}(v,w) < \Delta \\ \infty & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} \mathcal{C}(v,w) &= \text{cost of the link } (v,w) \\ \mathcal{D}(v,w) &= \text{delay on the link } (v,w) \\ \mathcal{P}(v) &= \text{delay on path from } s \text{ to } v \text{ in the tree} \end{aligned}$$

A second choice could take into account the fact that delay also plays a prominent role. We opted to use a function that modulates the cost of an edge by a factor that biases the choices towards lower delay. Define the residual delay at a node v as $\Delta - P(v)$. Then, it would be desirable to have a large value of residual delay. If the residual delay on a path is large, then the likelihood of reusing that path to reach other destinations improves. This tends to lower the cost of the tree. The function we used was simply the cost of an edge divided by the residual delay of that path. We call this selection function f_{CD}, and the corresponding variant of the distributed algorithm $DMCT_{CD}$.

$$f_{CD} = \begin{cases} \frac{\mathcal{C}(v,w)}{\Delta - [\mathcal{P}(v) + \mathcal{D}(v,w)]} & \text{if } \mathcal{P}(v) + \mathcal{D}(v,w) < \Delta \\ \infty & \text{otherwise} \end{cases}$$

Fig. 5 shows the edges $DMCT_C$ uses in constructing a multicast tree. In the first step (shown in Fig. 5 (a)), F can choose one of three edges, FA, FD, and FH with f_C values 1, 6 and 10. Thus, it chooses FA. The tree constructed by $DMCT_C$ has a cost of 20.

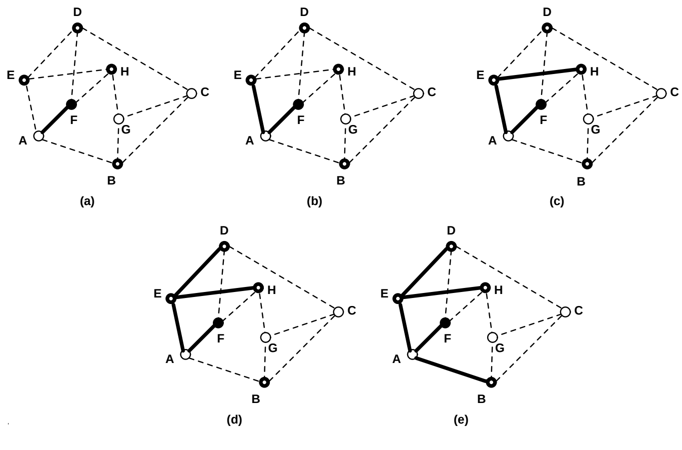

Figure 5: (a)-(d) Four stages in producing the tree using f_C. (e) The final tree produced by $DMCT_C$.

Fig. 6 shows an instance of the working of $DMCT_{CD}$. The tree generated by $DMCT_{CD}$ has a cost of 25. In the first step (shown in Fig. 6 (a)), F can choose one of three edges, FA, FD, and FH with f_{CD} values 0.3, 1.5, and 3.3, respectively. Thus, it chooses FA. We note that the algorithm also picks edge DC. However, we prune the tree to remove branches that lead to leaf nodes not in S.

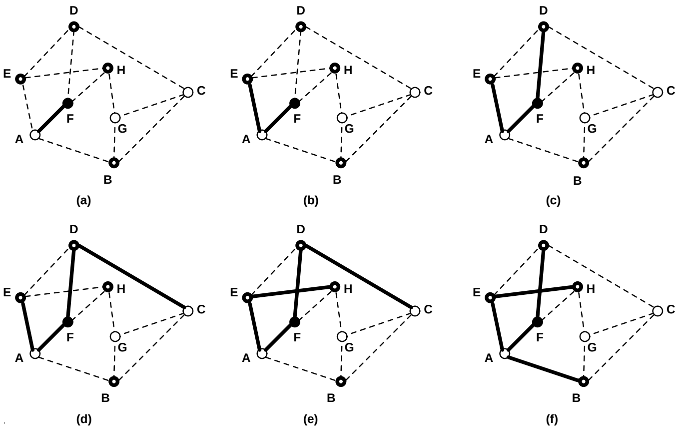

Figure 6: (a)-(e) Five stages in producing the tree using f_{CD}. (f) The final tree produced by $DMCT_{CD}$.

