Routing Reserved Bandwidth Multi-point Connections.

Dinesh C Verma
P. M. Gopal
IBM TJ Watson Research Center
PO Box 704,
Yorktown Heights, NY 10598

Abstract

Some important classes of multi-point bandwidth-intensive applications like video-conferencing with mixing and the distributed classroom can be characterized as consisting of a broadcast from a source node to several destinations nodes, and point-to-point flows from the destination nodes to the source node. Determining a tree in an arbitrary mesh network which satisfies the bandwidth constraints and minimizes the cost of reserved bandwidth is a NP-hard problem. In this paper, we look at some heuristics that can be used to solve the problem of routing these multi-point connections. The heuristics are based on finding the capacity-constrained minimum cost tree which minimizes the cost of bandwidth reserved for point-to-point communication from destinations to the source, and weights are assigned to minimize the number of extra nodes in the tree which increase the cost of bandwidth reserved from the source to the destination. A theoretical bound on the performance of some of the heuristics, as well as simulation results comparing their performance to that of the optimum solution are presented. The results are encouraging, the heuristics find a tree with a cost within 2% of the optimum on the average, and with a cost within 10% of the optimum in those cases when the heuristic fails to find the optimum tree.

SIGCOMM'93 - Ithaca, N.Y., USA /9/93

1. Introduction.

With the introduction and widespread deployment of fiber technology, the communications world is rapidly gearing up for multi-point applications. Many examples of such applications can be stated, such as video-conferencing, broadcast TV, or a broadcast classroom. Several of these applications can be modeled as a broadcast from a source node to several destination nodes, and point-to-point flows from the destinations to the source.

An important application is video-conferencing among some set of users across a communication network. One of the problems in video-conferencing is that the amount of bandwidth required to handle the incoming video streams is proportional to the number of participants in the conference. In order to reduce this bandwidth, the incoming video streams can be scaled down and superimposed by a "mixer" and the merged stream broadcast to all the participants. A mixer must be capable of handling the combined bandwidth of all the participants, but each individual participant needs to handle less bandwidth. It would be cheaper to place a small number of mixers inside a communication network than to have a mixer at each participating host.

Another important application is the digital class-room, where a teacher on his workstation may be broadcasting a lecture to a number of students, who are listening to him remotely on their desktop computers, and are able to provide interactive feedback to the teacher.

It is clear that these applications can be modeled as a forward broadcast from a source to several receivers, and a reverse set of flows from the receivers to the source.

There are several other applications which can also be modeled in a similar fashion. One such application is a dis-

tributed digital juke box. A client (the source) of a distributed digital juke-box sends his stream of requests to several servers (destinations), and mixes the responses of the different servers that reply. Another such application is a distributed simulation. The simulation may occur on several workstations (destinations) across a network, while a person (source) monitoring the simulation compares and analyses the streams of data transmitted back to him by the various simulators.

In this paper, we consider the problem of efficiently routing the class of applications described above. Specifically, we consider the problem of computing the optimum distribution tree which supports the forward and reverse bandwidth requirements. This problem is different from the Steiner tree problem, since the bandwidth on the point-to-point flows from destinations to the source have to be taken into account as well.

The problem of finding the optimal distribution-tree for applications like digital TV without bandwidth reservation is equivalent to the Steiner tree problem in graphs [1]. This problem in NP-Complete, but efficient heuristics ([2] [3]) exist to find a path which is not much worse than the optimum. These algorithms have been used to find the optimal path for multicasting in networks ([4], [5]) using both centralized and decentralized schemes. Modifications to the Steiner tree heuristics have been used to develop a routing strategy for video-conferences with bandwidth reservations [6]. It is also shown in [6] that simple Steiner tree routing may be inefficient for reserved bandwidth traffic.

In the next Section, we define the problem more precisely, introduce the terms that we will be using, and also present some basic algorithms that we will be using later in the development of the heuristics to solve the problem. We will solve the problem through a series of simpler problems, and provide a series of heuristics that solve the problem.

2. Problem Statement

The problem considered in this paper, which we call the *minimum cost capacity-constrained broadcast-reply tree* problem is defined as follows:

Consider a graph $G=(V,E)$, where V is the set of its vertices, and E is the set of its edges with a capacity matrix C, and a cost matrix W where $C[i][j]$ gives the capacity available from node i to node j and $W[i][j]$ gives the cost of reserving a unit bandwidth from node i to j. Given a source node $s \in V$, and a set of vertices D {s in not a member of D}, find a tree T rooted at s and spanning all the members of D such that

1. There is a capacity of F available along the path on T from s to any $d \in D$.
2. Each path from a member of D to s has capacity of R. If a node in the tree has k children in D {including itself}, then link between the node and its parent in the tree must have a capacity of kR to accommodate all the reverse paths passing through the node.
3. The aggregate cost of the bandwidth reserved on all the edges is minimized.

