On the Maintenance of Low Cost Multicast Trees with Bandwidth Reservation

Dirceu Cavendish[*], Aiguo Fei, Mario Gerla [†]

Department of Computer Science
University of California,Los Angeles CA 90024
{dirceu,afei,gerla}@cs.ucla.edu

Raphael Rom

Faculty of Electrical Engineering
Israel Institute of Technology Haifa 32000 Israel
rom@ee.technion.ac.il

Abstract

Emerging multimedia network technologies are bringing new challenges to protocols design, due to new QoS requirements. These requirements, when combined with traditional network functions, such as routing, render the design of such protocols more complex than ever before. Resource reservation is one of the novel features brought into the picture by multimedia applications. Another is the many-to-many nature of multicast connections. We have recently proposed a routing algorithm to build low cost multicast trees. In this paper, we study the maintenance of these multicast trees, in the face of multicast membership changes over time (join/leave). One of the key issues is the tradeoff between the incremental upgrade at each join/leave request and the recomputation of the entire tree at each step. Simulation results are used to illustrate these trade offs. A signalling mechanism for the implementation of the incremental tree updating is also included.

Technical Subject Areas: Broadband Switching Systems and ATM Networks, Multimedia and Broadband Services.

1 Introduction

One of the new challenges brought on by multimedia network technologies, such as ATM and the next generation Internet, is routing with resource reservations. By incorporating resource reservation into a routing algorithm, the chance that the resources required for carrying a given connection be available at connection setup time is greatly increased. Therefore, sucessful response to connection request depends heavily on the routing algorithm used.

In a QOS environment, routing algorithms must provide routes which conform with users end-to-end requirements while minimizing the cost of a connection. A connection cost is defined as the sum of the costs of individual links participating in the connection. Link costs can be assigned in various ways. Generally, they should reflect the desirability of using a given link as part of a connection path.

[*]Corresponding author - Fax # : (310) 825-7578

[†]This work was partially supported by CNPq under Grant 201597/93-4(NV)

Oftentimes, hop count is the metric used, which amounts to assigning a uniform cost of one to all network links.

QOS requirements translate into path constraints for a routing algorithm, and impact its computation complexity. Complexity is a major concern in network scalability since the routing algorithm must sometimes run in real time, specially in case of multicast connections. A given application may not afford to wait too long for a routing algorithm to compute a QOS suitable path, before a connection setup signalling can be initiated. Moreover, if route computation takes too long, topology information in which this computation is performed is likely to become stalled, leading to a higher probability of having a connection setup failure.

The first, and perhaps most important network resource is bandwidth. We assume a link state routing strategy, as in PNNI ATM technology or in Internet OSPF used as a Interior Gateway routing Protocol (IGP), where each route computation site knows about resources along the links of the entire network. The problem of routing unicast (one-to-one) unidirectional connections with bandwidth reservation was shown to be solved by a Dijkstra type algorithm [Wang95] for link state routing. Multicast traffic is typically carried through one (Core Based Tree) or more (Distance Vector Multicast Routing Protocol) multicast trees [Deering96], for efficiency of network resources.

In [Cavendish98], we have addressed the problem of constructing multicast trees, so as to provide not only connectivity among members of a multicast connection, but also bandwidth necessary to carry the traffic injected by multicast members. In this paper, we study mechanisms to maintain these multicast trees, when multicast membership changes due to the join/leave activity within the group.

The paper is organized as follows. Section 2 describes related work. Section 3 reviews the Bandwidth Constrained Steiner Tree (BCST) and the Bidirect-Dijkstra algorithms, introduced in [Cavendish98]. Section 4 describes the join/leave procedures for maintenance of a multicast group, including the signalling mechanism. In section 5, we show simulation experiments on join/leave routing operations. We conclude the paper in the last section.

2 Related Work

As mentioned earlier, the most important network resource to be allocated by a QOS network is bandwidth. Bandwidth constrained unicast (pt-pt) routing has been shown to be of polynomial complexity [Wang95], and thus scalable with the size of the network. The idea is to disregard any link which does not have enough available bandwidth to carry the bandwidth required by the single source in the computation of a minimum cost path. Using the same strategy for point-to-multipoint (pt-mtp) connections, a minimum cost spanning tree can be constructed with the same polynomial complexity. However, for truly multicast connections (mtp-mtp), we are indeed facing a NP-Complete problem.

Although the problem of constructing Steiner trees is hardly new (e.g. [Smith83]), its study has been intensified in the last ten years due to its direct applicability to multimedia networks [Frank85, Makki95]. Moreover, multimedia connections are likely to add some constraints to feasible trees, such as bandwidth and delay. Research work on the construction of Steiner trees with such constraints is less abundant in the literature. Among these, [Jiang91] surveys algorithms which are able to compute a set of Steiner trees of whose total bandwidth capacity matches the needs of multicast members. The work is limited, however, in a sense that it assumes that every multicast member has the same bandwidth needs, and does not provide performance guarantees of any kind. [Ravi95] establishes hard results, and proposes algorithms for the construction of constrained Steiner tree problems, where the constraints are the *diameter* of the tree and the *maximum degree* of a vertex. It also provides worst case guarantees for algorithms performance. Our previous contribution [Cavendish98] has extended that work in providing algorithms for constructing low cost Steiner trees with *bandwidth* constraints and worst case guarantees, for many-to-many casting applications.