3.2.3 Eliminating False Termination

We now address the problem of false termination. It is still possible to run into a situation where the algorithm reports no solution exists, while a solution does exist. Consider for example Fig. 7(a). The source is A, the destinations are $\{C, D\}$ with a delay constraint $\Delta = 6$. If f_C is being used, then the first step would be to choose AU, the second to choose AV, the third to select UW, and the fourth to pick WD, to get to the position in Fig. 7(b). Now it is impossible to meet the delay constraints to all the destinations. However, at this point we know what to do. Let the current tree be called $T = \{AU, AV, UW, WD\}$. After an ADJUST message is sent out, the source asks that a cycle be created by adding VW, and that it be broken by deleting the edge from W on the return path to A, *i.e.* WU, thus obtaining $T' = \{AU, AV, VW, WD\}$. Since the delay on the path from A to W in T' is smaller than the path from A to W in T, all the nodes reachable from W are still reachable in T'. However, now C is also reachable. Now complete the algorithm, and prune the unnecessary edge (AU), to yield the tree in Fig. 7(c).

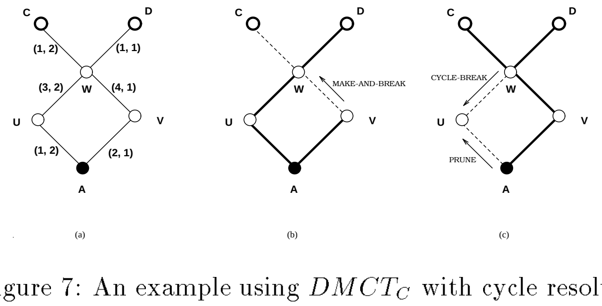

Figure 7: An example using $DMCT_C$ with cycle resolution.

This cycle "make-and-break" phase can now be included in the algorithm, between steps 6 and 7. It is given in Appendix 2.

4 Performance Analysis of the Heuristics

In [8], we showed that the performance of the centralized heuristics was very good. In fact, $CMCT_C$, the centralized heuristic that uses f_C, comes to within 5-10% of the cost of the optimal in the empirical performance evaluation of the centralized algorithms. Again, because the cost of comparing heuristics with the optimal is prohibitive, we compared the distributed algorithms against a pseudo-optimal heuristic, $CMCT_C$, which is the better of the two centralized heuristics, and the best among all the heuristics that we have come up with so far. The metric we used to compare the heuristics is the normalized surcharge with respect to the pseudo-optimal, given by:

$$\widehat{\delta}_H = \frac{T_H - T_{CMCT_C}}{T_{CMCT_C}}$$

where

$$T_H = \text{cost of tree using heuristic } H$$
$$T_{CMCT_C} = \text{cost of tree using heuristic } CMCT_C$$

4.1 Empirical Performance Analysis

The heuristics were run on randomly generated graphs with low average degree. The graphs we looked at in the evaluation of the heuristics had between 50 and 100 nodes, and an average degree between 5 and 15.

The nodes in the graphs were randomly placed in a unit square, and the edge delays were proportional to the Euclidean distance separating the endpoints. We used both unit edge costs and random edge cost generated uniformly from the set [1, 10]. The random edge costs match typical values for costs used in the NSFNET backbone network. However, earlier results indicate that the properties of the heuristics and their relative performance remain the same for both unit costs and random costs. Each experiment generated 100 graphs with identical parameters: number of nodes, maximum degree of each node, delay bound, and size of the multicast group. Each graph was then checked to ensure that a solution existed. The confidence interval for δ at the 95% confidence level is about 1-2% for all the graphs shown.