As an example, consider the graph in Figure 1. Suppose s is {1} and D is {2, 5}. Suppose $\forall ij W[i,j] = 1$, and the edges have capacity as shown in the figure. If we want to reserve a forward bandwidth F of 5 and a reverse bandwidth R of 6, and select the tree to consist of the edges {(1,3), (3, 2) and (3, 5)}, then we need an bandwidth of 5 on the forward edges {(1,3), (3, 2), (3, 5)}, a bandwidth of 6 on the reverse edges {(5,3) and (2, 3)}, but a bandwidth of 12 on the reverse edge (3,1), because the two reverse paths converge at node 3.

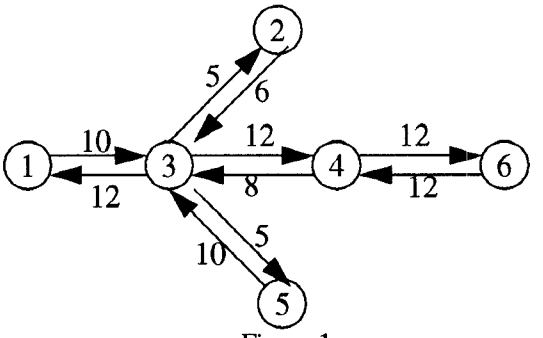

Figure 1.

We define the forward component of the cost of a tree to be the sum of the cost of bandwidth reserved on different edges for the forward transmission of information. It should be noted that the forward component of the cost depends on the number of extra nodes (similar to Steiner points) chosen to build the tree.

We define the reverse component of the cost of a tree to be the aggregate cost of the bandwidth reserved on different edges in order to obtain the reply back from the leaves of a tree to the destination.

The minimum cost capacity constrained broadcast-reply tree problem can be shown to be NP-hard, since it contains a known NP-hard problem (Steiner tree problem) as a special case when $R = 0$, and with all edge capacities $> F$.

Therefore, we must attempt to develop some good heuristics to solve the minimum cost capacity-constrained broadcast-reply tree problem. We approach the problem by solving a series of simpler problems, each progressively

harder than the previous one. As a first step, we solve the problem where the forward component of the cost is a constant. Then we consider the problem in the context of unit-cost graphs. Finally we propose the heuristic to solve the problem in the most general case.

We first define two subproblems and provide efficient solutions for these. These sub-problems will be used in the heuristics that we describe later.

Subproblem 1: The *capacity-constrained shortest path* problem is defined as follows:

Consider a graph $G=(V,E)$, where V is the set of its vertices, and E is the set of its edges with a capacity matrix C, and a cost matrix W where $C[i][j]$ gives the capacity available from node i to node j and $W[i][j]$ gives the cost of reserving a unit bandwidth from node i to j. Given two vertices d and s in V, find a sequence of nodes $n_0n_1n_2...n_k$, such that

1. $n_0=d$.
2. $n_k=s$.
3. $\forall i\,(0 \le i < k)\ ((C[n_i]\ [n_{i+1}]) \ge R)$
4. $\forall i\,(0 \le i < k)\ ((C[n_{i+1}]\ [n_i]) \ge F)$
5. The cost of the path $\sum_{0 \le i < k} R(W[n_i]\ [n_{i+1}]) + F(W[n_{i+1}]\ [n_i])$ is minimized.

Solution: A slight modification to any standard shortest path algorithm, e.g., [7], allows us to obtain the required minimum cost capacity-constrained path. We construct a directed graph $G'=(V,E')$ with the same set of vertices as the original graph G but an edge from node i to node j is only included in E' if $C[i][j] >= R$ and $C[j][i] >= F$. This edge is assigned a cost of $W[i][j]R + W[j][i]F$. We now find the minimum cost path from d to s in G' using the algorithm for shortest paths. The sequence of nodes obtained in this minimum cost path gives us the capacity-constrained minimum cost path in G.

In order to show the correctness of this approach, it is sufficient to show that each path from d to s in G satisfying the capacity constraints corresponds to a path from d to s in G' with the same cost. This correspondence is obvious, and can be proved by induction on the number of hops in the path.

In the rest of the paper we will refer to the capacity-constrained shortest path from d to s with a capacity of F from s to d and a capacity of R from d to s as *SHCAP(d,s,R,F)*. This path will be needed in the heuristics we propose for solving the multi-point routing problem under different circumstances.

