Dynamic Routing of Bandwidth Guaranteed Multicasts with Failure Backup

Murali Kodialam T.V. Lakshman
Bell Labs, Lucent Technologies
Holmdel, NJ 07733
muralik@lucent.com, lakshman@lucent.com

Abstract

This paper presents a new algorithm for dynamic routing of bandwidth guaranteed multicast tunnels with failure backup. The multicast routing problem arises in many contexts such as the routing of point-to-multipoint label switched paths in Multi-Protocol Label Switched (MPLS) Networks, and the provision of bandwidth guaranteed services under the "hose" model [4]. Failure backup implies that when a multicast tree is set-up alternate backup paths be also set-up so that the multicast is unaffected by single link or node failures. For dynamic routing, the multicast routing requests arrive one-by-one and there is no a priori knowledge regarding future requests. We believe that this is the first paper that addresses the issue of multicast routing with failure backup. Each multicast request consists of a source s, a set of receivers R, and a bandwidth requirement b. Offline multicast routing algorithms cannot be used since they require a priori knowledge of all multicast tunnel requests that are to be routed. The newly developed algorithm is an on-line algorithm that generates a reserved-bandwidth multicast tree with additional backup links that make the multicast tree resilient to single element failures in the network. It shares backup bandwidth when possible and only uses link usage information obtainable in a distributed manner.

1 Introduction

We consider the problem of dynamically routing bandwidth guaranteed multicasts with failure backup. For dynamic routing, we must be able to route multicast set-up requests, from a specified source to a specified set of receivers, that are given one-by-one with future requests being unknown. Also, we must be able to handle dynamic receiver joins and leaves from an already established multicast tree. Failure resilient multicast, for the purposes of this paper, implies the ability to locally bypass a single failed node or link thereby restoring connectivity of the multicast tree. Local switching permits fast recovery from failures in comparison to schemes that require rerouting from the source or from some point further away from the failed element. For efficiency, we would like different active multicast connections to share the backup reservations. This sharing of backup bandwidth is explained in detail in Section 1.4. Also, we would like the multicast routing algorithm to only use information that is obtainable in a distributed manner precluding the use of information such as the routes of all previously routed multicasts in the network. The contribution of this paper is an algorithm for on-line routing of bandwidth guaranteed multicasts that efficiently shares backup path bandwidth while being resilient to single element failures. It also only uses network usage information that can be obtained in a distributed fashion. Before describing the key ideas of the algorithm, we discuss the restoration scheme considered and the motivation for the assumptions regarding information obtainable from the network.

1.1 Restoration Model

Backup for connections in a network can be built so that there is *end-to-end restoration* or *local restoration*. In the case of end-to-end restoration, the idea is to provide a backup path from the source to the destinations for each multicast request. This backup path is (link or node) disjoint from the primary multicast tree. However the drawback with this approach is that when there is a link or node failure, this information has to propagate back to the sources of all multicasts which traverse this link. The sources in turn switch the multicast from the primary to the secondary path. Note that the link failure information has to be propagated to all nodes in the network. The end-to-end restoration in a multicast tree is illustrated in Figure 1. The source node for the multicast is node s and the destination node set $D = \{D_1, D_2, D_3, D_4\}$. The dark lines in the figure represents the links that are used on the active path. The dashed lines represents the backup paths. Note that there are four end-to-end backup paths one corresponding to each desti-

Proceedings of the 10 th IEEE International Conference on Network Protocols (ICNP'02)
1092-1648/02 $17.00 © 2002 IEEE

nation.

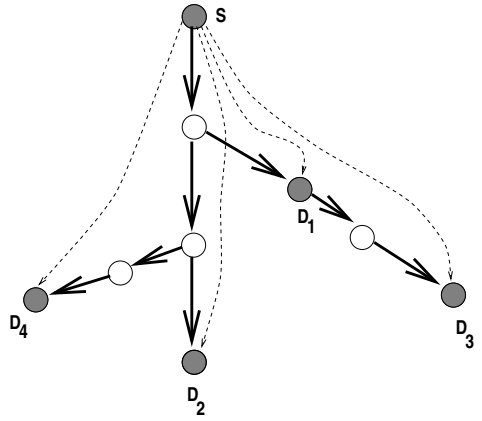

Figure 1. Paths for End-to-End Restoration

The time taken for node/link failure information propagation to the source (that is needed for end-to-end restoration) may not be acceptable for many applications. Therefore we consider a local restoration model. In the case of local restoration, the backup paths are provided locally and therefore failure information does not have to propagate back to the source before connections are switched to the backup path. The idea of local restoration is illustrated in Figure 2. The source and the destination set for the multicast is the same as in Figure 1. The active multicast tree is shown in dark lines and the backup paths are shown in dashed lines. Note that if any node or link fails in the active path, then there is connectivity on the backup path. In Section 7 we outline how to reserve bandwidth on the backup path. The advantage of having local restoration is outlined next.