Regarding the maintenance of multicast trees, there is a significant amount of work for the unconstrained routing problem [Waxman88, Westbrook93, Bauer96]. However, the maintainance of constrained multicast trees is a recent research topic. [Bauer96] has proposed that a member join/drop procedure join/leave affect a small sub-domain of the multicast tree only. This approach, although attractive, is not applicable to a bandwidth constrained multicast tree problem, due to the fact that a new member might require additional bandwidth along the entire tree.

3 Bandwidth Constrained Steiner Tree Construction

We model our network as a directed graph $G(V, E)$ where V is the set of nodes and E the set of directed *edges*. We refer to the *link* (i, j) as the pair of edges (i, j) and (j, i). We associate two values with every edge $(i, j) \in V$: a positive weight $w(i, j)$ and a nonnegative (bandwidth) capacity $b(i, j)$. Let $S \subseteq V$, $D \subseteq V$ be source and destination subsets of V, such that $\forall i \in S$, we associate $B_i \geq 0$ as the bandwidth needed by source i.

Define a multipoint connectivity structure (MCS) $\sigma(V', E')$ as a connected subgraph of $G(V, E)$ containing at least the nodes $S \cup D$ and having at least one path from each node $s \in S$ to every node $d \in D$. The bandwidth and weights associated with an edge in σ are those associated with the original edges of G.

Let $S(i, j) \subseteq S$ be the subset of sources contributing flow to edge (i, j). Then:

Definition 1 *An edge (i, j) is said to be underloaded if:*

$$b(i,j) \geq \sum_{p \in S(i,j)} B_p \tag{1}$$

otherwise, the link is said to be overloaded.

A link is underloaded if both edges comprising it are underloaded. An MCS is called *feasible* if all its edges are underloaded. The *weight* or *cost* of an MCS is the sum the weights of its edges.

3.1 The bidirectional connection

The simplest multipoint-to-multipoint (mtp-mtp) connection that one may conceive is a bidirectional unicast connection. In this case, $S = D = \{s, d\}$. Thus, we require that the MCS be a single bidirectional path connecting (s, d). To proceed, we need the following simple definitions:

Definition 2 *The length $d_p^c(s,d)$ of a path $p(s,d)$ under cost function c is defined as:*

$$d_p^c(s,d) = \begin{cases} \sum_{(i,j)\in p(s,d)} c(i,j) \\ \quad \textit{if } \forall (i,j) \in p(s,d), \qquad b(i,j) \geq B_s \textit{ and } b(j,i) \geq B_d \\ \infty \\ \quad \textit{otherwise} \end{cases}$$

i.e. the path length is the sum of its link lengths if its edge components are all underloaded. Otherwise, the path length is assumed to be ∞.

Definition 3 *The shortest path between (s,d) with respect to the cost function c is:*

$$\delta^c(s,d) = \min_{p \in P} d_p^c(s,d)$$

where P is the set of all $p(s,d)$ paths.

In [Cavendish98], we propose a Dijkstra type of algorithm to solve the single bidirectional path min-cost problem. The algorithm is called Bidirect Dijkstra (BD-Dijkstra), and is similar to the original Dijkstra algorithm. The difference is that only links which have sufficient bandwidth in both directions are considered in a relaxation step. BD-Dijkstra's pseudo-code is presented below. Its proof of correctness and complexity analysis can be found in [Cavendish97].

```
**BD-DIJKSTRA G(V,E) algorithm**
BD-Initialize(G,s);
$S \leftarrow \{\}$, $Q \leftarrow V[G]$;
while $Q \neq 0$ do
        u= BD-Extract-Min($Q, d$);
        $S \leftarrow S \cup \{u\}$;
        For each vertex $v \in Adj[u]$ do
            BD-Relax($B_s, B_d, u, v, b(u,v), b(v,u), c(u,v)$);

**BD-Initialize(G,s)**
For each vertex $v \in V[G]$
    Do $d[v] \leftarrow \infty$; $\pi[v] = NIL$ ;
$d[s] \leftarrow 0$;

**BD-Extract-Min(Q,d)**
$d_{min} \leftarrow \infty$; $u \leftarrow NAN$;
For $i \in Q$ do
    If $(d[i] < d_{min})$
        { $d_{min} \leftarrow d[i]$; $u \leftarrow i$;}
RETURN $u$;

**BD-Relax($B_s, B_d, u, v, b(u,v), b(v,u), c(u,v)$)**
If $((B_s < b(u,v))$and$(B_d < b(v,u)))$
    $\{w(u,v) = c(u,v);\}$
else
    $\{w(u,v) = \infty;\}$
If $(d[v] > d[u] + w(u,v))$
    $d[v] \leftarrow d[u] + w(u,v)$;
    $\pi[v] \leftarrow u$;
```

3.2 Multicast Tree Problems

For larger S and D sets, we are interested on a particular Multipoint Connectivity Structure (MCS), called multicast tree, which we now define:

Definition 4 *A multicast tree (mtree) $MT(E', V')$ is an* **acyclic** *MCS $\sigma(E', V'), E' \subseteq E, V' \subseteq V$ providing connectivity to every $m \in S \cup D$.*

One can easily see that an mtree is a Direct Acyclic Graph (DAG). Mtrees inherit the same feasibility definition as for any MCS. Regarding the construction of feasible mtrees, our problem of interest is:

Problem 1 *Construct a feasible mtree of minimum cost.*

A feasible mtree is depicted in Figure 1. The link label (Bd, Bu) represents the bandwidth allocated downstream, Bd, and upstream, Bu, respectively.