Subproblem 2: The *capacity-constrained tree-augmenting path* problem is defined as follows:

Consider a graph $G=(V,E)$, where V is the set of its vertices, and E is the set of its edges with a capacity matrix C, and a cost matrix W where $C[i][j]$ gives the capacity available from node i to node j and $W[i][j]$ gives the cost of reserving a unit bandwidth from node i to j. Given two vertices d and s in V, and a tree T in G, find a sequence of nodes $n_0n_1n_2...n_l$, such that

1. $n_0=d$.
2. $n_l=s$.
3. $\forall i\,(0 \le i < k)\ ((C[n_i]\ [n_{i+1}]) \ge R)$
4. $\forall i\,(0 \le i < k)\ ((C[n_{i+1}]\ [n_i]) \ge F)$
5. $n_0n_1n_2...n_l$ combined with T results in a tree, or if n_j is the first node in the path that is already in T, and there are l descendants of n_j in T, then removing the existing path from n_j to s in T, and adding the path $n_j...n_l$ results in a tree, and furthermore, $\forall i\,(j \le i < k)\ ((C[n_i]\ [n_{i+1}]) \ge kR + R)$

We will refer to this problem as *STCAP(d,s,R,F, T)*, which takes a existing tree or forest T, and finds a path p from d to s such that there is a capacity of F from s to d, and a capacity of R from d to s, and $p \cup T$ is a tree or forest. In other words, we have to choose paths that satisfy the tree constraint.

Solution: There are obviously many solutions to this problem, since we are not imposing a shortest path constraint. However, we propose a solution that can be used in the heuristic. The solution we present is an extension of the *SHCAP* algorithm, where we need to ensure that the paths we obtain preserves the tree property. The approach we take to solve this problem can be best illustrated by Figure 2. Suppose *u->f->s* (denoted by thick lines) is an existing tree, and we need to find the capacity-constrained tree-augmenting path from d to s. We find the capacity-constrained shortest path from d to s. If this results in a tree, we are done. However, in some cases, the shortest path may violate the tree property, as shown in Figure 2. In this case, adding the shortest path *d->f->b->s* to the existing tree does not result in a tree. We then find the second best capacity-constrained shortest path from d to s, which is *d->a->s*. We also find the capacity-constrained shortest cost path from f to s which can accommodate the combined bandwidth requirements of d and u, children of f in the new tree, this path is *f->c->s*. We then choose among the cheaper of two alternative paths, *d->f->s or d->a->s*. If we chose the former, the path from u to s would change to *u->f->c->s*.

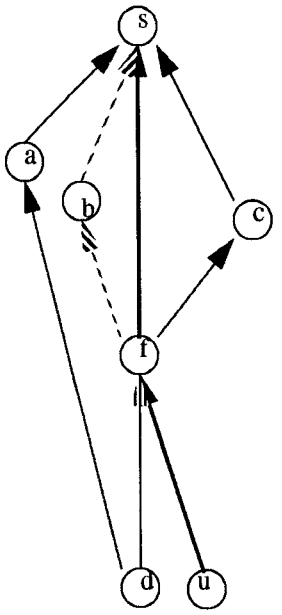

Figure 2.

Since even the new paths found may, in turn, violate the tree property, we need to call *STCAP* recursively in the following fashion:

STCAP(d,s,R,F,T)

1. Find the path p = *SHCAP(d,s,R,F)*.
2. Check if adding path p to tree T results in a tree or forest.
3. If the answer to Step 2 is yes, then p is the desired path.
4. If p U T is not a tree, and p consists of k nodes $n_0n_1n_2...n_k$, then there must be some nodes which are at fault. Node n_i along p is at fault if it is already in T and n_{i+1} is not the parent of n_i in T. Choose f as the node at fault which is closest to d and eliminate the edge going out of f in p from the graph. Find the path *STCAP(d,s,R,F,T)*, in the graph obtained after deleting the link from f.
5. Also examine *STCAP(d,s,R,F,T)* in the graph obtained after deleting the edge coming into f in p. Assign p_1 to be the cheaper of the two paths obtained in the previous and this step.[1]
6. If there are k descendants of n_i, the node at fault, in D, then add the bandwidth of kR to the existing path in T from n_i to s. Remove this path from T to obtain a forest T'. Find path p_2 = *STCAP(n_i,s,kR,F,T')*.
7. Suppose p_3 was the path from n_i to s along nodes in T. Let C_1 be the cost of the path p_1, and let C_2 be the sum of the cost of the fragment of p from d to n_i and k times the difference in costs between p_2 and p_3. If the former is smaller, assign p to be p_1, else assign p to be concatenation of $n_0n_1n_2...n_i$ and p_2. [2]
8. The result of step 7 is the desired path.

Notice that in some cases p_1 or p_2 may not be feasible, and the other one is chosen by default. Also, if n_i is the same as d, only p_2 is feasible.

3. Trees with Constant Forward Components.

As the first simplification, we solve the minimum cost capacity-constrained broadcast-reply tree problem when the cost of the forward component of the tree is fixed. There are several situations when we can simplify the problem to have a constant forward component cost.