An interesting feature of the graphs in Fig. 8 is that $DMCT_C$ outperforms $DMCT_{CD}$ for large graph sizes. Thus, as the tree size increases, it is apparent that introducing a delay component in the selection function becomes important for the success of the distributed algorithms. However, the same is not true for the centralized algorithms. Since they operate on a closure graph, which has an implicit delay component because of the manner in which it is constructed, it is better to use f_C than f_{CD} because there is no unnecessary trading of cost for lower delay.

For large multicast group sizes, the algorithms $CMCT_{CD}$, $CMCT_C$, $DMCT_{CD}$, and $DMCT_C$ converge to the minimum spanning tree. This is borne out by Fig. 9, where the deviation from $CMCT_C$ is very small.

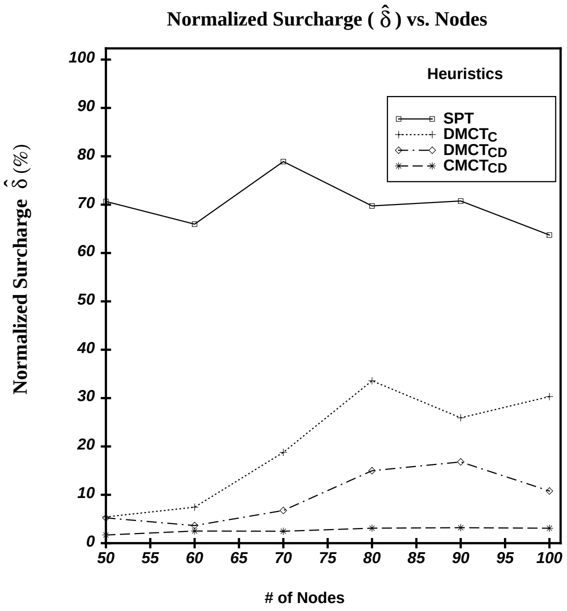

Figure 8: Normalized surcharge $\widehat{\delta}$ vs. number of nodes, for group size = 15, Δ = 80, and maximum degree = 30, when compared with $CMCT_C$.

For large delay tolerances, the performance of $DMCT_{CD}$ improves marginally, as can be seen from Fig. 10. This is to be expected, by observing that f_{CD} behaves like f_C when the value of Δ in the denominator is very large. The same is true for $CMCT_{CD}$ and $CMCT_C$.

4.2 Complexity of the Heuristics

If the source knows the network status, then Dijkstra's shortest paths algorithm takes $O(n^2)$ time [3]. The distributed Bellman-Ford algorithm for the shortest paths takes $O(n^3)$ time [1].

The distributed algorithms $DMCT_C$ and $DMCT_{CD}$ are based on the distributed MST algorithm. However, there are four differences. Firstly, instead of allowing the tree to be built up in fragments that eventually connect together, this tree is built from a single root. This corresponds to the distributed MST algorithm running with a single node woken up initially. This would cause the distributed MST algorithm to run with $O(n^3)$ number of messages [4]. Thus, the number of messages for $DMCT_C$ or $DMCT_{CD}$ would be $O(n^3)$ also. A second difference is that in the distributed MST algorithm, the nodes at which the decision on which edge should be added next moves to the endpoints of the new edge added to the tree. In our algorithms, this point of decision is at the source. However, this does not add to the number of messages sent. Finally, $DMCT_C$ and $DMCT_{CD}$ have two more phases. Firstly, the cycle make-and-break phase. The cycle-making phase requires an extra set of CYCLE-ADD messages to be sent from the source to each node in the tree,