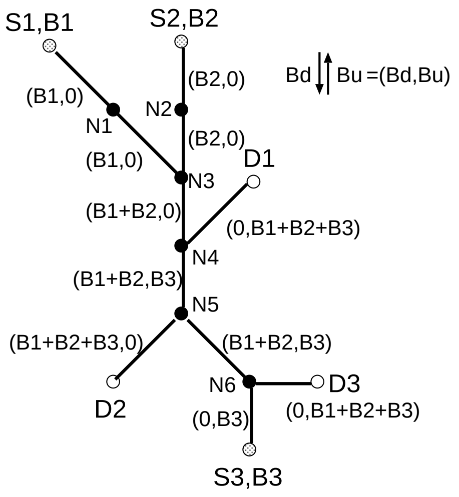


Figure 1: Feasible mtree. Only link bandwidths are shown

3.3 A cluster algorithm for low cost mtrees

The basic idea is to design an algorithm that works in phases [Ravi95]. In each phase, the algorithm connects a given number n of clusters together which were previously disconnected. The algorithm will stop after $\log_n K$ phases, for a multicast group of size K. The BCST-algorithm pseudo-code follows:

BCST-algorithms

1 Initialize the set of clusters in phase 1, Φ_1, to contain $\mid S \cup D \mid$ singleton sets, one for each member of the multicast connection. Define the single vertex as the "center" or head of its respective cluster. $i = 1$.

2 Repeat until there remains a single cluster in Φ_i:

   2.1 Construct a complete graph $G_i(E_i, V_i)$ as follows: $v \in V_i$ iff v is a center of a cluster in Φ_i. Between every pair of vertices $(x, y), x, v \in V_i$, include an edge in E_i of cost defined as the minimum path cost, computed by BD-Dijkstra, between these two vertices in which the bidirectional bandwidth constraint defined by the clusters bandwidths is required. The clusters bandwidth is defined to be the bandwidth required by the cluster head.

   2.2 Find a minimum cost matching of largest cardinality in G_i [Goemans93].

   2.3 For each edge (x, y) selected in the matching, merge their respective clusters C_x, C_y, by adding the path in the original graph corresponding to that edge to form cluster C_{xy}. The vertex (edge) set of the newly formed cluster is defined to be the union of the vertices (edges) of its cluster components, plus the vertices (edges) of the connecting path.

   2.4 Elect randomly a new cluster head for each newly formed cluster. $i = i + 1$.

BCST-Algorithm maintains a set of clusters Φ_i in each phase i. Each cluster is formed by multicast members which are already connected among themselves. An analysis of the BCST-algorithm, together with hard bounds on cost and bandwidth overload, is presented in [Cavendish98]. It suffices to say that BCST-algorithm is able to compute low cost multicast trees with $O(\log K |V^2| \log |V|)$ time complexity, where K is the size of the multicast group.

A typical run of the algorithm, for $|V| = n = 16$, is depicted in Figure 2. In the upper left square (a), a randomly generated instance of a 16 node topology is depicted. The next four squares (b,c,d,e) show a sequence of four phases that the algorithm goes through to build the multicast tree. The smallest dots are network vertices which are neither sources nor destinations. Dots surrounded by a gray circle are cluster heads in a given phase. Links in these middle squares are sets of shortest paths between cluster heads which define a low cost matching. The lower right square (f) depicts the final multicast tree.

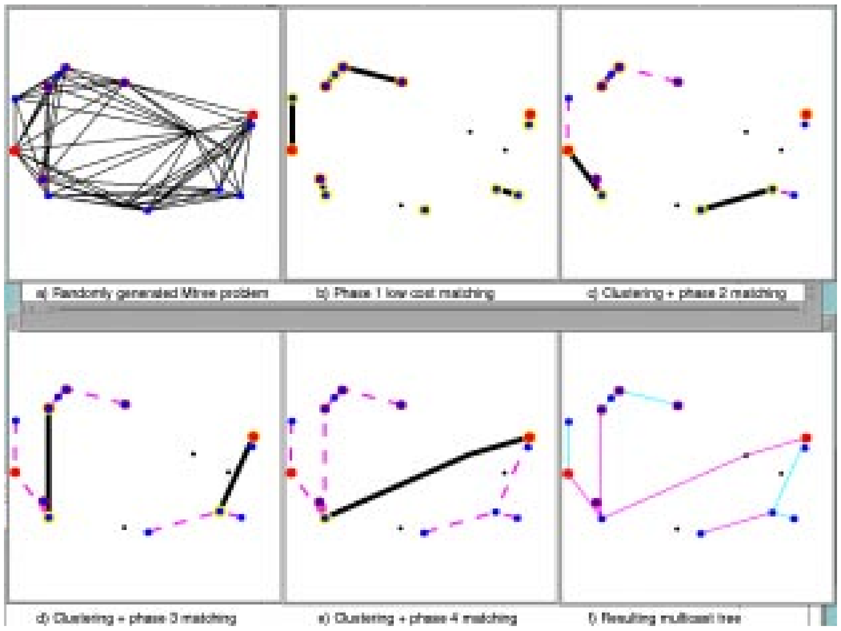

a) Randomly generated Mtree problem
b) Phase 1 low cost matching
c) Clustering + phase 2 matching
d) Clustering + phase 3 matching
e) Clustering + phase 4 matching
f) Resulting multicast tree

Figure 2: Four phases of a BCST-algorithm

4 Multicast Tree Maintenance

Membership of a multicast group is expected to vary greatly during the lifetime of the connection, for most applications. The join/drop operations have the effect of changing the original tree topology. Therefore, we must make sure that, at all times, the multicast tree keeps its suitability to transport the multicast traffic regardless its changes. Mtree suitability translates into both compliance with the constraints and low cost.