One situation is when the source broadcasts to all the nodes in a graph with all edges having unit cost. A source s is specified, and we need to reserve a bandwidth of F in the forward direction, i.e. from s to any other node in the path, and a reverse bandwidth of R from any node to s. If all edges have unit costs and all the nodes in a graph with $|V|$ vertices have to be spanned, then the forward component of the cost is $|V| - 1$, which is a constant. No matter which tree we select, we will be reserving the forward component F on $|V| - 1$ edges, thus minimizing the reverse component of the cost is sufficient to solve the problem.

A similar situation exists when F is equal to 0. Here the forward component of the cost of the tree is a constant, and we need to only minimize the reverse component of the cost. We call this problem the *minimum cost capacity-constrained reply tree* problem.

All that we need to do to find the minimum cost capacity-constrained reply tree in this graph is to find the shortest paths with the available capacity from all the nodes in the graph to the source node. This can be done by means of the algorithm outlined below.

1. The goal is to find a path from d to s which is the second least expensive capacity constrained path.

2. C_1 in this case is the extra cost obtained by adding p_1 to the tree, and C_2 is the extra cost added by rerouting k flows along the tree from p_3 to p_2. Therefore k times the difference between the cost of these two routes must be added to the partial path from d to n_i in order to find the additional cost.

MCRT(s,D)

1. Initialize tree T to consist only of the source node s.
2. For node d in D, which is closest to s but not in T, find the path p = *STCAP'*(d,s,R,F).[3]
3. Reduce the capacity on each edge of the graph by R on path p.
4. Assign T to be p U T.
5. If link u->v with a cost of *W[u][v]* is selected in path, add a dummy link v->u to the network, assign this link a cost of *-W[u][v]* to the node and assign the capacity of this link to be the flow on the link u->v.[4]
6. Repeat Steps 1-4 for all nodes in D.

As an example, consider the graph shown in Figure 3. Assume that all edges have unit capacity in each direction and the edges have costs as shown in the figure. 0 is the source node, and F and R are both unity. Suppose D consists of the nodes {3,4} and we attempt to first find a path from 4 to 0, and then a path from 3 to 0. The shortest path with a capacity of R from 4 to 0 would be {4,1,2,0}. We add dummy links with negative costs from 0->2, 2->1, and 1->4, each with a capacity of R. We reduce the capacity in the reverse direction by R unit along the path. We now need to determine the shortest path from 3 to 0 with a capacity of 1. In the new graph, it is obtained to be {3,2,1, 0}. On adding these links, we determine that link 2<->1 is no longer in the tree, and thus the final tree obtained consists of two paths, {0,1,4} and {0,2,3}.

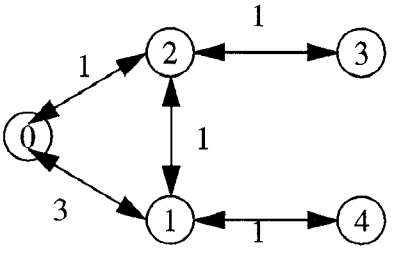

Figure 3.

Notice that this example illustrates the reason why we need to add the reverse path links in the graph. In the choice of the path from 4 to 0, we had chosen a path which blocked any path from 3 to 0. However, adding the dummy links in the graph allowed us to choose a path from 3 to 0, as well as reroute the path from 4 to 0 in a single step.

We now need to prove that the algorithm *MCRT(s,D)* presented above does actually find the minimum cost capacity-constrained reply tree. However, before we actually state the proof, we will assert some relations between the minimum cost capacity-constrained reply tree with n nodes in D, and with $n+1$ nodes in D. We assume that the $n+1^{th}$ element in D is the furthest one from the source node s.

Let us call the minimum cost capacity-constrained reply tree when the cardinality of D is n as *OPTR(n)*.

Proposition 1: *OPTR(n+1)* may differ from *OPTR(n) in* only the following three ways:

1. *OPTR(n)* is a subset of *OPTR(n+1)*, formed by eliminating the path from d_{n+1} to the point in *OPTR(n)* that is closest (least cost) to d_{n+1}.

2. Let u be the closest (least cost) node to d_{n+1} that also is along the path from some other node in D to the source s in *OPTR(n+1)*. Then *OPTR(n)* may have a different and less expensive path from u to s than in *OPTR(n+1)*.

3. Let u be as defined above. Then at most one element x in D which does not have u along its path to s may be routed through u in *OPTR(n)*. Let u_x be the node closest to x in *OPTR(n+1)* which shares the path to s with another node in D -$\{d_{n+1}, x\}$. u_y may take a different less expensive path to s than in *OPTR(n+1)*, and if so, there are no other differences between OPTR(n) and OPTR(n+1). Otherwise, at most one element y in D -$\{d_{n+1}, x\}$ can be routed through u_x. The same assertion recursively applies to u_y the node closest to y *in OPTR(n+1)* which shares the path to s with another node in D - $\{d_{n+1}, x,y\}$.