1.2 Failure Modes

The failure mode that we consider in this paper is protection against *single element (node or link) failures*. Algorithms for a less stringent version where we protect against single link failures can also be easily derived by modifying the algorithms in this paper. When a link fails, the router or switch interfaces at both ends of this link detect failure. Traffic is locally switched, from the upstream end of the failure, along the backup path as shown in Figure 2. The information propagation in the case of a single link failure is shown in Figure 3. The backup path that is activated is shown in the dark dashed line in the figure.

When a node fails all the link interfaces at that node fail. Therefore all links that are incident on the node fail and requests are routed across the failed node. This is illustrated in Figure 3. The relevant failed links are shown in light dotted lines and the backup path that is activated is shown as a dark dashed line.

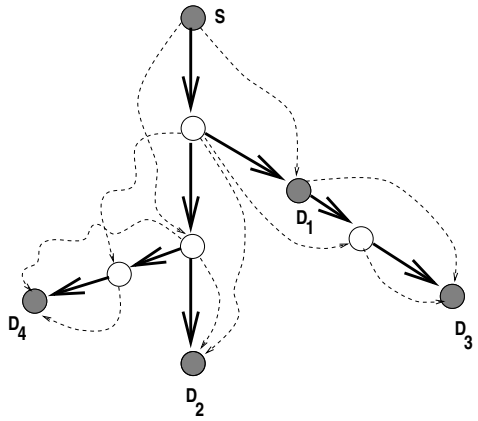

Figure 2. Local Restoration: Primary and Bypass Paths for Single Element Failures

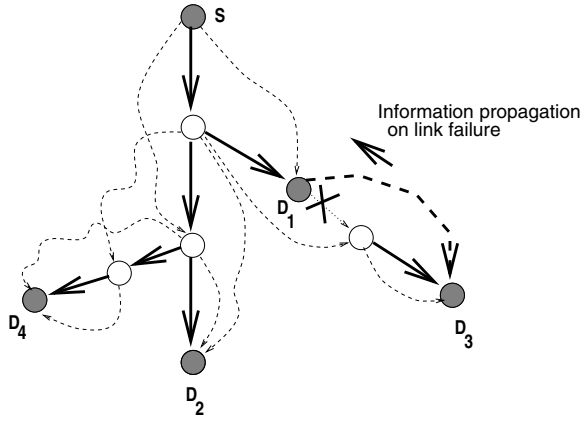

Figure 3. Restoration Path on Link Failure

If a node is a branch point then failure of that node will lead to the activation of multiple backup paths. This is illustrated in Figure 5 where the failed node is a branch point. Note that two backup paths are activated on this node's failure.

Note that we cannot provide backup to the failure of the source node. Taking care of single node failures almost handles all the link failures also except for the last link on the forward path where a backup path is needed to protect against the failure of this link.

1.3 Backup Paths

For the single link failure case, the backup path for a link (i, j) can be any path connecting nodes i and j that does not

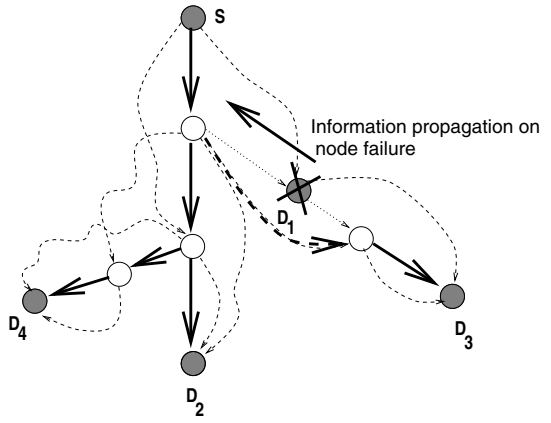

Figure 4. Restoration Path on Non-Branch-Point Node Failure-Single Path Activation

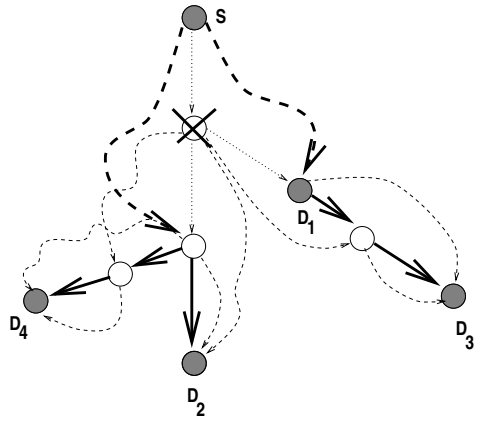

Figure 5. Restoration Path on Branch-Point Node Failure-Multiple Paths Activation

include link (i, j). This backup path for link (i, j) can traverse any link, including links on the primary or backup paths (other than those being considered for link (i, j)) for the current request being set-up. For the single element failure case, the backup path for the failure of node k involves doing the following: First determine the links (j, k) and (k, l) in the primary path. If node k fails, then it will result in the failure of all links incident on node k, in particular link (j, k). Therefore the failure will be detected at node j and if there is an alternate path from node j to node l (or some other node between l and the destination t) then node j can divert traffic onto this node k bypass or backup path. Note that the backup path for the failure of node k has to aviod all links incident on node k.