The join of a new multicast member is a more delicate operation than the drop. Most constrained multicast trees remain feasible if some tree edges are pruned off, regardless of the type of constrain. Therefore, we start with the drop operation.

4.1 Drop of a multicast member

Starting with a feasible mtree, it is obvious that the drop of a multicast member does not affect the tree feasibility. Therefore, we need to concentrate our attention on the tree cost only. The drop operation should result in the pruning off of the no longer necessary links of the original tree. These links are the ones to be traversed by following the tree from the party to be dropped towards the other members until a fork point is hit (a vertex with degree greater than 2), if this party is a leaf in the tree. If the party is an internal tree node, no link pruning results from the drop operation. Bandwidth release must be performed at appropriate mtree links. Figure 3 illustrates the drop of a sender $S3$ with bandwidth $B3$.

DropMember(m,Mtree)

1 Prune off Mtree by deleting links from m until the first mtree fork is encountered.

2 Update other Mtree links by subtracting the bandwidth used for member m in/out flows.

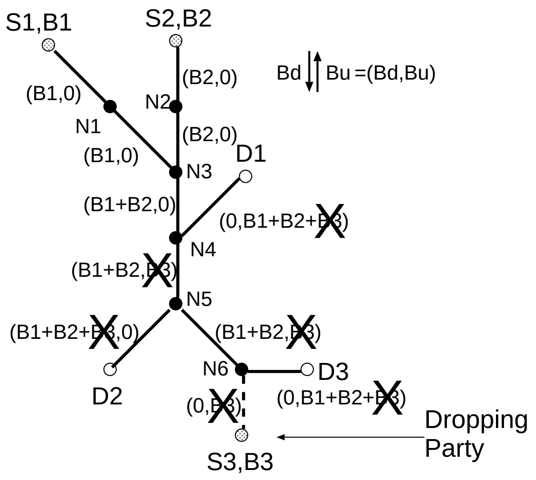


Figure 3: Drop operation from a bandwidth constrained mtree

We first observe that:

Theorem 1 *Upon a drop of a multicast member from the minimum cost mtree, the mtree resulting by pruning off unnecessary links which connect the droping member to the rest of the tree is not necessarily a minimum cost tree for the new multicast group.*

Proof of Theorem 1 *By counter-example. Consider the example shown in Figure 4. A multicast group of three members, $\{SD1, SD2, SD3\}$, both sources and destinations, with uniform bandwidth requirement of 1 is depicted. The network has only four links, three of which are part of the minimum cost mtree, $Mtree = \{(a,b),(b,c),(b,d)\}$, with uniform bandwidth $(1,2)$. The extra link, (a,c), has bandwidth $(1,2)$, thus insuficient to be part of the mtree. For sake of simplicity, link costs are assumed proportional to the euclidean distances. Upon the departure of $SD2$ from the multicast group, the mtree resulting from the pruning off of link (b,d), $Mtree1 = \{(a,b),(b,c)\}$, is clearly more expensive than the single link mtree $Mtree2 = \{(a,c)\}$, which now has become feasible.*

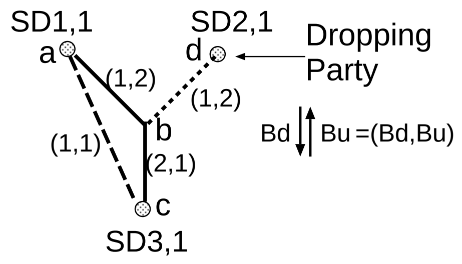


Figure 4: Drop operation from a minimum cost bandwidth constrained mtree

Theorem 1 states that sub-optimal mtrees may result from a drop operation. Therefore, a decision has to be made if/when the recostruction of the mtree is necessary. Since the construction of a minimum cost bandwidth constrained mtree is an NP-Complete problem [Cavendish97], it is also possible that the computation of a new tree by BCST-algorithm results in a more expensive tree than the existing one, although this is highly unlikely in non-degenerate instances of the problem. However, the computation of a new tree by BCST-algorithm may result in an unfeasible tree. We defer the discussion of this topic to the simulation section.

4.2 Join of a multicast member

Although the drop operation has similar tradeoffs for any type of mtree constraints, this is not the case for a join operation. The join operation to a bandwidth constrained mtree has a major difference from, for instance, the join operation to a delay constrained mtree. In the latter, the violation of the constraint affects the communication between the new member and the other members of the group only. It does not affect the communications among old members. In the former, an overload on a link caused by a badly joined new member may potentially cause damage to the communications among all multicast members. Of course we can always reject the join request, in case an overload link(s) results. However, we do not have the option of "living" with the overload, as in the delay constrained case. Figure 5 illustrates the point.

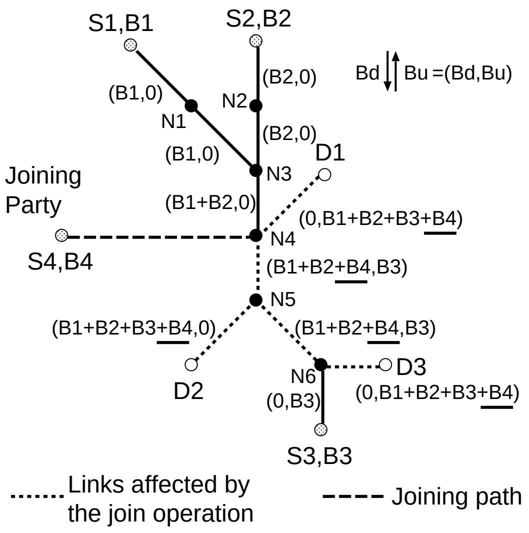


Figure 5: Join of a new member to a bandwidth sensitive multicast tree.