Proof: Consider the path in *OPTR(n+1)* from d to s. Let u be the node closest to d along this path that is also in *OPTR(n)*. The path from u to s in *OPTR(n)* may or may not have sufficient capacity to accommodate the reply from d. If it does, then the path from u to s should be the same in *OPTR(n+1)*, and the difference between *OPTR(n)* and *OPTR(n+1)* is given by step 1 of Proposition 1. Otherwise, u either has the same path to s in OPTR(n) and OPTR(n+1) or it does not. If it has a separate path, then the path from u

3. *STCAP'* is the same algorithm as *STCAP* except that the forward path capacity on the dummy edges introduced in step 5 is not checked. Whenever a dummy edge is traversed, an existing flow is being rerouted, and the forward flow is already feasible on that link.

4. If the new path added results in a rerouting of the path from one node to the source, (i.e. step 6 in STCAP algorithm is the cheaper alternative) it would be better to remove the link originally going out of the rerouted node. While not required for the correctness of the algorithms, it eliminate negative cycles due to dummy edges introduced, and allows the use of simple shortest-path algorithms to obtain SHCAP.

to s in *OPTR(n+1)* must require more bandwidth than the path from *u* to *s* in *OPTR(n)*, and will be more expensive. In this case, the difference between *OPTR(n)* and *OPTR(n+1)* would be given by step 2 in the proposition. If the path from *u* to s is the same in *OPTR(n)* and *OPTR(n+1)*, then some node in *D* that was being routed through u must have been rerouted along some other path. In this third and most complex case, a restructuring of the tree would take place. However, removing a node from *OPTR(n+1)* allows only one node from the other members in *D* to take the path through *u*. The effect of freeing up of capacity can cause a chain-reaction, rerouting other nodes along the path. However, at most 1 node can be affected in each sub-tree.

Lemma 1: *MCRT(s,D)* finds the minimum cost capacity-constrained reply tree for *s* and *D*.

Proof: We show the optimality of *MCRT(s,D)* by induction on the cardinality of *D*. Obviously, the algorithm works correctly for the case when *D* has a single element. In this case, the optimum tree is just the shortest path between *s* and the element in *D*. Suppose the algorithm works correctly when *D* has a cardinality of *n*, and produces the optimum reverse path tree *OPTR(n)*. We show that the algorithm will also produce *OPTR(n+1)*, the optimum reverse path tree when one more element d_{n+1} is added to *D*.

In order to show that the algorithm finds the tree *OPTR(n+1)*, we show that the shortest path from *d* to *s* in the graph formed by adding links as described in Step 4 of the algorithm where *OPTR(n)* is known will restructure the existing tree to obtain *OPTR(n+1)*. Let us consider the three possible cases one by one.

When *OPTR(n)* is a subset of *OPTR(n+1)*, the shortest path from *d* to *s* would consist of the shortest path from *d* to *u*, and the shortest path from *u* to *s*. If there was another shorter path, then *OPTR(n+1)* would not have been the optimum tree.

In the case 2 outlined in Proposition 1, we would have the shortest path as passing through *u*, but taking a path different than the one taken in *OPTR(n)*. Since the path from *u* to *s* in *OPTR(n)* does not have the capacity to support the additional destination, the shortest path from *d* to *s* would violate the tree property at node *u*. Thus, the algorithm would attempt to reroute the flows through *u*, (step 6 in *STCAP* algorithm) and result in *OPTR(n+1)*, where *u* takes the cheapest path to *s* where the additional required capacity is available.

Case 3 can best be described pictorially. Suppose *OPTR(n+1)* is as described by the solid lines in Figure 4 and *OPTR(n)* is defined by the dotted lines. Because, we

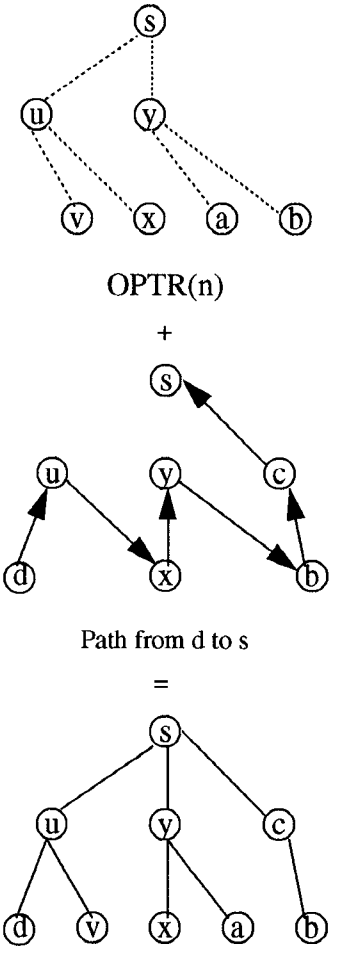

Figure 4.

have introduced dummy edges reversing the flows along all the links in *OPTR(n)*, we are able to take the path from *d* to *s* as shown in the figure. In such a case, we assert that the shortest path from *d* to *s* would consist of a path that restructures *OPTR(n)* into *OPTR(n+1)*. This path would be *d->u->x->y->b->c->s*. The cost of this path is exactly the difference between *OPTR(n+1)* and *OPTR(n)*. If this was not the shortest path, we would have obtained another tree with *n+1* nodes, *OPTR'(n+1)* which would have been less expensive than *OPTR(n+1)* which is impossible. We want to reiterate that the reason we are able to restructure the tree is due to the existence of the dummy links, introduced at step 5 in algorithm MCRT.