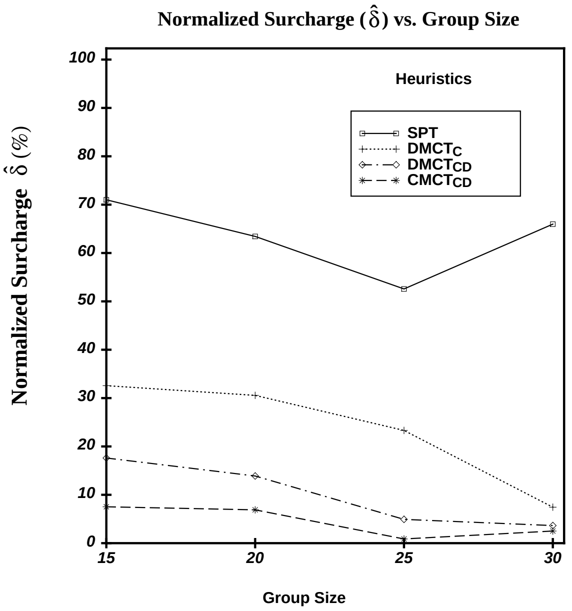

Figure 9: Normalized surcharge $\widehat{\delta}$ vs. group size, for number of nodes = 60 , and Δ = 80, with average degree of each node = 5, when compared with $CMCT_C$.

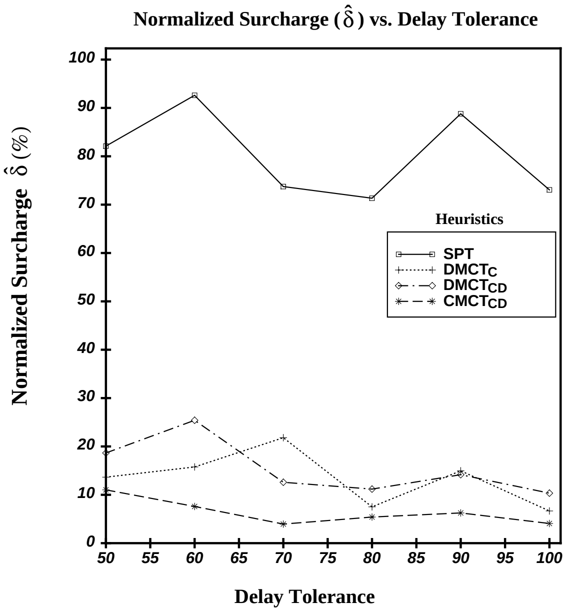

Figure 10: Normalized surcharge $\widehat{\delta}$ vs. delay tolerance for number of nodes = 50, and group size = 20, with average degree of each node = 10, when compared with $CMCT_C$.

a MAKE-AND-BREAK message, and a CYCLE-BREAK message. These can total at most $O(n)$ for each cycle breaking. This is because there can be at most $n-1$ edges in a tree. As noted before, each time a cycle is made and broken, the number of unreachable destinations decreases by 1. Thus, within $O(n^2)$ messages, the cycle make-and-break phases can be completed, still yielding an $O(n^3)$ algorithm.

5 Conclusions

Multimedia communications researchers are increasingly aware of the inadequacies of existing network protocols and algorithms to handle continuous media like audio and video. We have addressed one aspect of the problem, *i.e.*, routing algorithms that take into account quality of service parameters required to make interactive multimedia communication successful.

The analysis presented in [9] shows that the problem of multicast routing as presented is intractable. We have shown that there are fast distributed algorithms that produce extremely good solutions. They can scale to large sized graphs and still provide low cost routes (in terms of network resources expended to support a multicast) with bounded path delays. We also see that introducing a delay component into the function that determines the desirability of an edge in the graph yields lower cost trees. However, the standard routing algorithms that use minimum delay perform very badly, even when compared to the distributed algorithms.

References

[1] R.E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N.J., 1957.

[2] C.-H. Chow. On multicast path finding algorithms. *Proc. of IEEE INFOCOMM '91*, New York, NY, pp. 1274–1283, 1991.

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

[4] R.G. Gallager, P.A. Humblett, and P.M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Prog. Lang. Syst.*, 5:66–77, 1983.

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

[6] B.K. Kadaba and J.M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Trans. on Comm.*, COM-31(3):343–351, Mar. 1983.

[7] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York, 1972.

[8] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicast routing for multimedia communications. *ACM/IEEE Trans. on Networking.*, Jun. 1993.