1.4 Sharing Backup Links

Because of the need to provide bandwidth guarantees, capacity on the primary path cannot be shared. However, even with bandwidth guarantees it is possible to share capacity on links used for backup paths. This is because the backup link capacity is used only upon failure. This is explained in detail below. Capacity in the backup path can be shared at two levels. *Inter-request sharing* refers to the case of sharing the backup reservations belonging to different requests that do not share links along the primary path. Figure 6 illustrates inter-request sharing. The routing of two multicast requests is shown in the figure. Request 1 has source node s_1 and destination nodes $\{D_1, D_2\}$ and request 2 has source node s_2 with destination node D_2. The backup paths for the two requests are shown in dotted lines. Note that the two links indicated in Figure 6 are shared backup links for the two requests. Therefore, for example, if boths these requests are for one unit of bandwidth, then on the two shared links only one unit of bandwidth will be reserved. The second kind of sharing is intra-request sharing. Since the restoration is local in nature, there is a backup path to protect against the failure of each node/link in the current multicast tree, Since we assume that only one element can fail at any given time, the backup paths for different node failures in the current multicast tree can be shared, This kind of sharing of backup bandwidth between the different backup paths in the current request is termed *Intra-request* sharing. Figure 7 illustrates intra-request sharing. In Figure 7 note

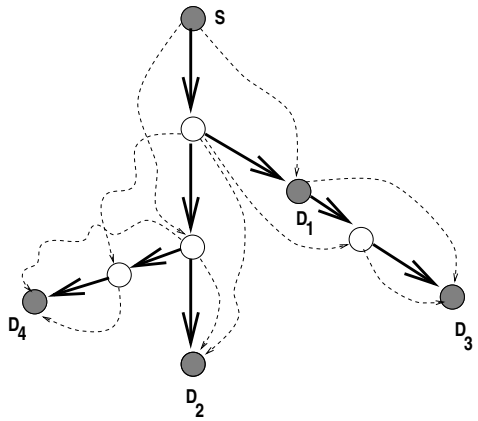

Figure 6. Inter-request Sharing

that link (a, D_2) is used to backup links (S, b) and (b, D_2) and also node failure b. This in turn means that backup capacity is shared on this link. The algorithms that we outline exploit both inter request and intra request sharing in order to minimize the amount of bandwidth consumed.

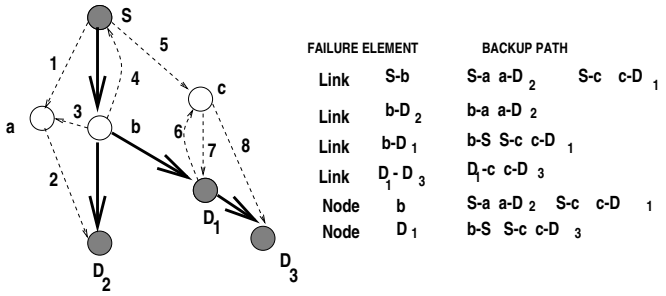

Figure 7. Intra-request Sharing

1.5 Information Model

The amount of sharing that can be achieved is a function of the amount of information that is shared between the nodes in the network. We consider three different information models as in [7]. In order to share the backup bandwidth, the best possible scenario is if every node knows how each request in the network is routed. This is the *complete information case*. This is a reasonable assumption if the routing is done in a centralized manner. However, if routing is done in a decentralized manner this assumption is not reasonable since the entire path (forward and backup) of every request has to be communicated to every node in the network. We however assume that some link state flooding mechanism exists in the network and that for each link the total amount of bandwidth on that link that is used by primary paths, the total amount of bandwidth that is used by backup paths, and the residual capacity on that link is propagated through the network. This is termed *partial information case*. It is easy to propagate this partial information to all nodes in the network by making minor modifications to existing link state unicast routing protocols like OSPF. In the *minimal information case* we assume that only the residual bandwidth at each link is known. In this case, there can be no inter-request sharing. Intra-request sharing is still possible. The minimal information case serves as a benchmark for comparison purposes.

2 Problem Definition

We consider a network of n nodes (switches/routers) and m directed links. Each multicast request consists of a source node s, a set of destination or receiver nodes D and a bandwidth b. The objective is to route b units of bandwidth from s to each $d \in D$ and determine local backup paths so that the active path can survive single element failures in the network. Each multicast request is assumed to arrive at the source node which then determines the active multicast tree along with associated backup paths. Multicast requests arrive one at a time and each request is either fully accepted, partially accepted or is rejected. In the case of partial acceptance where only some subset of the receiver set D can be reached, we route as many as possible. Note that when a multicast request is routed, all links on its active path will reserve b units of bandwidth for this request. We denote the bandwidth request for request k to be b_k. The paper explores the amount of sharing that can be achieved under three different information scenarios. Towards this end we define the following notation. Let A_{ij} represent the set of requests that use link (i, j) in its primary path, and B_{ij}^{uv} represent the set of requests that use link (i, j) as a backup for link (u, v) in the primary path for that request. In other words, $k \in B_{ij}^{uv}$ if

- Link (u, v) is on the active tree of request k.
- Link (i, j) is used as part of the local backup path for link (u, v) for the request k.