Notice that the shortest path found in this case may violate the tree property in cases where we may need to reroute the flows through some nodes in a manner analogous to what we did in case 2 above.

Having proved the above facts, it is easy to prove that the algorithm works correctly for *n+1* nodes in *D* if it works

correctly for n nodes in D. This, plus the base case for $n = 1$, shows that the algorithm does indeed find the minimum cost capacity-constrained reply tree.

4. Heuristic for unit cost graphs

Having solved the minimum cost capacity-constrained reply tree problem in the simplified case where the forward component is constant, we consider the next case where all edges have a unit cost. The heuristic is based on the observation that the forward component of the cost of a tree can be minimized by choosing as few extra nodes in the tree as possible, and the reverse component of the cost can be minimized by means of choosing the minimum cost reverse path tree with capacity R from each element in D to s.

The heuristic is also based on finding the minimum cost tree from the elements in set D to the source node s. However, we try to assign costs to different nodes in the graph so that the algorithm tends to pick up nodes that are already in the tree in preference to new nodes.

The *unit cost graph heuristic* consists of the following steps:

1. Initialize T to consist only of s.
2. Break up each node into two nodes (node x is broken into node xI and node xO). All edges from node x to node y are converted to edges between node xO and node yI, with the same capacity and a cost of carrying a unit bandwidth from x to y.
3. Nodes xI and node xO are connected by an edge with a large capacity, and a cost of F/R if $x \notin D$. Otherwise, the cost of this edge is 0. Note that this means that taking a flow of R along a node in D would cost F less than taking a flow through a node not in D. This is exactly the forward cost associated with picking a new node. Furthermore, a positive cost associated with the new node reduces the chances of picking up a new node, or a path with lots of new nodes.
4. For a node d in D, and the partially built tree T, find $STCAP'(d,s,R,F,T)$. Augment T by the new path obtained in this fashion.
5. Add dummy links with cost of -1 and increase the capacity of such links by R along the selected path. If y is a node that was not in D, and has been selected for the first time in this path, then mark the cost of edge from yI to yO as having zero cost. If y is a node that was not in the original set D, and the flow from yI to yO has been reduced to zero as a result of the new flow, mark the edge as having a cost of F/R. [5]
6. Repeat step 4-6 for all the nodes in D.

We would like to point out that steps 4-6 are the steps involved in finding a minimum cost flow through a graph, (see algorithm FO/S/D in [8], and any flow selected through a node which is already selected through it saves a cost of F, which is the overhead we would be getting if we select a new node. The only additional constraint is because of the tree structure which has to be imposed at every step.

5. Heuristic with Unequal Weights

In a real network, the cost of transmitting information on a link would be dependent on the current load on the link, as well as various other factors. Thus, a more realistic problem would be to find the minimum cost capacity-constrained broadcast-reply tree in a graph with different costs assigned to each link.

Our approach to this solution is the following. We would use the unit-cost graph heuristic, and extend it in the following fashion.

Node n is broken into two sub-nodes nO and nI, and an edge from x to y with capacity c is converted to an edge from node xO to node yI with the same capacity iff there is a capacity of at least F from y to x, and a capacity of at least R from x to y. This edge is given a cost of $FW[y][x] + RW[x][y]$, where W is the cost matrix.

The main structure of the algorithm remains the same as in the case for unit cost graph heuristic. When a flow from node a to node b of magnitude r is added, a edge of capacity r is added from b to a, and its cost is assigned to be the negative of the cost of edge from b to a.

In the case of nodes with different weights, it is not clear as to what weight should be assigned to the internal links of a node. We propose two alternative ways of adding the costs to the internal edges in a node, in the first alternative, *the zero-weight heuristic*, we assign no weight to any internal nodes. It is possible to show theoretical bounds of an algorithm using this approach, as we have done in Section 6. In the other alternative, the *distance to closest node heuristic*, we take the internal cost of an edge to be the cost of the shortest flow that would connect the node to a node already in the destination set or to a partially built tree. This scheme performs slightly better than the first alternative. Performance results for both heuristics have been presented in Section 7.

5. The cost of F/R forces each fresh node to have an extra cost of F associate with it. This is the forward component of the cost for the tree.