Unlike the drop operation, as long as a joining node has bandwidth requirements of its own, we need to care about tree feasibility in addition to cost. Moreover, the join of a new member includes the definition of a suitable (bandwidth) path to connect the member with the original tree, which involves the definition of an entry point to the tree. Potentially, any mtree node may become an entry point. The path (and corresponding entry point) should be such that not only the necessary bandwidth be available at the path itself, but also the connecting point on the tree be such that the mtree remains feasible. Two cases are relevant, regarding tree feasibility: the new member is a destination, i.e. it does not have any bandwidth requirement; the new member is also a source, in which case additional bandwidth to be injected into the mtree is a concern.

For a new destination member, the path connecting the member with the tree must have bandwidth requirement of $B = \sum_{p \in S} B_p$ from the tree to the new member, and zero in the opposite direction. Regarding the entry point, members which are destinations are the preferred nodes, since all multicast traffic converges to these points already. Hence, bandwidth constraints on the mtree links remain unaltered. Figure 6 illustrates the join operation of a destination member.

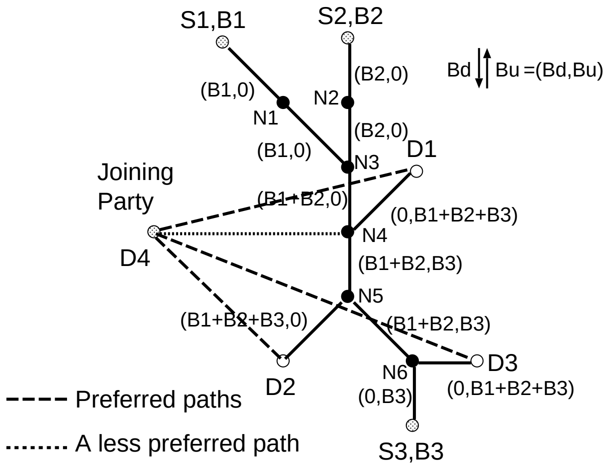


Figure 6: The join of a new destination multicast member

The destination join procedure is, then:

JoinDestination(d,Mtree)

```
1 Compute bidirectional shortest path to every destination member, using $BD-Dijkstra$ with bandwidth constraints of $(0, \sum_{p \in S} B_p)$.
2 Choose the shortest path among all computed paths.
3 If no feasible path is available, compute shortest path to all remaining nodes, with same bandwidth constraints as before.
    3.1 In increasing path length, check mtree feasibility.
    3.2 If no feasible path is available, reject the join request.
4 Update Mtree links' bandwidth accordingly.
```

Notice that the mtree feasibility check is necessary only if the entry point is not a destination member. Since BD-Dijkstra has $O(|V| \log |V|)$ complexity, and feasibility check has $O(|V|)$ (a tree has at most $|V| - 1$ links), **JoinDestination** has worst case time of $O(M|V| \log |V|)$, where $M = |S \cup D|$.

For a new source member s, the path connecting the member with the tree must have bandwidth requirement of $B = \sum_{p \in S} B_p$ from the tree to the new member, and B_s in the opposite direction. Regarding the entry point, members which are destinations are preferred over other nodes only if the new member is also a destination, for the same reason as before. Otherwise, any mtree node is equally likely to be the entry point. Figure 7 illustrates the join operation of a source member.

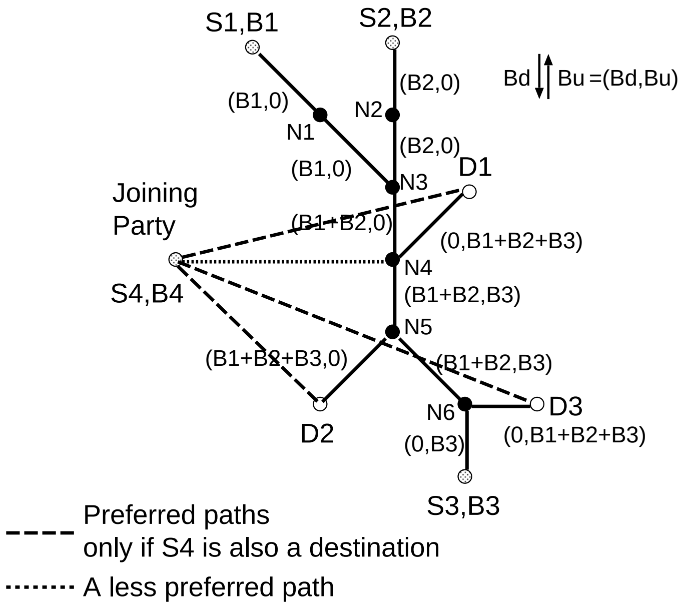


Figure 7: The join of a new source multicast member

The source join procedure is, then:

```
JoinSource(s,Mtree)
 1 If source is also a destination: Compute bidirectional shortest path to every destination member,
   using $BD - Dijkstra$ with bandwidth constraints of $(B_s, \sum_{p \in S} B_p)$.
     1.1 In order of increasing path length, check mtree feasibility.
 2 If no feasible path is available, compute shortest path to all remaining nodes, with same bandwidth
   constraints as before.
     2.1 In order of increasing path length, check mtree feasibility.
     2.2 If no feasible path is available, reject the join request.
 4 Update Mtree links' bandwidth accordingly.
```

For the same arguments as before, **JoinSource** has worst case time of $O(M|V|\log|V|)$, where $M = |S \cup D|$.