Let $\delta_{ij}^{uv} = \sum_{k \in B_{ij}^{uv}} b_k$. Let F_{ij} represent the total amount of bandwidth reserved for the requests that use link (i, j) on the primary path. Let G_{ij} represent the total amount of bandwidth reserved by links that use link (i, j) on its backup path. Let $R_{ij} = C_{ij} - F_{ij} - G_{ij}$ represent the residual bandwidth of link (i, j). In the case of *minimal information model* we assume that the only information known is R_{ij} for each link (i, j) in the network. Note that only the residual bandwidth is known, then it is not possible to share any backup capacity between different requests. It may still be possible to do intra-request bandwidth sharing. This minimal information based sharing case serves as a benchmark to compare the other cases where inter-request sharing is possible. In the case of the *complete information case* we assume that each node knows the sets A_{ij} for all links (i, j) and the set B_{ij}^{uv} for all link pairs $(i, j), (u, v)$, In the *partial information case* we assume that each node knows the value of F_{ij}, G_{ij} and R_{ij} for all links (i, j) in the network. Since we do not have any knowledge of the requests that will arrive in the future, the objective of the routing algorithm is to determine the active and backup path for the current tunnel request that "optimizes" the use of network infrastructure. One reasonable objective then is to *minimize the sum of the bandwidths that is used by the active and the backup paths*. In the case where no restoration is needed, i.e., we just have to determine the active path to all the destination node, this objective leads to the minimum directed Steiner tree problem. As stated earlier, the amount of sharing that can be achieved on the backup path is a function of how much information is known about the requests that are still currently active in the network. This objective of minimizing the total amount of bandwidth consumed by both the active and the backup paths can be formulated as an integer programming problem. However, the problem is NP-hard and we

use heuristics to sovle the problem. We now outline the key ideas in the design of the algorithm to solve this problem.

3 Key Ideas in the Algorithm Design

Some of the restoration problems posed in the previous section can be formulated as integer programming problems. Apart from using general purpose solvers there does not seem to be any easy way to solve thse integer programming problems. Therefore we design a heuristic algorithm to solve the multicast routing problem. In this section we outline the key ideas in algorithm design in three steps.

- In the first step we consider the dynamic multicast routing problem without backup. This problem is NP-hard and we adapt a Steiner tree heuristic that has good performance and more importantly lends itself very well to including the backup computation.
- In the second step we show how to include inter-request sharing in the Steiner tree heuristic.
- In the third step we show how to incorporate intra-request sharing.

3.1 Steiner Tree Computation

Without restorability, the multicast routing problem is a directed Steiner Tree problem. The *directed* Steiner tree problem is defined as follows: given a directed graph $G = (N, E)$, a specified source node $s \in N$, and a set of receivers $D \subseteq N$, the objective is to find the minimum cost arborescence rooted at s and spanning all the nodes in D, i.e., source s should have a path to every vertex in R. Apart from multicast routing, the Steiner tree problem has several other applications to network design and network routing. For a general survey on the use of Steiner tree problems in networks, see Winter [11]. The directed Steiner tree problem is $\mathcal{NP}$-complete even when the graph is induced by points in the plane [5], and is $\mathcal{MAXSNP}$-hard for general graphs. Therefore, several heuristics have been designed for the solution of this problem [1, 11]. We use the Nearest Neighbor First (NNF) heuristic developed in [8] as a subroutine to develop algorithms for the construction of multicast trees with local restoration. Our heuristic uses several iterations of Dijkstra's shortest path algorithm to construct the multicast tree. In the description of the algorithm, we assume that all executions of Dijkstra's algorithm go backwards from the destination to the source. The reason for this will be made clear in the next two sections.
The NNF Steiner tree heuristic works as follows: The set of active destinations D' is set to D. We run Dijkstra's shortest path algorithm from each destination node $d \in D'$ to the source. We pick the destination node d' that is closest to the source. The cost of all the links on the shortest path from s to d' is now set to zero and the set of active destinations is set to $D' \backslash d'$. The reasons for setting the costs to zero are the following:

1. Since these edges have already been added to the Steiner tree, the incremental cost of using them to reach other nodes is zero.
2. Setting these edge costs to zero encourages future runs of Dijkstra's algorithm to use them. This is consistent with the fact that increased sharing of paths to two different receivers leads to lower cost trees.

This process is repeated until the set of active destinations is empty. At any intermediate point in the algorithm, the solution will be a directed tree rooted at s and the leaves of the tree will be set of destinations which are already routed. Let $\tilde{D}$ represent the set of destinations for which paths have already been found. Let $T(\tilde{D})$ represent the multicast tree upto this point. When we attempt to find a path for the next destination, note that it is necessary to find the shortest path from that destination to $T(\tilde{D})$. The shortest path from a destination node to a given tree can be determined by executing Dijkstras's algorithm from the destination until some node in $T(\tilde{D})$ is permanently labelled.