The scheme to assign costs to internal edges is slightly different in the unit cost heuristic. This is because choosing an extra node in the tree with unit costs adds an overhead of F irrespective of the path we choose to take to that node. However, there is a dependency of the overhead on the path taken when the weights are not equal. That is the reason for the difference in the assignment of internal costs to the two proposed heuristics.

6. Performance

In this section, we present bounds on the performance of the proposed heuristics.

Lemma 2: The zero-weight heuristic is optimal when we need to span all the nodes of a unit cost graph, or when $F = 0$.

Proof: Follows trivially from Proposition 1.

Lemma 3:When the cost matrix W is symmetric, that is, $W[i][j] = W[j][i]$, then the ratio of the cost of the zero-weight heuristic tree to the cost of the minimum cost capacity constrained broadcast-reply tree is bounded above by $(1 + F/R)/(1 + F/dR)$, where d is the number of nodes in D.

Proof: If the cost matrix is symmetric, then adding the forward component of the cost to the cost of an edge as described in Section 5 does not change the relative weights of the edges. Thus, our heuristic will find the minimum cost capacity-constrained reply tree described in Section 3.

Consider the cost of a tree obtained due to our heuristic. Let $\{e_1, e_2, \ldots e_3\}$ be the edges that have been chosen in the tree. Each edge will have a forward cost w per unit flow, and reverse cost of w per unit flow. If k reverse flows are traversing the edge e_i, then the cost of the edge is $wF+kwR$. The cost of the spanning tree is given by summing up the cost of all the edges in this graph. For edge e_i, let $CF[e_i] = wF$ be the cost of the forward flow on the edge, and let $CR[e_i] = kwR$ be the cost of the reverse flows on the edges. It follows that

$$(\frac{F}{Rd}) CR[e_i] \le CF[e_i] \le (\frac{F}{R}) CR[e_i]. \qquad \text{(EQ 1)}$$

The cost of the spanning tree obtained by the heuristic can be obtained by summing up the cost of the forward and reverse components of all the edges in the tree. Let C_H be the cost of the tree, C_{HR} be the cost of the reverse components of all the edges in the tree, and C_{HF} be the cost of the forward components in all the trees. From equation 1, it follows that

$$(\frac{F}{Rd}) C_{HR} \le C_{HF} < \frac{F}{R} C_{HR}. \qquad \text{(EQ 2)}$$

Since $C_H = C_{HF} + C_{HR}$, it follows that

$$C_H \le (1 + \frac{F}{R}) C_{HR} \qquad \text{(EQ 3)}$$

Now, if C_O is the optimal cost tree, and C_{OF} and C_{OR} are the forward and reverse components of the cost of the optimal tree, it follows by a similar reasoning that

$$(1 + \frac{F}{Rd}) C_{OR} \le C_O \qquad \text{(EQ 4)}$$

Since the heuristic finds the minimum cost reverse flows, the reverse part component of the heuristic is better than the reverse part component of the optimal tree. Thus

$$C_{HR} \le C_{OR} \qquad \text{(EQ 5)}$$

Combining equations 2 and 4, using the knowledge in equation 5, we obtain

$$C_H \le C_O \frac{(1 + F/R)}{(1 + F/(Rd))}. \qquad \text{(EQ 6)}$$

7. Simulation Results

In order to see how the heuristics perform as opposed to the optimal solution, we compare it to exhaustive enumeration of all possible flows satisfying the constraints of the problem. Enumeration was done by selecting all possible subsets of the graph that contain all elements of D, and then finding out the minimum cost flow of capacity R from all the members of D to the source node s. We compared the performance of *zero-weight* and *distance to closest node* heuristics mentioned in Section 5.

We studied graphs with 7 nodes, selecting node 0, and some random 3 other nodes to be connected by a spanning tree. The graphs for the comparison study were generated randomly using the following method. Two nodes were connected by an edge with a probability of 0.6, and an bandwidth chosen on this link as a uniform number between 0 and 12, so that the expected average bandwidth on the link was 6 units. The weight of each link was a random number distributed uniformly between 0 and 1. Finding the optimum trees for nodes containing over 7 nodes took too much simulation time, and therefore was not attempted.

We experimented with three different combinations of F and R. In the first combination F and R were equal, having the value of 1. In the second combination F was 1, while R was 0.1, and in the third combination F was 0.1 while R was

1. The distributed classroom is an application where the forward component is expected to dominate, while video-conferencing with mixing is a case where both components are balanced. In the distributed simulation, or a distributed jukebox, the reverse component is expected to dominate.