[9] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos. Multicasting for multimedia applications. *Proc. of IEEE INFOCOMM '92*, Florence, Italy, May 1992.

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

[11] V.J. Rayward-Smith. The computation of nearly minimal Steiner trees in graphs. *Intl. J. Math. Educ. Sci. Tech.*, 14(1):15–23, 1983.

[12] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 6:573–577, 1980.

[13] D.W. Wall. *Mechanisms for Broadcast and Selective Broadcast*. PhD thesis, Elect. Eng. Dept., Stanford University, Jun 1980.

[14] B.M. Waxman. Routing of multipoint connections. *IEEE J. on Selected Areas in Comm.*, 6(9):1617–1622, Dec 1988.

6 Appendix 1

1. From the source s, send a FIND message up the tree formed so far to each node v to determine the best out-going edge.
2. At node v, the best out-going edge is an edge determined by:
   a. sending a TEST message on all adjacent edges (v, w)
   b. asking each adjacent node w which unreached destination nodes $d \in S'$ are reachable, if they are to incur a path delay of $P(v)$ from the source to v and $SD(w, d)$ from w to d
   c. for each edge (v, w), if at least one destination is reachable, and w is not in the tree, then mark that edge as *Usable*
   d. if at least one destination is reachable, but w is already in the tree, then mark that edge as *Cycle*
   e. if no destination is reachable, mark that edge as *Unusable*
   f. for each *Usable* edge, determine the desirability of that edge using some heuristic, $f(v, w)$. The edge with the minimum value of $f(v, w)$ is the best out-going edge
3. Propagate the best out-going edge back down the tree towards the source, replacing it by better choices (*i.e.*, with smaller values of $f(v, w)$) at other nodes along the way.
4. At the source s
   a. choose the best edge (x, y), and add it to the tree by sending an ADD message up the branch that contains it
   b. if y is a destination node, remove it from S', the set of unreached destinations
5. At the node x, the node to which the new edge (x, y) is to be added, mark the edge (x, y) as *Tree*
6. At the new node y just added to the tree
   a. mark y as being in the tree
   b. mark edge (y, x) as *Reverse* to denote that it is the first edge in the return path to s from y
   c. send an ADJUST message along each adjacent edge (except along (y, x)) notifying the endpoints that if they are already in the tree, this edge should be marked as a *Cycle* edge
   d. compute the path delay $P(y)$ from s to y to be the path delay from s to x plus the link delay on (x, y), *i.e.*, $P(y) = P(x) + \mathcal{D}(x, y)$
7. If a node v receives an ADJUST message to mark an edge as a *Cycle* edge
   a. if that edge is already known to be *Unusable*, do not change its status
   b. otherwise that edge must already be *Usable*, so change its status to *Cycle*
8. Repeat this process until all the destinations have been covered.
9. Prune the tree of unnecessary edges.

7 Appendix 2

6.1 At s, if no best out-going edge is passed down, then send a CYCLE-MAKE message up the tree.

6.2 At node v, when a CYCLE-MAKE message is received, forward it up the tree, look at each adjacent *Cycle* edge (v, w) that still leads to an unreached destination, and pick the most desirable of them, according to the selection function $f(v, w)$

6.3 The best out-going cycle-making edges are passed down the tree to the source s.

6.4 At the source, send a CYCLE-ADD message up the tree with the edge that is determined to be the best.

6.5 At node x, if a CYCLE-ADD message is received with edge (x, y), send a MAKE-AND-BREAK message to y.

6.6 At node y, if a MAKE-AND-BREAK message is received from node x

   a. send a CYCLE-BREAK message down its *Reverse* edge
   b. reset the status of the *Reverse* edge to *Unknown*
   c. reset the path delay to $P(x) + \mathcal{D}(x, y)$
   d. mark edge (y, x) as *Reverse*

6.7 At node x, if a CYCLE-BREAK message is received from node w, reset the status of edge (x, w) to *Cycle*.