3.2 Inter-Demand Sharing

The Steiner tree heuristic outlined above lends itself well to the inclusion of locally restorable paths. Consider some intermediate stage in the NNF algorithm. We now consider how to include the costs due to local backup path establishment. When running Dijkstra's algorithm, to find the cost of including link (i, j) in the primary path, we have to determine the cost of finding a backup from node i to the successor of node j without using any of the links incident on node j. Of course when the algorithm is run backwards from the destination, the successors of all the nodes that are permanently labeled by Dijkstra's algorithm are already known since a path as been established from that node to the destination. Therefore when running Dijkstra's algorithm, for any permanently labelled node j, let $S(j)$ represent the set of nodes from node j to the destination node currently under consideration. When node j fails, all the links incident on that node fails. Therefore the cost of the backup has to account for all the links failing simultaneously. When computing the cost of using link (i, j) in the primary path, we have to consider the cost of backing up requests that use link (j, l) for $l \in V$. Therefore the cost of links have to be modified as follows. Let θ_{ij}^{uv} represent the amount of backup bandwidth that has to be reserved on link (u, v) when link (i, j) is used in the active path. Recall that δ_{ij}^{uv} represents the amount of flow on backup link (u, v) if link

(i, j) goes down. In the complete information case

$$\theta_{ij}^{uv} = \begin{cases} 0 \\ \text{if } \sum_{(j,k)\in E} \delta_{jk}^{uv} + b \leq G_{uv} \text{ and } j \neq v \\ \\ \sum_{(j,k)\in E} \delta_{jk}^{uv} + b - G_{uv} \\ \text{if } \sum_{(j,k)\in E} \delta_{ij}^{uv} + b > G_{uv} \\ and \;\; \mathrm{R}_{uv} \geq \sum_{(j,k)\in E} \delta_{jk}^{uv} + b - G_{uv} \text{ and } j \neq v \\ \infty \quad \text{Otherwise} \end{cases}$$

If we consider the case where only partial information is available then the cost of link (u, v) is given by the equation below:

$$\theta_{ij}^{uv} = \begin{cases} 0 \\ \text{if} \quad \sum_{(j,k)\in E} F_{jk} + b \leq G_{uv} \text{ and } j \neq v \\ \\ \sum_{(j,k)\in E} F_{jk} + b - G_{uv} \\ \text{if } \sum_{(j,k)\in E} F_{jk} + b > G_{uv} \\ and \;\; \mathrm{R}_{uv} \geq \sum_{(j,k)\in E} F_{jk} + b - G_{uv} \text{ and } j \neq v \\ \infty \quad \text{Otherwise} \end{cases}$$

Therefore the next algorithmic idea is the following: *The cost of creating backup paths can be accounted for by modifying the cost of the links in the Steiner Tree computation.* Let $L_\theta(i, j)$ represent the length of the shortest path from node i to some node in $S(j)$ using a cost of θ_{ij}^{uv} on link (u, v). This can be computed easily by running Dijkstra's algorithm starting at node i until some node in $S(j)$ is labelled permanently. This permanent label of the node in $S(j)$, in fact is $L_\theta(i, j)$. There are two cases to consider now. If i is not in the current multicast tree then, as shown in Figure 10, the local backup is one path. The cost of this path is given by $L_\theta(i, j)$. In case i is in the current multicast tree, then an additional path has to be added to account for the failure of node i. This is illustrated in Figure 8. If node i is in the multicast tree, let $P(i)$ represent the predecessor to node i in the multicast tree. In other words $P(i)$ is the last node before node i on the path from the source node s to node i.

Therefore the cost of adding link (i, j) is given by

$$c_{ij} = \begin{cases} d + L_\theta(i, j) & \text{if } i \text{ not in multicast tree} \\ d + L_\theta(i, j) + L_\theta(P(i), j) & \text{if } i \text{ in multicast tree} \end{cases}$$

Note that the value of θ_{ij}^{uv} changes after each destination is routed and therefore the value of c_{ij} has to be computed dynamically.

3.3 Intra-Demand Sharing

Consider two links (i, j) and (k, l) which are both in the current multicast tree. Intra-request sharing of bandwidth occurs when the link (i, j) uses link (u, v) for a backup and reserves a bandwidth of w on link (u, v). If link (k, l) on the primary path wants to use link (u, v) for its backup then, in addition to any inter-request sharing it can use the already reserved bandwidth of w for free. At some intermediate point in the execution of the algorithm, all the backup reserved for the current tree comes for free. In addition since the shortest path algorithm is to be run backward from the destination all the backup that is reserved on the path from the current node to the current destination d' is also free. Let γ_{uv} represent total amount of backup reservations made on link (u, v) by the current multicast tree so far. We introduce a m-vector λ^u corresponding to node u in the network to accumulate the amount of reservation made by the current running of Dijkstra's algorithm. (Recall that the number of links in the network is m.) λ_{ij}^u represents the amount of bandwidth reserved by the current request for all the backup paths for all the links leading from node u to the current destination d'. This bandwidth reservation for the current request can be used to back up for free, the links from u to