Case 1: When F and R were both equal to 1, of the 600 graphs examined, there were no feasible trees found in 50 cases. In 491 of the other cases, the *distance to closest node* heuristic found the optimum tree, i.e., a tree with the same cost as obtained by exhaustive enumeration, and in the other cases, it found a sub-optimum tree. Averaged over the 550 cases where a feasible tree existed, the heuristic produced a tree which cost (on an average) only 0.6% more than the optimum, while considering only those cases where it did not find the optimum tree, the heuristic tree cost (on the average) 5% more than the optimum. The *zero-weight* heuristic found the optimum tree in 408 cases, and the heuristic tree cost about 1.9% more than the optimum over all the 550 graphs. In the 142 cases where the optimum tree was not found, the zero-weight heuristic found a tree that cost about 6.8% more than the optimum.

Case 2: When F was 1.0, and R was 0.1, there was no feasible tree in 23 cases out of 600, and in 466 cases the *distance to closest node* heuristic performed as well as exhaustive enumeration. On an average, the heuristic tree cost 2.8% more than the optimum, while considering only the 111 cases where the heuristic did not find the optimum tree, it found a tree which cost about 13% more than the optimum tree. The *zero-weight* heuristic fared much worse, it found the optimum tree in only 296 cases, and over all the possible graphs found a tree that was 10.2% worse. Over the 281 cases where it did not find the optimum tree, it found a tree which cost 18% more than the optimum.

Case 3: F of 0.1 and R of 1.0 was the best possible situation for the heuristic. In this case, a feasible tree did not exist in 31 cases, and the *distance to closest node* heuristic obtained the optimum tree in 552 cases. In the cases where it did not obtain the optimum tree, it found a tree which on the average cost only 0.8% more than the optimum. Overall, it found a tree which was, on the average, only 0.02% more expensive than the optimum. For the *zero-weight* heuristic, the optimum was found in 531 cases, the overall heuristic tree cost 0.06% more than the optimum, while the 38 cases in which the optimum tree was not found, a tree costing 1% more than the optimum was obtained.

The results of the simulations can be summarized in the following table.

	A	B	C	D	E
D-1	550	491	59	1.006	1.050
Z-1	550	408	142	1.019	1.068
D-2	577	466	111	1.028	1.130
Z-2	577	296	281	1.102	1.180
D-3	569	552	17	1.002	1.008
Z-3	569	531	38	1.006	1.010

D-1: Case, distance-to-closest-node heuristic
Z-1: Case 1, zero-weight heuristic
D-2: Case 2, distance-to-closest-node heuristic
Z-2: Case 2, zero-weight heuristic
D-3: Case 3, distance-to-closest-node heuristic
Z-3: Case 3, zero-weight heuristic.
A: Number of graphs with feasible communication trees
B: Number of trees where heuristic found optimum tree
C: Number of trees where heuristic did not find optimum.
D: Cost of heuristic tree / Cost of optimum tree
E: Same as column C, only for cases in column C.

Excluding the graphs in which no feasible tree existed, the *distance to closest node* heuristic obtained the optimum tree in 89% of the graphs, while the *zero-weight* heuristic obtained the optimum tree in 73% of the examined graphs.

It is clear from the results that the *distance to closest node* heuristic does fairly well in finding a minimum cost tree for routing multi-point connections with bandwidth reservation. Note that the heuristic performs better as the ratio of F to R decreases.

8. Conclusions and further work.

In this paper, we have presented heuristics that can be used to route multi-point connections with bandwidth requirements. We have shown some theoretical bounds on one of the heuristics, and examined their performance by means of simulations. The heuristics seem to perform adequately, and we feel that it can be used in practical networks without a lot of overhead.

There are several issues which remain open in the context of this work. The theoretical bounds that we have obtained are valid only under the constraint that the cost of edges be symmetric. An open problem is to obtain similar bound on the performance of the zero-weight heuristic under more general conditions. Another open problem is to obtain theoretical bounds on the performance of the distance to closest node heuristic.

References

[1] Hakimi, S. "Steiner's Problem in Graphs and Its Implications", Networks, vol. 1, pp 113-133, 1971.

[2] Kou L, Markowsky G, and Berman L, "A fast algorithm for Steiner Trees", Acta Informatica, vol. 15(1981) pp 141-145.

[3] Rayward-Smith V.J and Claire A, "On finding Steiner Vertices", Networks, Vol. 16 (1986), pp 283-294.

[4] Waxman B M, "Routing of Multipoint Connections", IEEE Journal on Selected Areas in Communication, vol. 6 (1988) pp 1617-1622.

[5] Subramanian N and Liu S, "Centralized Multi-point Routing in Wide Area Networks", Symposium on Applied Computing, Kansas City, MO, April 1991.

[6] Jiang X, "Routing Broadband Multicast Streams", Computer Communications, vol 15 (1992), no 1 pp. 45-51.

[7] Floyd R. W., "Algorithm 67 Shortest Path", Communications of the ACM, vol. 5(1962), pp. 345.

[8] Chachra V., Ghare P M and Moore J M, "Applications of Graph Theory Algorithms", North Holland, 1979.