Regarding cost of the new mtree, Theorem 2 is the counterpart of Theorem 1:

Theorem 2 *Upon a join of a multicast member from the minimum cost mtree, the mtree resulting by accrueing the links which connect the joining member to the rest of the tree is not necessarily a minimum cost tree for the new multicast group.*

Proof of Theorem 2 *By contradiction or counter-example. We choose the former. If a new minimum cost n-member mtree could be obtained from the join of a member to the (n-1)-member minimum cost mtree, then any minimum cost mtree could be built from scratch by first connecting two members using BD-Dijkstra, and then adding one-by-one the remaining group members, all in polynomial time. But [Cavendish97] has proven that the building of a bandwidth constrained minimum cost Steiner Tree is an NP-Complete problem.*

In [Bauer96], it has been proposed that a member join/drop procedure affect only a small sub-domain of the multicast tree. This approach, although attractive, is not applicable to our problem. This is because bandwidth must be reserved across the entire mtree, in order to support a new source member. Since [Bauer96] considers unconstrained mtrees only, any approximation scheme would impact on an extra increase in the cost of the mtree. In our case, rearranging parts of the tree could be disastrous in terms of bandwidth reservation across the links outside of the rearranged domain.

4.3 Signalling and Implementation Issues

In this subsection, we describe a signalling mechanism to carry out the changes in bandwidth reservation triggered by join/leave requests. We assume that packet interleaving from various sources can be appropriatly handled at destination sites. For completeness, we first describe the signalling involved in an mtree setup.

For sake of example, assume that the mtree shown in Fig. 1 is to be built. Basically, any multicast member site can compute the tree, as long as it knows the initial multicast membership. Let $S1$ be the site in which the mtree was computed. Following the computation of the mtree by BCST-algorithm, $S1$ builds a SETUP message that must "traverse" the tree so as to reserve the bandwidth necessary in the various links. We propose that this traversal be in a circuit or loop fashion, so as to facilitate the handling of pointers. A similar scheme is proposed in [OPENET95]. The SETUP message is a source routed message, containing a sequence of pairs - (node ID, bandwidth) - to be traversed/reserved. In this example, the complete message would be:

SETUP = {(N1,B1), (N3,B1), (N4,B1+B2), (N5,B1+B2), (D2,B1+B2+B3), (N5,0), (N6,B1+B2),(S3,0), (N6,B3), (D3,B1+B2+B3), (N6,0), (N5,B3), (N4,B3), (D1,B1+B2+B3), (N4,0), (N3,0), (N2,0), (S2,0), (N2,B2), (N3,B2), (N1,0), (S1,0)}.

Thus, the first entry, $(N1, B1)$, allocates bandwidth $B1$ on the link $(S1, N1)$, and so on. In case the reservation SETUP fails at any given point, a RELEASE message is generated from the point of failure in the reverse direction, in order to deallocate the bandwidth already reserved. For instance, if link $(N3, N4)$ has less than $B1 + B2$ available bandwidth, it will generate a reverse message as: $RELEASE = \{(N3, B1 + B2), (N1, B1), (S1, B1)\}$. We assume there is a bit indicating in which direction the bandwidth is affected by the packet, either in the direction it travels (FWD), or in the opposite direction (BWD). Thus, a RELEASE message followed by a reservation failure is a BWD type of message, since it affects bandwidth reservation in the **opposite** direction of its traversal. A setup message, on the other hand, is a FWD message, since it affects bandwidth reservation in its traversal direction.

In a join/drop operation, the multicast member wishing to join/drop is assumed to have not only network topology information but also have information about the mtree topology. There are two ways for a new member to learn about mtree topology: by broadcasting a packet across the network, asking for such information - multicast members will reply with the mtree topology; or by asking directly a "well known" multicast member about the mtree topology. The first method is wasteful, since multiple members might reply simultaneously, wasting network resources. We believe the second one to be more practical, since oftentimes a new member knows a multicast group by the leading member.

For a drop operation, the dropping party builds a (FWD) RELEASE message similar to the SETUP message. The bandwidth reported in such message, as in the setup failure case, is the amount of bandwidth to be deallocated at each hop of the mtree. For instance, in Fig. 3 the message generated by $S3$ would be:

RELEASE(FWD) = {(N6,B3),(D3,B3), (N6,0), (N5,B3), (N4,B3), (D1,B3), (N4,0), (N3,0), (N2,0), (S2,0), (N2,0), (N3,0), (N1,0), (S1,0), (N1,0), (N3,0), (N4,0), (N5,0), (D2,B3), (N5,0), (N6,0), (S3,0)}.

Notice that shortcuts are possible, if the member executing the operation is smart enough to compute such short traversal paths. For instance,

RELEASE(FWD) = {(N6,B3),(D3,B3), (N6,0), (N5,B3), (N4,B3), (D1,B3), (N4,0), (N5,0), (D2,B3), (N5,0), (N6,0), (S3,0)}

could be used by $S3$ to release its bandwidth [1].

For a join operation, once the joining path is computed, an ADD-PARTY message is constructed, similar to the SETUP message, requesting only the amount of extra bandwidth needed by the new member. In the example of Fig. 5, assume, for simplicity, that the joining path is a one link path. $S4$, then, will issue the (shortcut) message:

ADD-PARTY = {(N4,B4), (N5,B4), (D2,B4), (N5,0), (N6,B4), (S3,0), (N6,0), (D3,B4), (N6,0), (N5,0), (N4,0), (D1,B4), (N4,0), (S4,0)}.