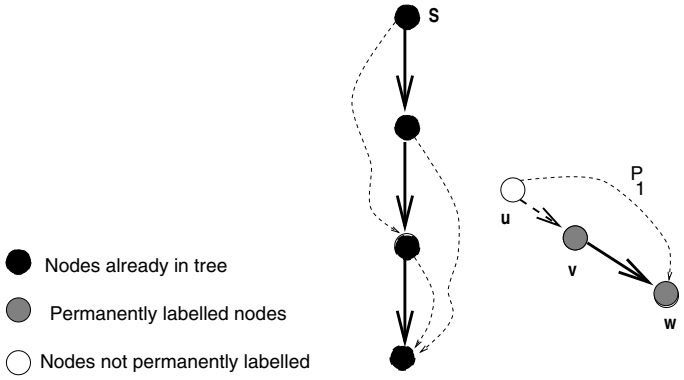

Figure 8. Backup Cost Determination if i is not in the Multicast Tree

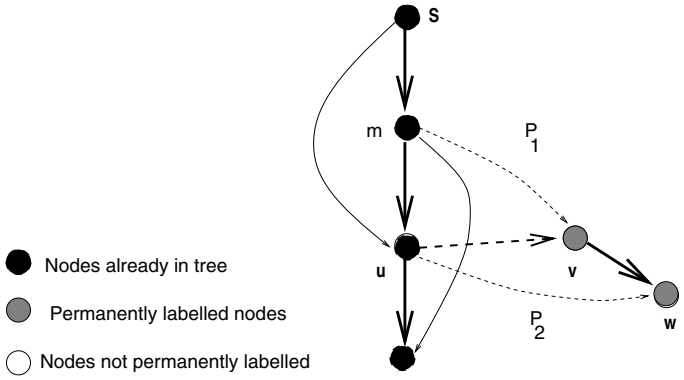

Figure 9. Backup Cost Determination if i is in the Multicast Tree

the source s that are yet to be determined. Consider the case where the backup path for link (k, j) is being determined. Assume that the shortest path is being determined in the backward direction from node j to node k. The m-vector λ^j represents the reservation made for this request for all the backup paths from node j to the destination. This path is known since there is a unique path from node j to the destination in the shortest path tree. We now outline how to adjust the cost of link (m, n) to account for intra-request sharing. Define

$$\kappa_{mn} = \sum_{(j,k)\in E} F_{kj} + d - G_{mn} - \gamma_{mn} - \lambda^j_{mn}.$$

Then the cost of link (m, n) when determining the shortest backup path is given by

$$l_{mn} = \begin{cases} 0 & \text{if } \kappa_{mn} \leq 0 \text{ and } n \neq j \\ \kappa_{mn} & \text{if } 0 \leq \kappa_{mn} \leq d \text{ and } R_{mn} \geq \kappa_{mn} \text{ and } n \neq j \\ \infty & \text{Otherwise} \end{cases}$$

The cost in the case of the complete information case can algo be modified similarly. Therefore the above procedure gives us a method for accounting for the intra-request sharing. This gives the next algorithm design idea:
Maintaining the link length vector at each node that gives us the amount of bandwidth reserved for the current request for backing up all links from the the given node to the destination can be used to account for intra-request sharing.

NEAREST_NEIGHBOR_FIRST(s, D)

- INITIALIZATION
  1: $X = D, \quad T = \emptyset$
- ITERATIVE STEP
  2: MIN = ∞
  3: For all $a \in X$
  $C =$ PATH_COST (a, T)
  if ($C <$ MIN)
  MIN $= C$
  $f = a$
  4: $T = T \cup$ PATH(f) $\quad X = X \backslash f \quad$ If $X \neq \emptyset$ Go to Step 2
- TERMINATION
  5: Exit.

4 Formal Description of the Algorithm

Let s represent the source of the multicast request and D the destination set. The main routine is NEAREST_NEIGHBOR_FIRST(s, D) which iterates through the set of destinations one at a time. On each iteration, it picks the "closest" destination to the current multicast tree and adds the path from the tree to that destination to the current tree. The iterations are repeated until a path is found from the source to all the destinations. The routine to find the distance between the current destination node a and the current (partial) multicast tree T is determined in the routine PATH_COST(a, T). Note that this routine runs Dijkstra's algorithm backwards from the destination a until some node in T is labelled permanently. The routine maintains the set of successor nodes for a given node j in the set SUCCSET(j). This set represents the set of all nodes from node j to the current destination a. The algorithm also maintains the predecessor node for node $j \in T$ as PRED(j) which is the node immediately before j on the path from s to j. The SUCCSET() and PRED() are used to determine the cost of backing up a given link. Unlike Dijkstra's algorithm where the cost of a link is fixed, in this case the cost of link (c, j) is determined dynamically by the routine COMPUTE_COST(c, j). The COMPUTE_COST() routine shown here is for the partial information case. Note that there are two cases to consider here as shown in Figures 4 and 5. If node c is not part of the current multicast tree T then one backup path has to be determined. If node $c \in T$ then two backup paths have to be determined. We use Dijkstra's algorithm for SHORTEST_PATH(). Note that in this description we have not taken into consideration intra-request sharing. This can be incorporated into the COMPUTE_COST() routine by keeping track of bandwidth that is already reserved for the current multicast request. The algorithms presented in the next section incorporate intra-request sharing. However in order to keep the clarity of presentation, intra-request sharing as well as the backup path for the last link in the active path is left out from the high level description in this section.