For a join of a destination party into the multicast group, as in Fig. 6, in most cases a very short message can be used, if the new member joins the group at a destination site. Assuming a one link joining path $(D4, D2)$, the ADD-PARTY message would be: $ADD-PARTY = \{(D2, 0), (D4, B1+B2+B3)\}$. Notice that an ADD-PARTY message may fail to reserve bandwidth at a particular mtree link. In this case, similarly to the SETUP case, a reverse RELEASE message rolls back the reservations to where they were before the ADD-PARTY was issued.

5 Simulation Study

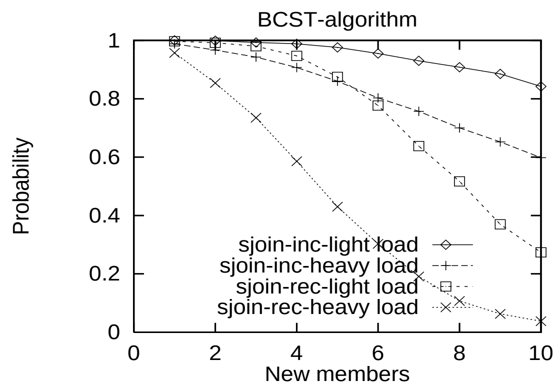
As in [Cavendish98], we have adopted the Euclidean distance between any two vertices i, j as the cost of link (i, j). Each problem instance is generated as follows. $|V| = n$ vertices are randomly placed on a grid. Among $n(n-1)/2$ links of the complete graph formed by the pairwise distances, a connectivity factor C_f is used to prune off the $(1 - C_f) * n(n-1)$ longest links. Edge bandwidths are independently chosen from a $U(B_{min}, B_{max})$ uniform distribution. A vertex is designated as a source with probability p_s and a destination with probability p_d (hence the number of sources and destinations has a binomial distribution). Source traffic rates (i.e, required bandwidths) are chosen from a $U(0, B_s)$, where B_s is computed by:

[1]Notice that, if a dropping party is both source and destination, the traversal of the whole tree might be necessary, in order to release bandwidth used by the member flow in (source) and out (sink) of the network

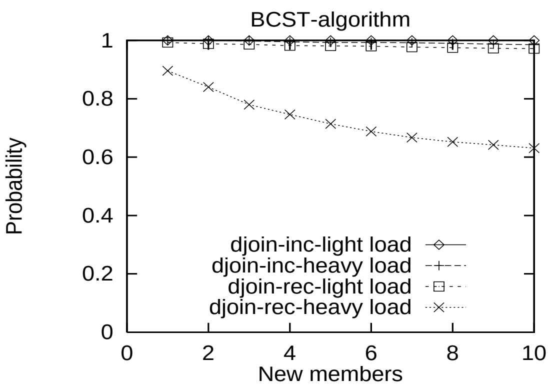
$$B_s = \frac{B_{min} + B_{max}}{n * p_s} \times h_f \tag{2}$$

Equation 2 is such that h_f gives a percentage of the average link bandwidth per source as compared with the average source bandwidth. In other words, $B_T = n \times B_s/2$ being the total average bandwidth requested by the multicast connection, and $B_e = (B_{min} + B_{max})/2$ the average edge bandwidth, $h_f = B_T/B_e$ is a hardness factor for a given problem. Heavy load and light load situations can be simulated by varying h_f within the $(0, 1)$ interval.

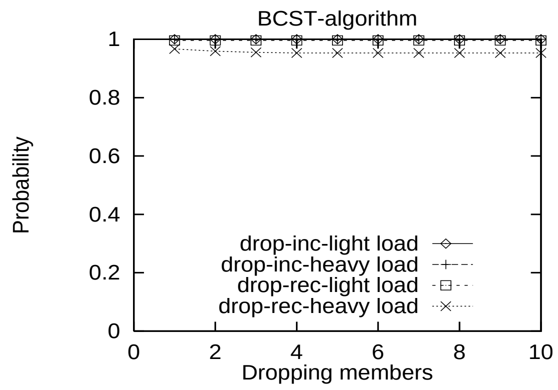
For each multicast routing problem generated, we use BCST-algorithm to compute the initial mtree. Once the tree is found, two set of experiments are conducted: a series of drop operations upon the mtree; a series of join operations upon the mtree. In order to evaluate how the mtree cost is affected along membership activity, for each join/drop operation, we compute the cost of the resulting tree, and also recompute the entire tree, as in a new problem. In the set of figures that follows, we report the ratio of these two quantities. Each point represents an average of 1000 randomly generated instances of size $|V| = 32$ vertices. By varying h_f, we are able to simulate light and heavy network load situations, as mentioned before. We have also experimented with various graph connectivity degrees. We present the most representative results only (heavy loaded networks with fixed graph connectivity), for sake of conciseness.