COMPUTE_COST(j, c)

1: Set link costs θ^{uv}_{ij}.

2: Let $v =$SHORTEST_PATH(j,SUCCSET$(j)\backslash\{c\}$).

3: If $c \in N\backslash T$, then Return(v).

3: If $c \in T$, then $v = v+$SHORTEST_PATH(PRED (j), SUCCSET(j)).

4: Return(v)

PATH_COST(a, T)

- INITIALIZATION
  - 1: $P = a$, $c = a$, $Q = N \backslash a$ and SUCCSET$(a) = \emptyset$.
  - 2: $L(a) = 0$ and $L(i) = \infty$ if $i \neq a$.
- ITERATIVE STEP
  - 3: if $c \in T$ go to Step 6.

    For each j such that $(j, c) \in E$
    d_{jc} =COMPUTE_COST (j, c)
    if $(L(j) \geq d_{jc} + L(c))$
    $L(j) = L(c) + d_{jc}$
    PRED$(c) = j$
    SUCCSET$(j) =$
    SUCCSET$(c) \cup \{c\}$

  - 4: $c =$ Arg $\min_{j \in Q} L(j), P = P \cup \{c\}, Q = Q \backslash c$.
  - 5: Go to Step 3.
- TERMINATION
  - 6: Set PATH(a) to the shortest path from c to a.

5 Experimental Results

In this paper, we developed a heuristic algorithm for online routing of multicast requests with failure backup. It is actually surprisingly difficult to even formulate this problem exactly as an integer programming problem let alone solve it. Therefore the main objective of this experimental section is to compare the performance of the heuristic algorithm with the three different information models. The experimental set up is the following: We performed experiments on two networks

- Network 1: 15 nodes and 56 arcs.
- Network 2: 70 nodes and 206 arcs.

he primary objective of the experiments is to determine the amount of sharing that can be achieved in the the different information models. Therefore we test the following scenarios:

1. Multicast with minimal information (MMI).
2. Multicast with partial information (MPI).
3. Multicast with complete information (MCI).

We measure the effectiveness of the different schemes by doing the following experiments. Multicast requests arrive one at a time to the network. The source node is picked randomly from the set of nodes. The rest of the nodes have a ten percent chance of being in the multicast tree in Network 1 and a five percent chance in Network 2. (If there are no destinations selected then the request is not used for the experiment.) The bandwidth requested is uniformly distributed between 1 and 6 units. The multicast request is routed. In the minimal information case, only the residual bandwidth of each link is known and only intra-request sharing is possible. In the case of partial information, the forward flow, backup reservation and the residual bandwidth of each link is known to all the nodes and therefore intra-request as well as some inter-request sharing is possible. In the complete information case all the actual path taken by all requests is known to all nodes and therefore intra-request as well as complete inter-request sharing is possible. The objective of the experiments is to determine how much bandwidth is saved by this sharing. For experiments with this objective, the capacity of the links in the network can be set to infinity.

We load a fixed number (100) of requests onto the network and measure the following two quantities:

1. The *total amount of bandwidth consumed* by the requests (for both active and backup).
2. The ratio of the amount of bandwidth consumed for the backup to the bandwidth consumed on the active path, *backup to active bandwidth ratio*.

Figure 10 shows the evolution of the backup to active bandwidth ratio as requests are loaded onto the network. Note that as more requests are loaded onto the network, there will be more sharing and leads to better backup bandwidth efficiency. However the backup to active ratio reaches some steady state value which depends on the information model used. Note that the ratio is about 0.5 for MCI, 1.5 for MPI and 2.5 for MMI for Network 1 and 0.3, 2 and 4 for MCI, MPI and MMI for Network 2. Therefore there is significant benefit to getting partial information over minimal information. The actual amount of gain depends on the structure of the network as well as the number of destinations per multicast request.

Figures 11 and 13 show the total amount of bandwidth consumed for 100 multicast requests for the different information models for five different experiments. Note from the figures that the partial information models shows significant bandwidth reduction for the backup for some minimal extra information over the no-sharing case. Figure 12 and 14 give the backup to active bandwidth ratio for the same five experiments. There seems to be a significant benefit to having partial information.

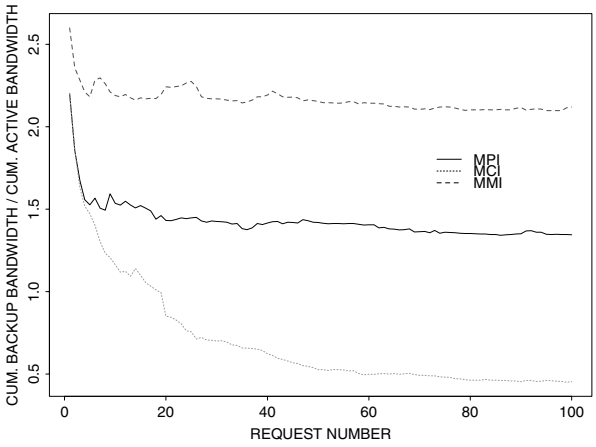

Figure 10. Network 1: Ratio of Backup to Active Bandwidth Consumed versus Request Number

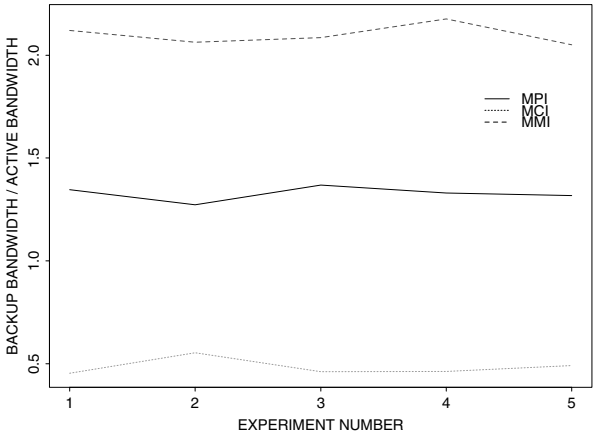

Figure 11. Network 1: Total Bandwidth Consumed for Routing 100 Multicast Requests

References

[1] P. Berman and V. Ramaiyer. Improved approximation algorithms for the Steiner tree problem. *Journal of Algorithms*, 17:381–408, 1994.

[2] R. Callon, N. Feldman, A. Fredette, G. Swallow, A. Viswanathan. A Framework for Multiprotocol Label Switching, *Internet Draft draft-ietf-mpls-framework-03.txt*, June 1999.

[3] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li. Approximation Algorithms for Directed Steiner Problems. *Symposium on Discrete Algorithms (SODA)*, 1998.

[4] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan. A Flexible Model for Resource Management in Virtual Private Networks. *Proceedings of ACM SIGCOMM 1999*, pp. 95-104, September 1999.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W.H. Freeman and Company, New York, 1979.

[6] K. Kar, M. Kodialam, T. V. Lakshman, "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications", *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 12, December 2000.

[7] M. Kodialam, T. V. Lakshman. Dynamic Routing of Bandwidth Guaranteed Tunnels with Restoration, *INFOCOM 2000*, Tel-Aviv, Israel.

[8] M.Kodialam, T. V. Lakshman and S. Sengupta, Multicast Routing With Bandwidth Guarantees: A New Approach Using Multicast Network Flow, *SIGMETRICS 2000.*, San Jose, CA.

[9] D. Ooms, W. Livens, B. Sales, M. Ramalho, A. Acharya, F. Griffoul, F. Ansari. Framework for IP Multicast in MPLS. *MPLS Working Group Internet Draft*, June 1999.

[10] B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal of Selected Areas in Communications*, pp. 1617–1622, 1988.

[11] P. Winter. Steiner problem in networks: a survey, *Networks*, 17:129–167, 1987.

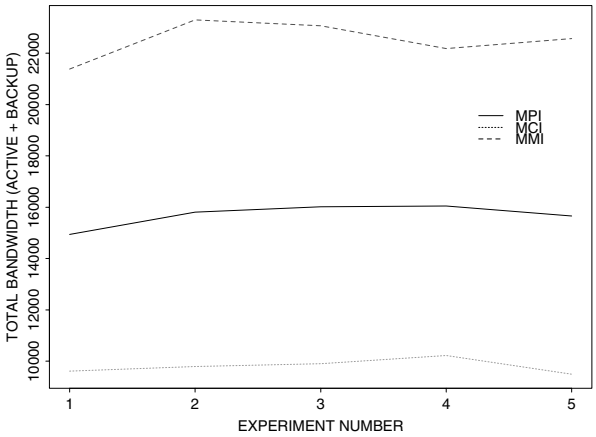

Figure 12. Network 2: Total Bandwidth Consumed for Routing 100 Multicast Requests

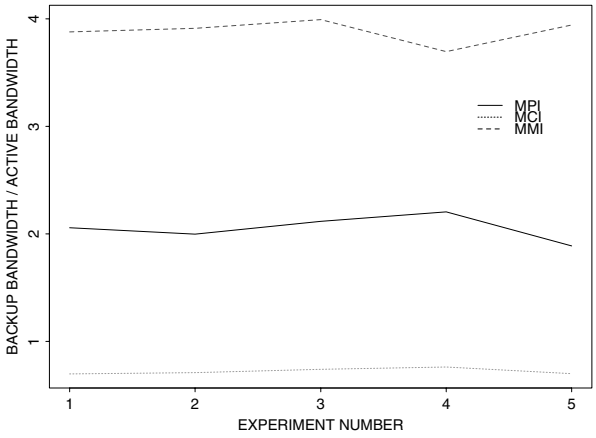

Figure 13. Network 2: Ratio of Backup to Active Bandwidth for 100 Multicast Requests