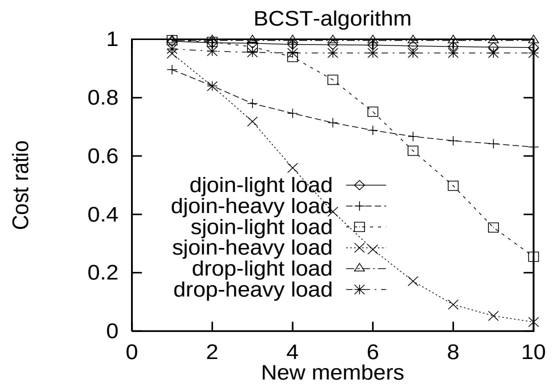
a) Source join-success



b) Destination join-success



c) Member drop-success



d) Member-cost ratio

Figure 8: **Incremental Multicast cost/performance tradeoffs**

In Figure 8, the main experimental results are reported. In each plot, incremental (inc) and recomputation (rec) strategies are evaluated, for both light and heavy load cases. Figures 8 a) and b) show the success ratio (i.e. fraction of accepted joins/leaves) when updating the mtree incrementally as well as computing an entire new tree. For a source join (Fig. 8 a), incremental update has always better chance of success, regardless of network load (heavy/light). For a destination join (Fig. 8 b), the only case in which recomputation of the mtree is to be avoided is in heavy load. For drop operations (Fig. 8 c), we notice that incremental drop always succeed. The recomputation of the mtree should be avoided in the drop operation. The fact that the reoptimization of the tree at each step does not lead to a better success ratio is counter -intuitive. It is not surprising, however, since it confirms trends noted in other problems such as routing of connections in a circuit switched network and bin packing problems.

Another key performance measure is, of course, cost. Figure 8 d) reports the mtree cost increase when sources and destinations incrementally join the multicast group. The interesting curves are the ones which depart from the 1.0 ratio, because this means an mtree recomputation pays off in terms of cost. These are destination join, under heavy load, and source join, irrespective of network load condition. An inspection of the previous figures show that these are exactly the cases in which the success ratio of recomputations declines, as compared with incremental mtree computation. Therefore, there is a clear trade off in cost gain versus mtree update success probability. A routing strategy driven by minimizing the cost of the mtree will not only disrupt more severely the multicast connection by completely rebuilding the mtree, but it will also fail more often in finding a feasible mtree, thus causing extra blocking. Of course, a strategy of trying both approaches in parallel could be exercised, if computation time is not an issue.

6 Conclusion and Future Work

In this paper, we have analysed the problem of maintaining a low cost feasible Steiner Tree which is constrained by bandwidth requirements of its connecting vertices upon incremental join/drop of new vertices. This problem finds applications in multimedia networks, in which source/destination sites with well defined bandwidth requirements, dynamically join/drop a given multicast group. We have defined apropriate join/drop procedures in order to keep tree feasibility and low cost. We have compared various performance tradeoffs in a simulation study.

We are currently investigating appropriate algorithms to build Steiner Trees with multiple constraints (say bandwidth, delay). The study of the evolution of such trees under join/drop operations is also topic for future research.

References

[Cavendish98] D. Cavendish, A. Fei, M. Gerla, R. Rom "On the Construction of Low Cost Multicast Trees with Bandwidth Reservation," *To appear at High Performance Computing and Networking, HPCN Europe'98.*

[Cavendish97] D. Cavendish, A. Fei, M. Gerla, R. Rom "On the Construction of Low Cost Multicast Trees with Bandwidth Reservation," *UCLA Technical Report Number 970043*, December 1997.

[Bauer96] F. Bauer and A. Varma, "ARIES: A Rearrangeable Inexpensive Edge-based On-line Steiner Algorithm", *Proceedings of IEEE INFOCOM '96*, San Francisco, 1996, 369-376.

[Deering96] S. Deering et al., "The PIM Architecture for Wide-area Multicast Routing", *IEEE/ACM Transactions on Networking*, vol. 4, pp.153-162, April 1996.

[Frank85] A. Frank, L. Wittie, and A. Bernstein, "Multicast Communication in Network Computers", *IEEE Software*, Vol. 2, No. 3, pp 49-61, 1985.

[Goemans93] H. N. Gabow, M. X. Goemans and D. P. Williamson, "An Efficient Approximation Algorithm for the Survivable Network Design Problem", *In Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pp. 57-74, 1993.

[Jiang91] X. Jiang, "Path Finding Algorithms for Broadband Multicast", *Third Intl. Conf. on High Speed Networking*, pp. 153-164, North-Holland, 1991.

[Makki95] K. Makki, N. Pissinou, O. Frieder, “On Multicast in a Communication Network”, *Proceedings of the ACM SIGCOMM*, pp. 647-654, 1995.

[OPENET95] I. Cidon, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M. Sidi, “The OPENET Architecture”, *Sun Microsystems Laboratories Technical Report SMLI TR-95-37*, 1995.

[Ravi95] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, H. B. Hunt, ”Bicriteria Network Design Problems”, *Proceedings of 22nd International Colloquium on Automata Languages and Programming*, LNCS 944, pp. 487-498, 1995.

[Smith83] V. J. Rayward-Smith, “The Computation of Nearly Minimal Steiner Trees in Graphs”, *Intl. J. Math. Educ. Sci. Tech.*, Vol. 14(1), pp. 15-23, 1983.

[Wang95] Z. Wang and J. Crowcroft, “Bandwidth-Delay Based Routing Algorithms”, *GLOBECOM’95*, pp. 2129-2133.

[Waxman88] B. Waxman, “Routing of Multiple Connections”, *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp.1617-1622, Dec. 1988.

[Westbrook93] J. Westbrook and D. Yan, “Greedy Algorithms for the On-line Steiner Tree and Generalized Steiner Problems”, *in Algorithms and data structures. Third Workshop, WADS’93*, Montreal, Quebec, Canada, Aug. 1993, pp. 621-633.